

# Package ‘smoppix’

July 4, 2025

**Type** Package

**Title** Analyze Single Molecule Spatial Omics Data Using the Probabilistic Index

**Version** 1.1.4

**Description** Test for univariate and bivariate spatial patterns in spatial omics data with single-molecule resolution. The tests implemented allow for analysis of nested designs and are automatically calibrated to different biological specimens. Tests for aggregation, colocalization, gradients and vicinity to cell edge or centroid are provided.

**License** GPL-2

**Encoding** UTF-8

**Imports** spatstat.geom(>= 3.2.0), spatstat.random, methods, BiocParallel, SummarizedExperiment, SpatialExperiment, scam, Rdpack, stats, utils, extra(>= 1.0.11), Matrix, spatstat.model, openxlsx

**Suggests** testthat, rmarkdown, knitr, DropletUtils, polyCub, RImageJROI, sp, ape, htmltools, funkycells, glmnet, doParallel

**RdMacros** Rdpack

**RoxygenNote** 7.3.2

**biocViews** Transcriptomics, Spatial, SingleCell

**Depends** R (>= 4.4.0)

**VignetteBuilder** knitr

**BugReports** <https://github.com/sthawinke/smoppix/issues>

**URL** <https://github.com/sthawinke/smoppix>

**LinkingTo** Rcpp

**git\_url** <https://git.bioconductor.org/packages/smoppix>

**git\_branch** devel

**git\_last\_commit** fc1f725

**git\_last\_commit\_date** 2025-06-03

**Repository** Bioconductor 3.22

**Date/Publication** 2025-07-03

**Author** Stijn Hawinkel [cre, aut] (ORCID:  
<<https://orcid.org/0000-0002-4501-5180>>)

**Maintainer** Stijn Hawinkel <[stijn.hawinkel@psb.ugent.be](mailto:stijn.hawinkel@psb.ugent.be)>

## Contents

addCell . . . . .	3
addDesign . . . . .	5
addNuclei . . . . .	5
addTabObs . . . . .	7
buildDataFrame . . . . .	8
buildFormula . . . . .	9
buildHyperFrame . . . . .	10
buildMoransIDataFrame . . . . .	11
buildMoransIWeightMat . . . . .	12
calcIndividualPIs . . . . .	12
calcNNPI . . . . .	14
calcWindowDistPI . . . . .	14
centerNumeric . . . . .	15
checkFeatures . . . . .	16
checkPi . . . . .	16
constructDesignVars . . . . .	17
convertToOwins . . . . .	17
crossdistWrapper . . . . .	18
Eng . . . . .	18
estGradients . . . . .	19
estPis . . . . .	21
evalWeightFunction . . . . .	24
extractResults . . . . .	25
findEcdfsCell . . . . .	25
findOverlap . . . . .	26
fitGradient . . . . .	27
fitLMMs . . . . .	28
fitPiModel . . . . .	30
getCoordsMat . . . . .	31
getDesignVars . . . . .	31
getElement . . . . .	32
getFeatures . . . . .	32
getGp . . . . .	33
getHypFrame . . . . .	33
loadBalanceBplapply . . . . .	34
makeDesignVar . . . . .	34
makePairs . . . . .	35
moransI . . . . .	35
named.contr.sum . . . . .	36
nestRandom . . . . .	36
plotCells . . . . .	37
plotExplore . . . . .	38
plotTopResults . . . . .	40
plotWf . . . . .	41
smoppix . . . . .	42
sortGp . . . . .	42
splitWindow . . . . .	43
subSampleP . . . . .	43
sund . . . . .	44
writeToXlsx . . . . .	44

<i>addCell</i>	3
Yang . . . . .	45
<b>Index</b>	<b>46</b>

---

<i>addCell</i>	<i>Add cell boundaries and event-wise cell identifiers to a hyperframe.</i>
----------------	-----------------------------------------------------------------------------

---

## Description

Add the list of the cells and their centroids in the hyperframe, check in which cell each event lies and add a cell marker.

## Usage

```
addCell(
  hypFrame,
  owins,
  cellTypes = NULL,
  findOverlappingOwins = FALSE,
  warnOut = TRUE,
  coords = c("x", "y"),
  verbose = TRUE,
  addCellMarkers = TRUE,
  overwriteCells = FALSE,
  ...
)
```

## Arguments

<code>hypFrame</code>	A hyperframe
<code>owins</code>	A list containing a list of owins per point pattern. The length of the list must match the length of the hyperframe, and the names must match. Also lists of geojson objects, coordinate matrices or rois are accepted, see details.
<code>cellTypes</code>	A dataframe of cell types and other cell-associated covariates. If supplied, it must contain a variable 'cell' that is matched with the names of the owins
<code>findOverlappingOwins</code>	a boolean, should windows be checked for overlap? Can be computationally intensive.
<code>warnOut</code>	a boolean, should warning be issued when points are not contained in window?
<code>coords</code>	The names of the coordinates, if the windows are given as sets of coordinates.
<code>verbose</code>	A boolean, should verbose output be printed?
<code>addCellMarkers</code>	A boolean, should cell identities be added? Set this to FALSE if cell identifiers are already present in the data, and you only want to add windows and centroids.
<code>overwriteCells</code>	A boolean, should cells already present in hyperframe be overwritten?
<code>...</code>	Further arguments passed onto <a href="#">convertToOwins</a>

## Details

First the different cells are checked for overlap per point pattern if 'findOverlappingOwins' is TRUE. If no overlap is found, each event is assigned the cell that it falls into. Events not belonging to any cell will trigger a warning and be assigned 'NA'. Cell types and other variables are added to the marks if applicable. This function employs multithreading through the BiocParallel package. If this leads to excessive memory usage and crashes, try serial processing by setting register(SerialParam()). Different formats of windows are allowed, if the corresponding packages are installed. A dataframe of coordinates or a list of spatstat.geom owins is always allowed, as the necessary packages are required by smoppix. A 'SpatialPolygonsDataFrame' object is allowed if the 'polycub' package is installed, and a list of 'ijroi' object or a single 'ijzip' object if the 'RImageJROI' package is installed.

## Value

The hyperframe with cell labels added in the marks of the point patterns

## Note

By default, overlap between windows is not checked. Events are assigned to the first window they fall in. If you are not sure of the quality of the segmentation, do check your input or set checkOverlap to TRUE, even when this make take time.

## See Also

[buildHyperFrame](#), [convertToOwins](#)

## Examples

```
library(spatstat.random)
set.seed(54321)
n <- 1e3 # number of molecules
ng <- 25 # number of genes
nfov <- 3 # Number of fields of view
conditions <- 3
# sample xy-coordinates in [0, 1]
x <- runif(n)
y <- runif(n)
# assign each molecule to some gene-cell pair
gs <- paste0("gene", seq(ng))
gene <- sample(gs, n, TRUE)
fov <- sample(nfov, n, TRUE)
condition <- sample(conditions, n, TRUE)
# construct data.frame of molecule coordinates
df <- data.frame(gene, x, y, fov, "condition" = condition)
# A list of point patterns
listPPP <- tapply(seq(nrow(df)), df$fov, function(i) {
  ppp(x = df$x[i], y = df$y[i], marks = df[i, "gene", drop = FALSE])
}, simplify = FALSE)
# Regions of interest (roi): Diamond in the center plus four triangles
w1 <- owin(poly = list(x = c(0, .5, 1, .5), y = c(.5, 0, .5, 1)))
w2 <- owin(poly = list(x = c(0, 0, .5), y = c(.5, 0, 0)))
w3 <- owin(poly = list(x = c(0, 0, .5), y = c(1, 0.5, 1)))
w4 <- owin(poly = list(x = c(1, 1, .5), y = c(0.5, 1, 1)))
w5 <- owin(poly = list(x = c(1, 1, .5), y = c(0, 0.5, 0)))
hypFrame <- buildHyperFrame(df,
```

```

    coordVars = c("x", "y"),
    imageVars = c("condition", "fov")
  )
  nDesignFactors <- length(unique(hypFrame$image))
  wList <- lapply(seq_len(nDesignFactors), function(x) {
    list("w1" = w1, "w2" = w2, "w3" = w3, "w4" = w4, "w5" = w5)
  })
  names(wList) <- rownames(hypFrame) # Matching names is necessary
  hypFrame2 <- addCell(hypFrame, wList)

```

---

addDesign

*Add design variables to hyperframe*


---

### Description

Add design variables to hyperframe

### Usage

```
addDesign(hypFrame, desMat, designVec)
```

### Arguments

hypFrame	The hyperframe
desMat	The design matrix
designVec	The design vector

### Value

The hyperframe with design variables added

---

addNuclei

*Add nuclei to a hyperframe*


---

### Description

Add the nuclei identifiers to a hyperframe already containing cells.

### Usage

```

addNuclei(
  hypFrame,
  nucleiList,
  checkSubset = TRUE,
  verbose = TRUE,
  coords = c("x", "y"),
  overwriteNuclei = FALSE,
  ...
)

```

## Arguments

hypFrame	A hyperframe
nucleiList	A list containing a list of owin per point pattern. The length of the list must match the length of the hyperframe, and the names must match. Also lists of geojson objects, coordinate matrices or rois are accepted, see <a href="#">addCell</a>
checkSubset	A boolean, should be checked whether nuclei are encompassed by cells?
verbose	A boolean, should verbose output be printed?
coords	The names of the coordinates, if the nuclei are given as sets of coordinates.
overwriteNuclei	A boolean, should existing nuclei be replaced?
...	Further arguments passed onto <a href="#">convertToOwins</a>

## Details

The nuclei names must match the cell names already present, all other nuclei are dropped. A warning is issued when nuclei are not encompassed by their cell.

## Value

The hyperframe with nuclei added as entry

## See Also

[addCell](#), [convertToOwins](#)

## Examples

```
library(spatstat.random)
set.seed(54321)
n <- 1e3 # number of molecules
ng <- 25 # number of genes
nfov <- 3 # Number of fields of view
conditions <- 3
# sample xy-coordinates in [0, 1]
x <- runif(n)
y <- runif(n)
# assign each molecule to some gene-cell pair
gs <- paste0("gene", seq(ng))
gene <- sample(gs, n, TRUE)
fov <- sample(nfov, n, TRUE)
condition <- sample(conditions, n, TRUE)
# construct data.frame of molecule coordinates
df <- data.frame(gene, x, y, fov, "condition" = condition)
# A list of point patterns
listPPP <- tapply(seq(nrow(df)), df$fov, function(i) {
  ppp(x = df$x[i], y = df$y[i], marks = df[i, "gene", drop = FALSE])
}, simplify = FALSE)
# Regions of interest (roi): Diamond in the center plus four triangles
w1 <- owin(poly = list(x = c(0, .5, 1, .5), y = c(.5, 0, .5, 1)))
w2 <- owin(poly = list(x = c(0, 0, .5), y = c(.5, 0, 0)))
w3 <- owin(poly = list(x = c(0, 0, .5), y = c(1, 0.5, 1)))
w4 <- owin(poly = list(x = c(1, 1, .5), y = c(0.5, 1, 1)))
w5 <- owin(poly = list(x = c(1, 1, .5), y = c(0, 0.5, 0)))
```

```

hypFrame <- buildHyperFrame(df,
  coordVars = c("x", "y"),
  imageVars = c("condition", "fov")
)
nDesignFactors <- length(unique(hypFrame$image))
wList <- lapply(seq_len(nDesignFactors), function(x) {
  list("w1" = w1, "w2" = w2, "w3" = w3, "w4" = w4, "w5" = w5)
})
names(wList) <- rownames(hypFrame) # Matching names is necessary
hypFrame2 <- addCell(hypFrame, wList)
# The nuclei
n1 <- owin(poly = list(x = c(0.2, .4, 0.8, .4), y = c(.4, .2, .4, .8)))
n2 <- owin(poly = list(x = c(0.1, 0.1, .4), y = c(.4, .1, .1)))
n3 <- owin(poly = list(x = c(0.1, 0.1, .4), y = c(1, .75, 1)))
n4 <- owin(poly = list(x = c(1, 1, .6), y = c(.7, .9, .9)))
n5 <- owin(poly = list(x = c(.95, .95, .7), y = c(.1, .4, .1)))
nList <- lapply(seq_len(nDesignFactors), function(x) {
  list("w1" = n1, "w2" = n2, "w3" = n3, "w4" = n4, "w5" = n5)
})
names(nList) <- rownames(hypFrame) # Matching names is necessary
hypFrame3 <- addNuclei(hypFrame2, nList)

```

addTabObs

*Add tables with gene counts to the hyperframe, presort by gene and x-ccordinate and add design varibales*

## Description

Add tables with gene counts to the hyperframe, presort by gene and x-ccordinate and add design varibales

## Usage

```
addTabObs(hypFrame)
```

## Arguments

hypFrame      The hyperframe

## Value

The hyperframe with tabObs added

---

buildDataFrame	<i>Extract a data frame for a certain gene and PI from a fitted object</i>
----------------	----------------------------------------------------------------------------

---

## Description

Based on a fitted object, a dataframe with results for a certain feature and PI is built, e.g. in preparation for linear modelling.

## Usage

```
buildDataFrame(
  obj,
  gene,
  pi = c("nn", "nnPair", "edge", "centroid", "nnCell", "nnPairCell"),
  piMat,
  moransI = FALSE,
  numNNs = 8,
  weightMats,
  pppDf,
  prepMat,
  prepTabs,
  prepCells
)
```

## Arguments

obj	A results object. For distances to fixed objects, the result of a call to <a href="#">estPis</a> ; for nearest neighbour distances, the result of a call to <a href="#">addWeightFunction</a>
gene	A character string indicating the desired gene or gene pair (genes separated by double hyphens)
pi	character string indicating the desired PI
piMat	A data frame. Will be constructed if not provided, for internal use.
moransI	A boolean, should Moran's I be calculated? in the linear mixed model
numNNs	An integer, the number of nearest neighbours in the weight matrix for the calculation of the Moran's I statistic
weightMats	List of weight matrices for Moran's I calculation.
pppDf	Dataframe of point pattern-wise variables. It is precalculated in <code>fitLMMsSingle</code> for speed, but will be newly constructed when not provided.
prepMat, prepTabs, prepCells	Preconstructed objects to avoid looping over genes. For internal use mainly

## Value

A dataframe with estimated PIs and covariates

## See Also

[addWeightFunction](#), [buildMoransIDataFrame](#)



Examples

```
example(addWeightFunction, "smoppix")
dfUniNN <- buildDataFrame(yangObj, gene = "SmVND2", pi = "nn")
# Example analysis with linear mixed model
library(lmerTest)
mixedMod <- lmer(pi - 0.5 ~ day + (1 | root),
  weight = weight, data = dfUniNN,
  contrasts = list("day" = "contr.sum")
)
summary(mixedMod)
# Evidence for aggregation
```

---

buildFormula	<i>Build a formula from different components</i>
--------------	--------------------------------------------------

---

Description

Build a formula from different components

Usage

```
buildFormula(Formula, fixedVars, randomVars, outcome = "pi - 0.5")
```

Arguments

- Formula            A formula. If not supplied or equals NULL, will be overridden
- fixedVars, randomVars            Character vectors with fixed and random variables
- outcome            A character vector describing the outcome

Details

Random intercepts are assumed for the random effects, if more complicated designs are used, do supply your own formula.

Value

A formula

See Also

[fitLMMs,formula](#)

---

buildHyperFrame	<i>Build a hyperframe containing all point patterns of an experiment.</i>
-----------------	---------------------------------------------------------------------------

---

### Description

Build a spatstat hyperframe with point patterns and metadata. Matrices, dataframe, lists and SpatialExperiment inputs are accepted.

### Usage

```
buildHyperFrame(x, ...)

## S4 method for signature 'data.frame'
buildHyperFrame(
  x,
  coordVars,
  imageIdentifier = imageVars,
  imageVars,
  pointVars = setdiff(names(x), c(imageVars, imageIdentifier, coordVars, featureName)),
  featureName = "gene",
  ...
)

## S4 method for signature 'matrix'
buildHyperFrame(
  x,
  imageVars,
  imageIdentifier = imageVars,
  covariates,
  featureName = "gene",
  ...
)

## S4 method for signature 'list'
buildHyperFrame(
  x,
  coordVars = c("x", "y"),
  covariates = NULL,
  idVar = NULL,
  featureName = "gene",
  ...
)

## S4 method for signature 'SpatialExperiment'
buildHyperFrame(x, imageVars, pointVars, imageIdentifier = imageVars, ...)
```

### Arguments

x	the input object, see methods('buildHyperFrame')
...	additional constructor arguments

coordVars	Names of coordinates
imageIdentifier	A character vector of variables whose unique combinations define the separate point patterns (images)
imageVars	Covariates belonging to the point patterns
pointVars	Names of event-wise covariates such as gene or cell for each single point
featureName	The name of the feature identifier for the molecules.
covariates	A matrix or dataframe of covariates
idVar	An optional id variable present in covariates, that is matched with the names of covariates
list	A list of matrices or of point patterns of class 'spatstat.geom::ppp'

**Value**

An object of class 'hyperframe' from the 'spatstat.geom' package

**See Also**

[hyperframe](#)

**Examples**

```
data(Yang)
hypYang <- buildHyperFrame(Yang,
  coordVars = c("x", "y"),
  imageVars = c("day", "root", "section")
)
```

---

buildMoransIDataFrame *Build a data frame with Moran's I as outcome variable*

---

**Description**

Build a data frame with Moran's I as outcome variable

**Usage**

```
buildMoransIDataFrame(pi, piMat, weightMats)
```

**Arguments**

pi	character string indicating the desired PI
piMat	A data frame. Will be constructed if not provided, for internal use.
weightMats	List of weight matrices for Moran's I calculation.

**Value**

A modified data frame, containing the estimated Moran's I and its variance

**See Also**

[Moran.I](#), [buildMoransIWeightMat](#)

---

buildMoransIWeightMat	<i>Build a weight matrix based on nearest neighbourhood for Moran's I calculations</i>
-----------------------	----------------------------------------------------------------------------------------

---

### Description

Build a weight matrix based on nearest neighbourhood for Moran's I calculations

### Usage

```
buildMoransIWeightMat(coordMat, numNNs)
```

### Arguments

coordMat	A matrix of centroid coordinates
numNNs	An integer, the number of nearest neighbours in the weight matrix for the calculation of the Moran's I statistic

### Value

A sparse matrix of weights for Moran's I statistic, of equal value for the numNNs nearest neighbours and zero otherwise

### Note

The choice of numNN nearest neighbours is far less memory-glutton than a weight decaying continuously with distance

### See Also

[buildMoransIDataFrame](#), [Moran.I](#)

---

calcIndividualPIs	<i>Calculate individual PI entries of a single point pattern</i>
-------------------	------------------------------------------------------------------

---

### Description

Calculate individual PI entries of a single point pattern

### Usage

```
calcIndividualPIs(
  p,
  tabObs,
  pis,
  pSubLeft,
  owin,
  centroids,
  null,
  features,
```

```

    ecdfAll,
    ecdfsCell,
    loopFun,
    minDiff,
    minObsNN
  )

```

### Arguments

p	The point pattern
tabObs	A table of observed gene frequencies
pis	The PIs to be estimated or for which weighting functions is to be added
pSubLeft	The subsampled overall point pattern returned by subSampleP
owins, centroids	The list of windows corresponding to cells, and their centroids
null	A character vector, indicating how the null distribution is defined. See details.
features	A character vector, for which features should the probabilistic indices be calculated?
ecdfAll, ecdfsCell	Empirical cumulative distribution functions of all events and of cells within the cell, under the null
loopFun	The function to use to loop over the features. Defaults to bplapply except when looping over features within cells
minDiff	An integer, the minimum number of events from other genes needed for calculation of background distribution of distances. Matters mainly for within-cell calculations: cells with too few events are skipped.
minObsNN	An integer, the minimum number of events required for a gene to be analysed. See details.

### Details

For the single-feature nearest neighbour distances, the PI is average over the point pattern

### Value

A list containing PI entries per feature

### See Also

[estPis](#), [calcNNPI](#)

---

calcNNPI	<i>Estimate the PI for the nearest neighbour distances, given a set of ranks, using the negative hypergeometric distribution</i>
----------	----------------------------------------------------------------------------------------------------------------------------------

---

### Description

Estimate the PI for the nearest neighbour distances, given a set of ranks, using the negative hypergeometric distribution

### Usage

```
calcNNPI(Ranks, n, m, ties, r = 1)
```

### Arguments

Ranks	The (approximate) ranks, number of times observed distance is larger
n	the total number of observed distances minus the number of distances under consideration (the number of failures or black balls in the urn)
m	the number of observed distances (successes or white balls in the urn)
ties	The number of times the observed distance is equal to a null distance, of the same length as Ranks
r	The rank of distances considered, r=1 is nearest neighbour distance

### Details

Ties are counted half to match the definition of the PI.

### Value

A vector of evaluations of the negative hypergeometric distribution function

### See Also

[pnhyper](#), [calcIndividualPIs](#)

---

calcWindowDistPI	<i>Estimate the PI for the distance to a fixed object of interest, such as a cell wall or centroid</i>
------------------	--------------------------------------------------------------------------------------------------------

---

### Description

Estimate the PI for the distance to a fixed object of interest, such as a cell wall or centroid

### Usage

```
calcWindowDistPI(pSub, oWins, centroids, ecdfAll, pi)
```

**Arguments**

pSub	The subset point pattern containing only a single gene
owins, centroids	The list of windows corresponding to cells, and their centroids
ecdfAll	the cumulative distribution function under the null
pi	The type of PI to calculate

**Details**

Analysis of the distance to the border was introduced by (Joyner et al. 2013) in the form of the B-function. The independent evaluations of the B-functions under the null hypothesis represented by *ecdfAll* per cell are here returned as realizations of the probabilistic index.

**Value**

A list of vectors of estimated probabilistic indeces per event

**References**

Joyner M, Ross C, Seier E (2013). “Distance to the border in spatial point patterns.” *Spat. Stat.*, **6**, 24 - 40. ISSN 2211-6753, doi:[10.1016/j.spasta.2013.05.002](https://doi.org/10.1016/j.spasta.2013.05.002).

**See Also**

[addCell](#), [estPis](#)

---

centerNumeric	<i>Center numeric variables</i>
---------------	---------------------------------

---

**Description**

Center numeric variables

**Usage**

```
centerNumeric(x)
```

**Arguments**

x	The dataframe whose numeric variables are being centered
---	----------------------------------------------------------

**Value**

The adapted dataframe

---

checkFeatures	<i>Check if features are present in hyperframe</i>
---------------	----------------------------------------------------

---

**Description**

Check if features are present in hyperframe

**Usage**

```
checkFeatures(hypFrame, features)
```

**Arguments**

hypFrame	A hyperframe
features	A character vector, for which features should the probabilistic indices be calculated?

**Value**

Throws error when features not found

---

checkPi	<i>Check if the required PI's are present in the object</i>
---------	-------------------------------------------------------------

---

**Description**

Check if the required PI's are present in the object

**Usage**

```
checkPi(x, pi)
```

**Arguments**

x	The result of the PI calculation, or a weighting function
pi	A character string indicating the desired PI

**Value**

Throws an error when the PIs are not found, otherwise returns invisible



---

constructDesignVars	<i>Check for or construct design matrix</i>
---------------------	---------------------------------------------

---

**Description**

Run checks on design variables, or construct them as vector them if missing

**Usage**

```
constructDesignVars(designVars, lowestLevelVar, allCell, resList)
```

**Arguments**

designVars	The initial design variables
lowestLevelVar	Variable indicating the lowest level of nesting
allCell	A boolean, are all PIs cell-related?
resList	The results list

**Value**

A vector of design variables

**See Also**

[buildDataFrame](#)

---

convertToOwins	<i>Convert windows to spatstat.geom owin format</i>
----------------	-----------------------------------------------------

---

**Description**

Convert a list of windows in different possible formats to owin, for addition to a hyperframe.

**Usage**

```
convertToOwins(windows, namePPP, coords, ...)
```

**Arguments**

windows	The list of windows. See <a href="#">addCell</a> for accepted formats.
namePPP	the name of the point pattern, will be added to the cell names
coords	The names of the coordinates, if the windows are given as sets of coordinates.
...	passed onto as.owin

**Details**

Order of traversal of polygons may differ between data types. Where applicable, different orders are tried before throwing an error.

**Value**

A list of owin

**See Also**

[addCell](#), [as.owin](#)

---

`crossdistWrapper`

*A wrapper for C-functions calculating cross-distance matrix fast*

---

**Description**

A wrapper for C-functions calculating cross-distance matrix fast

**Usage**

```
crossdistWrapper(x, y)
```

**Arguments**

`x, y` the matrices or point patterns between which to calculate the cross distances

**Value**

a matrix of cross distances

---

Eng

*Spatial transcriptomics data of mouse fibroblast cells*

---

**Description**

Single-molecule spatial transcriptomics seqFISH+ data containing measurements of 10,000 genes in NIH/3T3 mouse fibroblast cells by (Eng et al. 2019). Molecule locations, gene identity and design variables are included, a subset of eight most expressed genes is included in the package, and the dataset was subsampled to 100,000 observations for memory reasons. In addition, a list of regions of interest (rois) is given describing the cell boundaries.

**Usage**

```
data(Eng)
```

**Format**

1. **Eng** A data frame with variables
  - x,y** Molecule coordinates
  - gene** Character vector with gene identities
  - experiment,fov** Design variables
2. **EngRois** A list of lists of regions of interest (ROIs): the cell boundaries

**Source**

[doi:10.1038/s415860191049y](https://doi.org/10.1038/s415860191049y)

**References**

Eng CL, Lawson M, Zhu Q, Dries R, Koulina N, Takei Y, Yun J, Cronin C, Karp C, Yuan G, Cai L (2019). “Transcriptome-scale super-resolved imaging in tissues by RNA seqFISH+.” *Nature*, **568**(7751), 235 - 239. ISSN 1476-4687, [doi:10.1038/s415860191049y](https://doi.org/10.1038/s415860191049y).

---

estGradients	<i>Estimate gradients over multiple point patterns, and test for significance</i>
--------------	-----------------------------------------------------------------------------------

---

**Description**

estGradients() estimate gradients on all single-molecule point patterns of a hyperframe. estGradientsSingle() is the workhorse function for a single point pattern. getPvaluesGradient() extracts the p-values of the fits.

**Usage**

```
estGradients(
  hypFrame,
  gradients = c("overall", if (!is.null(hypFrame$owins)) "cell"),
  fixedEffects = NULL,
  randomEffects = NULL,
  verbose = FALSE,
  features = getFeatures(hypFrame),
  silent = TRUE,
  loopFun = "bplapply",
  ...
)

estGradientsSingle(
  hypFrame,
  gradients,
  fixedForm,
  randomForm,
  fixedFormSimple,
  effects = NULL,
  ...
)

getPvaluesGradient(res, gradient, method = "BH")
```

**Arguments**

hypFrame	A hyperframe
gradients	The gradients types to be estimated: "overall" or within cell ("cell")

<code>fixedEffects, randomEffects</code>	Character vectors of fixed and random effects present in the hyperframe, modifying the baseline intensity. See details.
<code>verbose</code>	A boolean, whether to report on progress of the fitting process.
<code>features</code>	A character vector, for which features should the gradients indices be calculated?
<code>silent</code>	A boolean, should error messages from <code>spatstat.model::mppm</code> be printed?
<code>loopFun</code>	The function to use to loop over the features.
<code>...</code>	Passed onto <code>fitGradient</code>
<code>fixedForm, randomForm, fixedFormSimple</code>	Formulae for fixed effects, random effects and fixed effects without slopes respectively
<code>effects</code>	Character vector of fixed and random effects
<code>res</code>	The fitted gradients
<code>gradient</code>	The gradient to be extracted, a character vector equal to "overall" or "cell".
<code>method</code>	Method of multiplicity correction, see <a href="#">p.adjust</a> . Defaults to Benjamini-Hochberg.

### Details

The test for existence of a gradient revolves around interaction terms between x and y coordinates and image identifiers. If this interactions are significant, this implies existence of gradients in the different point patterns, albeit with different directions. Yet be aware that a gradient that is significant for a computer may look very different from the human perspective; many spatial patterns can be captured by a gradient to some extent. Baseline intensity corrections for every image or cell are included by default. The fixed and random effects modify the baseline intensity of the point pattern, not the gradient! Random effects can lead to problems with fitting and are dissuaded.

### Value

For `estGradients()`, a list with the estimated gradients

For `estGradientsSingle()`, a list containing

`overall` Overall gradients

`cell` Gradients within the cell

For `getPvaluesGradient()`, a vector of p-values

### Note

Fitting Poisson point processes is computation-intensive.

### See Also

[fitGradient](#)

### Examples

```
# Overall Gradients
data(Yang)
hypYang <- buildHyperFrame(Yang,
  coordVars = c("x", "y"),
  imageVars = c("day", "root", "section"))
```

```

)
yangGrads <- estGradients(hypYang[seq_len(2), ],
  features = getFeatures(hypYang)[1],
  fixedEffects = "day", randomEffects = "root")
# Gradients within cell
data(Eng)
hypEng <- buildHyperFrame(Eng[Eng$fov %in% c(1,2),],
  coordVars = c("x", "y"),
  imageVars = c("fov", "experiment"))
) #Subset for speed
hypEng <- addCell(hypEng, EngRois[rownames(hypEng)], verbose = FALSE)
# Limit number of cells and genes for computational reasons
engGrads <- estGradients(hypEng[seq_len(2),],
  features = feat <- getFeatures(hypEng)[1])
pVals <- getPValuesGradient(engGrads, "cell")

```

estPis

*Estimate probabilistic indices for single-molecule localization patterns, and add the variance weighting function.*

## Description

Estimate different probabilistic indices for localization on all point patterns of a hyperframe, and integrate the results in the same hyperframe. `estPisSingle()` is the workhorse function for a single point pattern.

`addWeighFunction()` adds a weighting function based on the data to the object by modeling variance as a non-increasing spline as a function of the number of events.

## Usage

```

estPis(
  hypFrame,
  pis = c("nn", "nnPair", "edge", "centroid", "nnCell", "nnPairCell"),
  verbose = TRUE,
  null = c("background", "CSR"),
  nPointsAll = switch(null, background = 50000, CSR = 2000),
  nPointsAllWithinCell = switch(null, background = 5000, CSR = 1000),
  nPointsAllWin = 10000,
  minDiff = 20,
  minObsNN = 1L,
  features = getFeatures(hypFrame),
  ...
)

estPisSingle(
  p,
  pis,
  null,
  tabObs,
  oWins = NULL,
  centroids = NULL,
  window = p$window,

```

```

    loopFun = "bplapply",
    features,
    nPointsAll,
    nPointsAllWithinCell,
    nPointsAllWin,
    minDiff,
    minObsNN
)

addWeightFunction(
  resList,
  pis = resList$pis,
  designVars,
  lowestLevelVar,
  maxObs = 1e+05,
  maxFeatures = 1000,
  minNumVar = 3,
  ...
)

```

### Arguments

hypFrame	A hyperframe
pis	The PIs to be estimated or for which weighting functions is to be added
verbose	A boolean, whether to report on progress of the fitting process.
null	A character vector, indicating how the null distribution is defined. See details.
nPointsAll, nPointsAllWithinCell	How many points to subsample or simulate to calculate the overall nearest neighbour distance distribution under the null hypothesis. The second argument (nPointsAllWithinCell) applies to within cell calculations, where a lower number usually suffices.
nPointsAllWin	How many points to subsample or simulate to calculate distance to cell edge or centroid distribution
minDiff	An integer, the minimum number of events from other genes needed for calculation of background distribution of distances. Matters mainly for within-cell calculations: cells with too few events are skipped.
minObsNN	An integer, the minimum number of events required for a gene to be analysed. See details.
features	A character vector, for which features should the probabilistic indices be calculated?
...	Additional arguments passed on to the <a href="#">scam</a> function, fitting the spline
p	The point pattern
tabObs	A table of observed gene frequencies
owins, centroids	The list of windows corresponding to cells, and their centroids
window	An window of class owin, in which events can occur
loopFun	The function to use to loop over the features. Defaults to bplapply except when looping over features within cells

resList	A results list, from a call to estPis().
designVars	A character vector containing all design factors (both fixed and random), that are also present as variables in hypFrame.
lowestLevelVar	The design variable at the lowest level of nesting, often separating technical replicates. The conditional variance is calculated within the groups of PIs defined by this variable.
maxObs, maxFeatures	The maximum number of observations respectively features for fitting the weighting function. See details.
minNumVar	The minimum number of observations needed to calculate a variance. Groups with fewer replicates are ignored.

## Details

The null distribution used to calculate the PIs can be either 'background' or 'null'. For 'background', the observed distributions of all genes is used. Alternatively, for null = 'CSR', Monte-Carlo simulation under complete spatial randomness is performed within the given window to find the null distribution of the distance under study. See (Hawinkel et al. 2025) for precise definition of the PI.

The 'nn' prefix indicates that nearest neighbour distances are being used, either univariately or bivariate. The suffix 'Pair' indicates that bivariate probabilistic indices, testing for co- and antilocalization are being used. 'edge' and 'centroid' calculate the distance to the edge respectively the centroid of the windows added using the [addCell](#) function. The suffix 'Cell' indicates that nearest neighbour distances are being calculated within cells only.

It can be useful to set the minObsNN higher than the default of 5 for calculations within cells when the number of events is low, not to waste computation time on gene (pairs) with very variable PI estimates.

Provide either 'designVars' or 'lowestLevelVar'. The 'designVars' are usually the same as the regressors in the linear model. In case 'lowestLevelVar' is provided, the design variables are set to all imageVars in the hypFrame object except lowestLevelVar. When the PI is calculated on the cell level ("nnCell" or "nnPairCell"), the cell is always the lowest nesting level, and inputs to 'designVars' or 'lowestLevelVar' will be ignored for these PIs. The registered parallel backend will be used for fitting the trends of the different PIs. For computational and memory reasons, for large datasets the trend fitting is restricted to a random subset of the data through the maxObs and maxFeatures parameters.

## Value

For estPis(), the hyperframe with the estimated PIs present in it

For estPisSingle(), a list of data frames with estimated PIs per gene and/or gene pair:

pointDists	PIs for pointwise distances overall
windowDists	PIs for distances to cell wall or centroid
withinCellDists	PIs for pointwise distances within cell

For addWeightFunction(), the input object 'resList' with a slot 'Wfs' added containing the weighting functions.

## See Also

[buildDataFrame](#), [estPis](#)

**Examples**

```

data(Yang)
hypYang <- buildHyperFrame(Yang,
  coordVars = c("x", "y"),
  imageVars = c("day", "root", "section")
)
yangPims <- estPis(hypYang[c(seq_len(4), seq(27, 29)), ], pis = "nn",
  nPointsAll = 4e2)
# Univariate nearest neighbour distances
yangObj <- addWeightFunction(yangPims, designVars = c("day", "root"))
# Add the weight functions
yangObj <- addWeightFunction(yangPims, lowestLevelVar = "section",
  pi = "nn")
# Alternative formulation with 'lowestLevelVar'

```

---

evalWeightFunction	<i>Evaluate a variance weighting function</i>
--------------------	-----------------------------------------------

---

**Description**

Evaluate the variance weighting function to return unnormalized weights

**Usage**

```
evalWeightFunction(wf, newdata)
```

**Arguments**

wf	The weighting function
newdata	A data frame with new data

**Value**

A vector of weights, so the inverse of predicted variances, unnormalized

**See Also**

[predict.scam](#), [addWeightFunction](#)

**Examples**

```

data(Yang)
hypYang <- buildHyperFrame(Yang, coordVars = c("x", "y"),
  imageVars = c("day", "root", "section")
)
yangPims <- estPis(hypYang, pis = "nn", features = getFeatures(hypYang)[12:19], nPointsAll = 5e2)
# First Build the weighting function
yangObj <- addWeightFunction(yangPims, designVars = c("day", "root"))
evalWeightFunction(yangObj$Wfs$nn, newdata = data.frame("NP" = 2))

```



---

extractResults	<i>Extract results from a list of fitted LMMs. For internal use mainly.</i>
----------------	-----------------------------------------------------------------------------

---

### Description

Extract results from a list of fitted LMMs. For internal use mainly.

### Usage

```
extractResults(
  models,
  hypFrame,
  subSet = "piMod",
  fixedVars = NULL,
  method = "BH"
)
```

### Arguments

models	The models
hypFrame	The original hyperframe
subSet	The name of the subset to be extracted, either PI or Moran's I
fixedVars	The fixed effects for which the effect is to be reported
method	Multiplicity correction method passed onto p.adjust

### Value

A list of matrices, all containing estimate, standard error, p-value and adjusted p-value

### See Also

[fitLMMs](#), [p.adjust](#)

---

findEcdfsCell	<i>Construct empirical cumulative distribution functions (ecdfs) for distances within the cell</i>
---------------	----------------------------------------------------------------------------------------------------

---

### Description

The distance distribution under the null hypothesis of complete spatial randomness (CSR) within the cell is the same for all genes. This function precalculates this distribution using Monte-Carlo simulation under CSR, and summarizes it in an ecdf object

### Usage

```
findEcdfsCell(p, owins, nPointsAllWin, centroids, null, pis, loopFun)
```

**Arguments**

p	The point pattern
owins, centroids	The list of windows corresponding to cells, and their centroids
nPointsAllWin	How many points to subsample or simulate to calculate distance to cell edge or centroid distribution
null	A character vector, indicating how the null distribution is defined. See details of <a href="#">estPis</a> .
pis	The PIs to be estimated or for which weighting functions is to be added
loopFun	The function to use to loop over the features. Defaults to bplapply except when looping over features within cells

**Value**

The list of ecdf functions

**See Also**

[ecdf](#)

---

findOverlap	<i>Find overlap between list of windows</i>
-------------	---------------------------------------------

---

**Description**

The function seeks overlap between the list of windows supplied, and throws an error when found or returns the id's when found.

**Usage**

```
findOverlap(owins, centroids = NULL, returnIds = FALSE, numCentroids = 30)
```

**Arguments**

owins	the list of windows
centroids	The centroids of the windows
returnIds	A boolean, should the indices of the overlap be returned? If FALSE an error is thrown at the first overlap
numCentroids	An integer, the number of cells with closest centroids to consider looking for overlap

**Value**

Throws an error when overlap found, otherwise returns invisible. When returnIds=TRUE, the indices of overlapping windows are returned.

## Examples

```
library(spatstat.geom)
owins <- replicate(10, owin(
  xrange = runif(1) + c(0, 0.2),
  yrange = runif(1) + c(0, 0.1)
), simplify = FALSE)
idOverlap <- findOverlap(owins, returnIds = TRUE)
```

---

fitGradient

*Test for presence of gradient in a hyperframe of point patterns*


---

## Description

A Poisson process is fitted to the data assuming exponential relationship with intensity of the interaction between x and y variables and image identifier. This is compared to a model without this interaction to test for the significance of the gradient.

## Usage

```
fitGradient(
  hypFrame,
  fixedForm,
  randomForm,
  fixedFormSimple,
  returnModel = FALSE,
  silent,
  ...
)
```

## Arguments

hypFrame	the hyperframe
fixedForm, randomForm, fixedFormSimple	Formulae for fixed effects, random effects and fixed effects without slopes respectively
returnModel	A boolean, should the entire model be returned? Otherwise the p-value and coefficient vector are returned
silent	A boolean, should error messages from spatstat.model::mppm be printed?
...	passed onto <a href="#">mppm</a>

## Value

A list containing

- pVal The p-value for existence of gradients
- coef The model coefficients

or a mppm model when returnModel is true

## See Also

[estGradients](#)

fitLMMs

*Fit linear (mixed) models for all probabilistic indices (PIs) and all genes***Description**

The PI is used as outcome variable in a linear (mixed) model, with design variables as regressors. Separate models are fitted for every combination of gene and PI. `fitLMMsSingle()` is the workhorse function for a single point pattern,

`getResults()` extracts effect size estimates, standard errors and adjusted p-values for a certain parameter from a linear model.

**Usage**

```
fitLMMs(
  obj,
  pis = obj$pis,
  fixedVars = NULL,
  randomVars = NULL,
  verbose = TRUE,
  returnModels = FALSE,
  Formula = NULL,
  randomNested = TRUE,
  features = getEstFeatures(obj),
  moranFormula = NULL,
  addMoransI = FALSE,
  numNNs = 10,
  ...
)

fitLMMsSingle(
  obj,
  pi,
  fixedVars,
  randomVars,
  verbose,
  returnModels,
  Formula,
  randomNested,
  features,
  addMoransI,
  weightMats,
  moranFormula
)

getResults(obj, pi, parameter, moransI = FALSE)
```

**Arguments**

`obj`                      The result object

pis	Optional, the pis required. Defaults to all pis in the object
fixedVars	Names of fixed effects
randomVars	Names of random variables
verbose	A boolean, should the formula be printed?
returnModels	a boolean: should the full models be returned? Otherwise only summary statistics are returned
Formula	A formula; if not supplied it will be constructed from the fixed and random variables
randomNested	A boolean, indicating if random effects are nested within point patterns. See details.
features	The features for which to fit linear mixed models. Defaults to all features in the object
moranFormula	Formula for Moran's I model fitting
addMoransI	A boolean, include Moran's I of the cell-wise PIs in the calculation
numNNs	An integer, the number of nearest neighbours in the weight matrix for the calculation of the Moran's I statistic
...	Passed onto fitLMMsSingle
pi	The desired PI
weightMats	A list of weight matrices for Moran's I
parameter	The desired parameter
moransI	A boolean, should results for the Moran's I be returned?

## Details

Genes or gene pairs with insufficient observations will be silently omitted. When randomVars is provided as a vector, independent random intercepts are fitted for them by default. Providing them separated by '\ ' or ':' as in the lmer formulas is also allowed to reflect nesting structure, but the safest is to construct the formula yourself and pass it onto fitLMMs.

It is by default assumed that random effects are nested within the point patterns. This means for instance that cells with the same name but from different point patterns are assigned to different random effects. Set 'randomNested' to FALSE to override this behaviour.

The Moran's I statistic is used to test whether cell-wise PIs ("nnCell", "nnCellPair", "edge" and "centroid") are spatially autocorrelated across the images. The numeric value of the PI is assigned to the centroid location, and then Moran's I is calculated with a fixed number of numNNs nearest neighbours with equal weights.

## Value

For fitLMMs(), a list of fitted objects

For fitLMMsSingle(), a list of test results, if requested also the linear models are returned

For getResults(), the matrix with results, with p-values in ascending order

Estimate	The estimated PI
se	The corresponding standard error
pVal	The p-value
pAdj	The Benjamini-Hochberg adjusted p-value

See Also

[buildMoransIDataFrame](#), [buildDataFrame](#)

Examples

```
example(addWeightFunction, "smoppix")
lmmModels <- fitLMMs(yangObj, fixedVars = "day", randomVars = "root")
res <- getResultts(lmmModels, "nn", "Intercept") #Extract the results
head(res)
```

---

fitPiModel	<i>Fit a linear model for an individual gene and PI combination</i>
------------	---------------------------------------------------------------------

---

Description

Fit a linear model for an individual gene and PI combination

Usage

```
fitPiModel(Formula, dff, contrasts, Control, MM, Weight = NULL)
```

Arguments

Formula	A formula; if not supplied it will be constructed from the fixed and random variables
dff	The dataframe
contrasts	The contrasts to be used, see <a href="#">model.matrix</a>
Control	Control parameters
MM	A boolean, should a mixed model be tried
Weight	A weight variable

Value

A fitted model

See Also

[fitLMMsSingle](#)

---

getCoordsMat	<i>Extract coordinates from a point pattern or data frame</i>
--------------	---------------------------------------------------------------

---

**Description**

Extract coordinates from a point pattern or data frame

**Usage**

```
getCoordsMat(x)
```

**Arguments**

x                      the point pattern, dataframe or matrix

**Value**

the matrix of coordinates

---

getDesignVars	<i>getDesignVars() returns all design variables, both at the level of the point pattern and the level of the event</i>
---------------	------------------------------------------------------------------------------------------------------------------------

---

**Description**

getDesignVars() returns all design variables, both at the level of the point pattern and the level of the event

getPPPvars() extract variables related to point patterns

getEventVars() extracts variables related to events, e.g. the cell (the marks)

**Usage**

```
getDesignVars(x)
```

```
getPPPvars(
  x,
  exclude = c("tabObs", "centroids", "owins", "ppp", "pimRes", "image", "nuclei")
)
```

```
getEventVars(x, exclude = c("x", "y", "z"))
```

**Arguments**

x                      The results list, output from estPis

exclude                variables to exclude

**Details**

getDesignVars() returns all design variables, [getPPPvars](#) returns design variables related to the different images and [getEventVars](#) returns design variables related to the individual events

**Value**

A vector of design variables

---

getElement	<i>Extract an element from a matrix or vector</i>
------------	---------------------------------------------------

---

**Description**

Extract an element from a matrix or vector

**Usage**

```
getElement(x, e)
```

**Arguments**

x	the matrix or vector
e	The column or element name

**Value**

The desired element

---

getFeatures	<i>Extract all unique features from an object, or the ones for which PIs were estimated</i>
-------------	---------------------------------------------------------------------------------------------

---

**Description**

Extract all unique features from an object, or the ones for which PIs were estimated

**Usage**

```
getFeatures(x)
```

**Arguments**

x	A hyperframe or a results list containing a hyperframe
---	--------------------------------------------------------

**Value**

A vector of features

**Examples**

```
data(Yang)
hypYang <- buildHyperFrame(Yang,
  coordVars = c("x", "y"),
  imageVars = c("day", "root", "section")
)
head(getFeatures(hypYang))
```



---

getGp	<i>Helper function to get gene pair from a vector or list</i>
-------	---------------------------------------------------------------

---

**Description**

When provided with argument "geneA–geneB", looks for this gene pair as well as for "geneB–geneA" in the provided object.

**Usage**

```
getGp(x, gp, drop = TRUE, Collapse = "--", notFoundReturn = NULL)
```

**Arguments**

x	The object in which to look
gp	A character string describing the gene pair
drop	A boolean, should matrix attributes be dropped in [] subsetting
Collapse	The character separating the gene pair
notFoundReturn	value to return if element is not found

**Value**

The element sought

**Examples**

```
mat <- t(cbind(
  "gene1--gene2" = c(1, 2),
  "gene1--gene3" = c(2, 3)
))
getGp(mat, "gene3--gene1")
```

---

getHypFrame	<i>Extract the hyperframe</i>
-------------	-------------------------------

---

**Description**

Extract the hyperframe

**Usage**

```
getHypFrame(x)
```

**Arguments**

x	The hyperframe, or list containing one
---	----------------------------------------

**Value**

the hyperframe

---

loadBalanceBplapply	<i>Parallel processing with BiocParallel with load balancing</i>
---------------------	------------------------------------------------------------------

---

### Description

The vector to iterate over (iterator) is split into as many parts as there are cores available, such that each core gets an equal load and overhead is minimized. The registered backend is then used by default to multithread using [bplapply](#).

### Usage

```
loadBalanceBplapply(iterator, func, loopFun = "bplapply")
```

### Arguments

iterator	The vector to iterate over
func	The function to apply to each element
loopFun	The looping function, can also be 'lapply' for serial processing

### Value

A list with the same length as iterator

---

makeDesignVar	<i>Make design variable by combining different design variables</i>
---------------	---------------------------------------------------------------------

---

### Description

Make design variable by combining different design variables

### Usage

```
makeDesignVar(x, designVars, sep = "_")
```

### Arguments

x	the design matrix
designVars	the design variables to be combined
sep	The string to separate the components

### Value

a vector of design levels

---

makePairs	<i>An aux function to build gene pairs</i>
-----------	--------------------------------------------

---

**Description**

An aux function to build gene pairs

**Usage**

```
makePairs(genes)
```

**Arguments**

genes	The genes to be combined
-------	--------------------------

**Value**

A character vector of gene pairs

**Examples**

```
genes <- paste0("gene", seq_len(4))
makePairs(genes)
```

---

moransI	<i>Calculate the Moran's I test statistic for spatial autocorrelation</i>
---------	---------------------------------------------------------------------------

---

**Description**

The Moran's I test statistic and its variance are calculated

**Usage**

```
moransI(x, W)
```

**Arguments**

x	A vector of outcomes
W	The matrix of weights, with dimensions equal to the length of x

**Details**

The implementation is inspired on the one from `ape::Moran.I`, but more bare-bones for a sparse weight matrix with certain properties as prepared by `buildMoransIWeightMat`, making it faster and using less memory.

**Value**

A vector of length 2: the Moran's I statistic and its variance

**Note**

Calculations are only correct for weight matrices as prepared by buildMoransIWeightMat!

**See Also**

[Moran.I](#)

---

named.contr.sum	<i>A version of contr.sum that retains names, a bit controversial but also clearer</i>
-----------------	----------------------------------------------------------------------------------------

---

**Description**

A version of contr.sum that retains names, a bit controversial but also clearer

**Usage**

```
named.contr.sum(x, ...)
```

**Arguments**

`x, ...` passed on to contr.sum

**Value**

The matrix of contrasts

**Note**

After <https://stackoverflow.com/questions/24515892/r-how-to-contrast-code-factors-and-retain-meaningful-labels-in-output-summary>

---

nestRandom	<i>Nest random effects within fixed variables, in case the names are the same</i>
------------	-----------------------------------------------------------------------------------

---

**Description**

Nest random effects within fixed variables, in case the names are the same

**Usage**

```
nestRandom(df, randomVars, fixedVars)
```

**Arguments**

<code>df</code>	The dataframe
<code>randomVars</code>	The random variables
<code>fixedVars</code>	The fixed variables

**Value**

The dataframe with adapted randomVars

plotCells

*Plot the n cells with highest abundance of a feature***Description**

After testing for within-cell patterns, it may be useful to look at the cells with the most events for certain genes. These are plotted here, but the spatial location of the cells in the point pattern is lost! The choice and ranking of cells is one of decreasing gene (pair) expression.

**Usage**

```
plotCells(
  obj,
  features = getFeatures(obj)[seq_len(3)],
  nCells = 100,
  Cex = 1.5,
  borderColVar = NULL,
  borderCols = rev(palette()),
  Mar = c(0.5, 0.1, 0.75, 0.1),
  warnPosition = TRUE,
  summaryFun = "min",
  plotNuclei = !is.null(getHypFrame(obj)$nuclei),
  nucCol = "lightblue",
  ...
)
```

**Arguments**

obj	A hyperframe, or an object containing one
features	The features to be plotted, a character vector
nCells	An integer, the number of cells to be plotted
Cex	The point expansion factor
borderColVar	The variable to colour borders of the cell
borderCols	Colour palette for the borders
Mar	the margins
warnPosition	A boolean, should a warning be printed on the image that cells are not in their original location?
summaryFun	A function to summarize the gene-cell table in case multiple genes are plotted, to determine which cells are plotted. Choose "min" for cells with the highest minimum, or "sum" for highest total expression of the combination of genes
plotNuclei	A boolean, should nuclei be added?
nucCol	A character string, the colour in which the nucleus' boundary is plotted
...	Additional arguments, currently ignored

**Value**

Plots cells with highest expression to the plotting window, returns invisible

## Examples

```
example(addCell, "smoppix")
plotCells(hypFrame2, "gene1")
plotCells(hypFrame2, "gene1", borderColVar = "condition", nCells = 10)
```

---

plotExplore

*Plot a hyperframe with chosen features highlighted*


---

## Description

All points of the hyperframe are plotted in grey, with a subset of features highlighted in colour. A selection of point patterns is plotted that fit in the window, by default the first six. This function is meant for exploratory purposes as well as for visual confirmation of findings.

## Usage

```
plotExplore(
  hypFrame,
  features = getFeatures(hypFrame)[seq_len(6)],
  ppps,
  numPps,
  maxPlot = 1e+05,
  Cex = 1,
  plotWindows = !is.null(hypFrame$owins),
  plotPoints = TRUE,
  plotNuclei = !is.null(hypFrame$nuclei),
  piEsts = NULL,
  Xlim = NULL,
  Ylim = NULL,
  Cex.main = 1.1,
  Mar = c(0.5, 0.1, 0.9, 0.1),
  titleVar = NULL,
  piColourCell = NULL,
  palCols = c("blue", "yellow"),
  nucCol = "lightblue",
  border = NULL,
  CexLegend = 1.4,
  CexLegendMain = 1.7,
  Nrow,
  Cols
)
```

## Arguments

hypFrame	The hyperframe
features	A small number of features to be highlighted. Defaults to the first 5.
ppps	The rownames or indices of the point patterns to be plotted. Defaults to maximum 99.
numPps	The number of point patterns with highest expression to be shown. Ignored is pps is given.

maxPlot	The maximum number of events plotted per point pattern
Cex, Cex.main	Point and title expansion factors, respectively
plotWindows	A boolean, should windows be plotted too?
plotPoints	A boolean, should the molecules be plotted as points?
plotNuclei	A boolean, should the nuclei be plotted?
piEsts	Set of PI estimates, returned by estPis
Xlim, Ylim	plotting limits
Mar	the margins
titleVar	Image variable to be added to the title
piColourCell	PI by which to colour the cell
palCols	Two extremes of the colour palette for colouring the cells
nucCol	The colour for the nucleus window
border	Passed on to plot.owin, and further to graphics::polygon
CexLegend, CexLegendMain	Expansion factor for the legend and its title respectively
Nrow	Number of rows of the facet plot. Will be calculated if missing.
Cols	colours vector named by features. If missing a default palette is used

## Details

When cell-specific PIs are calculated ("nnCell", "nnCellPair", "edge", "centroid"), the cells can be coloured by them to investigate their spatial distribution, for instance those discovered through Moran's I statistic. The colour palette is taken from the output of palette(), so set that one to change the colour scheme.

## Value

Plots a facet of point patterns to output

## Note

palCols sets the pseudo-continuous scale to colour cells.

## Examples

```
example(buildHyperFrame, "smoppix")
plotExplore(hypYang)
plotExplore(hypYang, titleVar = "day")
plotExplore(hypYang, features = c("SmRBRb", "SmTMO5b", "SmWER--SmAHK4f"))
```

---

plotTopResults

*Plot the most significant findings for a certain PI*


---

## Description

Extract the most significant features for a certain PI and direction of effect, and plot them using an appropriate function: either [plotExplore](#) or [plotCells](#)

## Usage

```
plotTopResults(
  hypFrame,
  results,
  pi,
  effect = "Intercept",
  what = if (pi %in% c("nn", "nnCell")) {
    "aggregated"
  } else if (pi %in%
    c("nnPair", "nnPairCell")) {
    "colocalized"
  } else if (pi %in% c("edge",
    "centroid")) {
    "close"
  },
  sigLevel = 0.05,
  numFeats = 2,
  piThreshold = switch(effect, Intercept = 0.5, 0),
  effectParameter = NULL,
  ...
)
```

## Arguments

hypFrame	The hyperframe with the data
results	The results frame
pi	A character string, specifying the probabilistic index
effect	The name of the effect
what	Which features should be detected? Can be abbreviated, see details.
sigLevel	The significance level
numFeats	The number of features to plot
piThreshold	The threshold for PI, a minimum effect size
effectParameter	A character string, indicating which parameter to look for when effect is provided
...	passed onto plotting functions <a href="#">plotCells</a> or <a href="#">plotExplore</a>



## Details

The "what" argument indicates if features far from or close to cell wall or centroid should be shown for pi "edge" or "centroid", aggregated or regular features for "nn" and "nnCell" and colocalized or antilocalized features for "nnPair" and "nnPairCell". Partial matching is allowed. Defaults to small probabilistic indices: proximity, aggregation and colocalization. For fixed effects, provide the name of the parameter, in combination with what. For instance, what = "regular", effect = "Var1" and effectParameter = "level1" will return features more regular at level1 of the variable than at baseline.

## Value

A plot from plotCells or plotExplore, throws an error when no features meet the criteria

## See Also

[plotCells](#), [plotExplore](#), [fitLMMs](#)

## Examples

```
example(fitLMMs, "smoppix")
plotTopResults(hypYang, lmmModels, "nn")
#For the sake of illustration, set high significance level, as example dataset is small
plotTopResults(hypYang, lmmModels, "nn",
  effect = "day", what = "reg",
  effectParameter = "day0", sigLevel = 1-1e-10)
```

---

plotWf

*Plot the variance weighting function*

---

## Description

The observation weights are plotted as a function of number of events. For a univariate PI, this is a line plot, for a bivariate PI this is a scatterplot of majority gene as a function of minority gene, with the weight represented as a colour scale. The minority respectively majority gene are the genes in the gene pair with least and most events

## Usage

```
plotWf(obj, pi = obj$pis[1])
```

## Arguments

obj	The result of a call to <a href="#">addWeightFunction</a>
pi	The PI for which to plot the weighting function

## Value

For univariate PI, returns a line plot; for bivariate PI a ggplot object

## Examples

```
example(addWeightFunction, "smoppix")
plotWf(yangObj, "nn")
```

smoppix

*smoppix: Analyze Single Molecule Spatial Omics Data Using the Probabilistic Index***Description**

Test for univariate and bivariate spatial patterns in spatial omics data with single-molecule resolution. The tests implemented allow for analysis of nested designs and are automatically calibrated to different biological specimens. Tests for aggregation, colocalization, gradients and vicinity to cell edge or centroid are provided.

**Author(s)**

**Maintainer:** Stijn Hawinkel <stijn.hawinkel@psb.ugent.be> ([ORCID](#))

**See Also**

Useful links:

- <https://github.com/sthawinke/smoppix>
- Report bugs at <https://github.com/sthawinke/smoppix/issues>

sortGp

*Sort feature pairs alphabetically***Description**

Sort feature pairs alphabetically

**Usage**

```
sortGp(featurePairs)
```

**Arguments**

featurePairs    The feature pairs to be sorted

**Value**

A character vector of the same length as the features, with pairs sorted

---

splitWindow	<i>Split a number of plots into rows and columns</i>
-------------	------------------------------------------------------

---

**Description**

Split a number of plots into rows and columns

**Usage**

```
splitWindow(x)
```

**Arguments**

x	The number of plots
---	---------------------

**Value**

A vector of length 2 with required number of rows and columns

---

subSampleP	<i>Subsample a point pattern when it is too large</i>
------------	-------------------------------------------------------

---

**Description**

Subsample a point pattern when it is too large

**Usage**

```
subSampleP(p, nSims, returnId = FALSE)
```

**Arguments**

p	The point pattern
nSims	The maximum size
returnId	A boolean, should the id of the sampled elements be returned?

**Value**

A point pattern, subsampled if necessary

---

sund	<i>Helper function to spit gene pairs</i>
------	-------------------------------------------

---

**Description**

Helper function to spit gene pairs

**Usage**

```
sund(x, sep = "--")
```

**Arguments**

x	character string
sep	The character used to split

**Value**

The split string

**Examples**

```
GenePair <- "gene1--gene2"
sund(GenePair)
```

---

writeToXlsx	<i>Write effect sizes and p-values results to excel worksheet</i>
-------------	-------------------------------------------------------------------

---

**Description**

The results of the linear models are written to an excel spreadsheet with different tabs for every sign (PI smaller than or larger than 0.5) of every PI, sorted by increasing p-value.

**Usage**

```
writeToXlsx(obj, file, overwrite = FALSE, digits = 3, sigLevel = 0.05)
```

**Arguments**

obj	The results of linear model fitting
file	The file to write the results to
overwrite	A boolean, should the file be overwritten if it exists already?
digits	An integer, the number of significant digits to retain for the PI, raw and adjusted p-values
sigLevel	The significance level threshold to use for the adjusted p-values, only features exceeding the threshold are written to the file. Set this parameter to 1 to write all features

## Details

If no feature exceeds the significance threshold for a certain pi and parameter combination, an empty tab is created. For fixed effects, a single tab is written for PI differences of any sign. The "baseline" tabs indicate the overall patterns, the other tabs are named after the fixed effects and indicate departure from this baseline depending on this fixed effect

## Value

Returns invisible with a message when writing operation successful, otherwise throws an error.

## See Also

[createWorkbook](#), [writeData](#), [addWorksheet](#), [saveWorkbook](#)

## Examples

```
example(fitLMMs, "smoppix")
writeToXlsx(lmmModels, "tmpFile.xlsx")
file.remove("tmpFile.xlsx")
```

---

Yang

*Spatial transcriptomics data of Selaginella moellendorffii roots*

---

## Description

Single-molecule spatial transcriptomics smFISH data of *Selaginella moellendorffii* roots of a replicated experiment by (Yang et al. 2023). Molecule locations, gene identity and design variables are included. Only a subset of the data, consisting of roots 1-3 and sections 1-5 is included in the package for computational and memory reasons. The data are in table format to illustrate conversion to [hyperframe](#) using [buildHyperFrame](#).

## Usage

```
data(Yang)
```

## Format

A data matrix

**x,y** Molecule coordinates

**gene** Character vector with gene identities

**root,section,day** Design variables

## Source

[doi:10.1016/j.cub.2023.08.030](https://doi.org/10.1016/j.cub.2023.08.030)

## References

Yang X, Poelmans W, Grones C, Lakehal A, Pevernagie J, Bel MV, Njo M, Xu L, Nelissen H, Rybel BD, Motte H, Beeckman T (2023). "Spatial transcriptomics of a lycophyte root sheds light on root evolution." *Curr. Biol.*, **33**(19), 4069 - 4084. ISSN 0960-9822, [doi:10.1016/j.cub.2023.08.030](https://doi.org/10.1016/j.cub.2023.08.030).

# Index

- \* **datasets**
  - Eng, [18](#)
  - Yang, [45](#)
- \* **internal**
  - smoppix, [42](#)
- addCell, [3](#), [6](#), [15](#), [17](#), [18](#), [23](#)
- addDesign, [5](#)
- addNuclei, [5](#)
- addTabObs, [7](#)
- addWeightFunction, [8](#), [24](#), [41](#)
- addWeightFunction (estPis), [21](#)
- addWorksheet, [45](#)
- as.owin, [18](#)
- bplapply, [34](#)
- buildDataFrame, [8](#), [17](#), [23](#), [30](#)
- buildFormula, [9](#)
- buildHyperFrame, [4](#), [10](#), [45](#)
- buildHyperFrame, data.frame-method
  - (buildHyperFrame), [10](#)
- buildHyperFrame, list-method
  - (buildHyperFrame), [10](#)
- buildHyperFrame, matrix-method
  - (buildHyperFrame), [10](#)
- buildHyperFrame, SpatialExperiment-method
  - (buildHyperFrame), [10](#)
- buildMoransIDataFrame, [8](#), [11](#), [12](#), [30](#)
- buildMoransIWeightMat, [11](#), [12](#)
- calcIndividualPIs, [12](#), [14](#)
- calcNNPI, [13](#), [14](#)
- calcWindowDistPI, [14](#)
- centerNumeric, [15](#)
- checkFeatures, [16](#)
- checkPi, [16](#)
- constructDesignVars, [17](#)
- convertToOwins, [3](#), [4](#), [6](#), [17](#)
- createWorkbook, [45](#)
- crossdistWrapper, [18](#)
- ecdf, [26](#)
- Eng, [18](#)
- EngRois (Eng), [18](#)
- estGradients, [19](#), [27](#)
- estGradientsSingle (estGradients), [19](#)
- estPis, [8](#), [13](#), [15](#), [21](#), [23](#), [26](#)
- estPisSingle (estPis), [21](#)
- evalWeightFunction, [24](#)
- extractResults, [25](#)
- findEcdfsCell, [25](#)
- findOverlap, [26](#)
- fitGradient, [20](#), [27](#)
- fitLMMs, [9](#), [25](#), [28](#), [41](#)
- fitLMMsSingle, [30](#)
- fitLMMsSingle (fitLMMs), [28](#)
- fitPiModel, [30](#)
- formula, [9](#)
- getCoordsMat, [31](#)
- getDesignVars, [31](#)
- getElement, [32](#)
- getEventVars, [31](#)
- getEventVars (getDesignVars), [31](#)
- getFeatures, [32](#)
- getGp, [33](#)
- getHypFrame, [33](#)
- getPPPPvars, [31](#)
- getPPPPvars (getDesignVars), [31](#)
- getPvaluesGradient (estGradients), [19](#)
- getResults (fitLMMs), [28](#)
- hyperframe, [11](#), [45](#)
- loadBalanceBplapply, [34](#)
- makeDesignVar, [34](#)
- makePairs, [35](#)
- model.matrix, [30](#)
- Moran.I, [11](#), [12](#), [36](#)
- moransI, [35](#)
- mppm, [27](#)
- named.contr.sum, [36](#)
- nestRandom, [36](#)
- p.adjust, [20](#), [25](#)
- plotCells, [37](#), [40](#), [41](#)

plotExplore, [38](#), [40](#), [41](#)  
plotTopResults, [40](#)  
plotWf, [41](#)  
pnhyper, [14](#)  
predict.scam, [24](#)  
  
saveWorkbook, [45](#)  
scam, [22](#)  
smoppix, [42](#)  
smoppix-package (smoppix), [42](#)  
sortGp, [42](#)  
splitWindow, [43](#)  
subSampleP, [43](#)  
sund, [44](#)  
  
writeData, [45](#)  
writeToXlsx, [44](#)  
  
Yang, [45](#)