Package 'SITSAR'

November 6, 2026

Title Information Theory, Geometry and Statistics for Synthetic Aperture Radar Data

Version 1.0.0

Description Statistical models, information-theoretic measures and image analysis tools for Synthetic Aperture Radar (SAR) and polarimetric SAR (PolSAR) data. It implements intensity models (Gamma, generalized Gamma and G0I) and matrix models (scaled complex Wishart, G0 polarimetric and the compound truncated Poisson and compound geometric complex Wishart laws), maximum likelihood and expectation-maximization estimation, closed-form Shannon entropies with their asymptotic variances, entropy-based hypothesis tests and change detection, entropy-based image segmentation by (stochastic) expectation-maximization, stochastic distances and a family of divergence-based classifiers (Kullback-Leibler, Jeffreys, Bhattacharyya and Renyi) for Gaussian vector features and complex Wishart matrix features. The main tools are 'R6' classes and the classifiers are available as a 'parsnip' model, so they can be used inside 'tidymodels' workflows. Methods are described in Ferreira, Coelho and Nascimento (2021) <doi:10.1080/01431161.2020.1826060>, Ferreira and Nascimento (2020) <doi:10.1109/JSTARS.2020.2997666>, Ferreira, Nascimento and Frery (2022) <doi:10.3390/rs14205083> and Nascimento, Ferreira and Frery (2023) <doi:10.1007/s11222-023-10298-0>.

License MIT + file LICENSE

URL https://github.com/jodavid/SITSAR, https://jodavid.github.io/SITSAR/

BugReports https://github.com/jodavid/SITSAR/issues

Depends R (>= 4.1.0)

Imports generics, graphics, grDevices, parsnip (>= 1.0.0), R6 (>= 2.5.0), rlang, stats, tibble, utils

Suggests dials, knitr, recipes, rmarkdown, rsample, testthat (>= 3.0.0), tune, workflows, yardstick

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

LazyDataCompression xz

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Jodavid Ferreira [aut, cre] (ORCID:
<https://orcid.org/0000-0002-2131-6464>),
Abraao Nascimento [aut] (ORCID:
<https://orcid.org/0000-0003-2673-219X>)

Maintainer Jodavid Ferreira <jodavid@protonmail.com>

Repository CRAN

Date/Publication 2026-11-06 16:20:02 UTC

Contents

change_detection_metrics . . . 3
change_map . . . 4
CompoundWishart . . . 5
compound_wishart . . . 7
dg0i . . . 9
dg0p . . . 10
dgammasar . . . 11
dggamma . . . 12
distancesg0i . . . 13
distancesg0itable . . . 14
DivergenceClassifier . . . 15
divergences . . . 17
divergence_classifier . . . 19
dwishart_sar . . . 21
entropy . . . 22
EntropySegmenter . . . 23
entropy_map . . . 26
entropy_test . . . 27
G0I . . . 28
g0iestimation . . . 30
G0Polarimetric . . . 31
GammaSAR . . . 32
GGammaSAR . . . 34
kappa_statistics . . . 36
plot_map . . . 37
polsar-format . . . 38
polsar_example_sigma . . . 38
renyi_order . . . 39
SanFrancisco150 . . . 40

SanFrancisco150_gt 40
SARImage 41
SARModel 44
sarmodel-methods 47
sar_features 48
segment_sar 49
WishartSAR 50

Index **52**

`change_detection_metrics`

Accuracy measures for change detection

Description

Computes the usual accuracy measures of binary change maps: detection rate (DR, percentage of true changes that were detected), false alarm rate (FA, percentage of unchanged pixels flagged as changed), and the percentages of false positives (FP) and false negatives (FN) with respect to all pixels.

Usage

```
change_detection_metrics(truth, estimate)
```

Arguments

`truth`	Logical (or 0/1) vector/matrix of true changes.
`estimate`	Logical (or 0/1) vector/matrix of detected changes.

Value

A tibble with `DR`, `FA`, `FP`, `FN` (in percent) and the overall accuracy `OA` (in percent).

Examples

```
truth <- c(TRUE, TRUE, FALSE, FALSE, FALSE)
est <- c(TRUE, FALSE, TRUE, FALSE, FALSE)
change_detection_metrics(truth, est)
```

change_map *Entropy-based change detection between two images*

Description

Applies the entropy test of `entropy_test()` pixel by pixel, using the observations of a moving window in two co-registered images, and returns the map of p-values and the binary change map.

Usage

```
change_map(
  image1,
  image2,
  model = c("gamma", "g0i", "ggamma", "wishart", "g0p"),
  window = 5,
  L = NULL,
  channel = 1,
  level = 0.01
)
```

Arguments

`image1`, `image2`	Co-registered images (SARImage objects or arrays) of the same size.
`model`	One of `"gamma"`, `"g0i"`, `"ggamma"` (intensity models, applied to each channel) or `"wishart"`, `"g0p"` (matrix models, for PolSAR images).
`window`	Odd side of the moving window.
`L`	Known number of looks. If `NULL` (default) it is estimated in every window.
`channel`	Channel used by intensity models.
`level`	Significance level used to declare a change.

Value

A list of class `sitsar_change` with the matrices `statistic`, `p_value` and `change` (logical), and the `level`.

Examples

```
set.seed(1)
a <- matrix(rgammasar(40 * 40, mu = 1, L = 4), 40)
b <- a
b[11:30, 11:30] <- rgammasar(400, mu = 4, L = 4)
cd <- change_map(a, b, model = "gamma", window = 5, L = 4)
cd
plot_map(cd$change, main = "Detected changes")
```

`CompoundWishart` *Compound complex Wishart models (R6)*

Description

R6 class for the compound truncated Poisson (CTPCW) and compound geometric (CGCW) complex Wishart laws of Ferreira, Nascimento and Frery (2022); see `dctpcw()`. Parameters are estimated by the expectation-maximisation (EM) algorithm, treating the number of scatterers N as missing data:

- E-step: $\hat{N}_i = E(N \mid S_i)$;
- M-step: $\hat{\Sigma} = \bar{S}/(L\bar{N})$ and $\hat{\lambda}/(1 - e^{-\hat{\lambda}}) = \bar{N}$ (CTPCW) or $\hat{p} = 1/\bar{N}$ (CGCW).

The Shannon entropy has no closed form and is approximated by Monte Carlo.

Value

An R6::R6Class generator. `CompoundWishart$new()` returns a new `CompoundWishart` object; its methods are described in the Methods section below.

Super class

`SARModel` -> `CompoundWishart`

Public fields

`counting` Counting law: `"poisson"` (zero-truncated) or `"geometric"`.

`iterations` Number of EM iterations of the last fit.

Methods

Public methods:

- `CompoundWishart$new()`
- `CompoundWishart$density()`
- `CompoundWishart$random()`
- `CompoundWishart$fit()`
- `CompoundWishart$entropy()`
- `CompoundWishart$entropy_variance()`
- `CompoundWishart$clone()`

`CompoundWishart$new()`**:** Create a compound Wishart model.

Usage:

```
CompoundWishart$new(
  counting = c("poisson", "geometric"),
  Sigma = diag(3) + (0+0i),
  L = 4,
  lambda = 1,
  p = 0.5
)
```

Arguments:

`counting` `"poisson"` (CTPCW) or `"geometric"` (CGCW).

`Sigma` Hermitian positive definite matrix.

`L` Number of looks of each component.

`lambda, p` Parameter of the counting law.

`CompoundWishart$density()`**:** (Log-)density.

Usage:

```
CompoundWishart$density(x, log = FALSE)
```

Arguments:

`x` PolSAR observations.

`log` Return log-densities?

`CompoundWishart$random()`**:** Simulate PolSAR observations.

Usage:

```
CompoundWishart$random(n, output = "array")
```

Arguments:

`n` Number of observations.

`output` `"array"` or `"vec"`.

`CompoundWishart$fit()`**:** Fit by the EM algorithm.

Usage:

```
CompoundWishart$fit(x, L, tol = 1e-06, max_iter = 500L)
```

Arguments:

`x` PolSAR observations.

`L` Known number of looks of each component.

`tol` Convergence tolerance (relative change of the parameters).

`max_iter` Maximum number of iterations.

`CompoundWishart$entropy()`**:** Monte Carlo approximation of the Shannon entropy.

Usage:

```
CompoundWishart$entropy(nsim = 2000L)
```

Arguments:

`nsim` Number of simulated observations.

`CompoundWishart$entropy_variance()`**:** Not available for compound models.

Usage:

```
CompoundWishart$entropy_variance(n = self$n)
```

Arguments:

`n` Ignored.

`CompoundWishart$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CompoundWishart$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Ferreira, J. A., Nascimento, A. D. C. and Frery, A. C. (2022). PolSAR models with multimodal intensities. *Remote Sensing*, 14(20), 5083. doi:10.3390/rs14205083

Examples

```
set.seed(1)
S <- rctpcw(200, lambda = 1.5, Sigma = polsar_example_sigma("forest"), L = 4)
m <- CompoundWishart$new(counting = "poisson")$fit(S, L = 4)
m$params$lambda
m$iterations
```

`compound_wishart` *Compound complex Wishart distributions*

Description

Density and random generation for the compound truncated Poisson complex Wishart (CTPCW) and compound geometric complex Wishart (CGCW) laws (Ferreira, Nascimento and Frery, 2022). They describe the PolSAR return as a random sum $S = \sum_{i=1}^{N} Z_i$ of independent (unscaled) complex Wishart matrices $Z_i \sim \mathcal{W}_m^C(\Sigma, L)$ with $E(Z_i) = L\Sigma$, where the number of scatterers N follows a zero-truncated Poisson law with mean parameter λ (CTPCW) or a geometric law on $\{1, 2, \ldots\}$ with success probability p (CGCW). Their densities are

$$f(S) = \sum_{k=1}^{\infty} P(N = k) \frac{|S|^{kL-m}}{|\Sigma|^{kL}\Gamma_m(kL)} \exp\{-\mathrm{tr}(\Sigma^{-1}S)\},$$

which are able to describe multimodal intensities. The series is evaluated in log-scale and truncated when the terms become negligible.

Usage

```
dctpcw(x, lambda, Sigma, L, log = FALSE, kmax = 500L)

rctpcw(n, lambda, Sigma, L, output = c("array", "vec"))

dcgcw(x, p, Sigma, L, log = FALSE, kmax = 500L)

rcgcw(n, p, Sigma, L, output = c("array", "vec"))
```

Arguments

x	PolSAR observations: a complex array `m x m x n`, a single Hermitian matrix or an `n x m(m+1)/2` matrix (see `as_polsar_vec()`).
lambda	Mean parameter of the zero-truncated Poisson law ($\lambda > 0$).
Sigma	Hermitian positive definite `m x m` matrix (the mean).
L	Number of looks of each Wishart component ($L > m - 1$).
log	Logical; if `TRUE` the log-density is returned.
kmax	Maximum number of terms of the series.
n	Number of observations.
output	Format of the generated sample: `"array"` (`m x m x n`) or `"vec"` (`n x m(m+1)/2`).
p	Success probability of the geometric law ($0 < p < 1$).

Value

`dctpcw()`/`dcgcw()` return (log-)densities; `rctpcw()`/`rcgcw()` return the simulated sample, with the simulated numbers of components in the attribute `"N"`.

References

Ferreira, J. A., Nascimento, A. D. C. and Frery, A. C. (2022). PolSAR models with multimodal intensities. *Remote Sensing*, 14(20), 5083. doi:10.3390/rs14205083

See Also

CompoundWishart for the R6 interface and EM estimation.

Examples

```
Sigma <- polsar_example_sigma("forest")
set.seed(1)
S <- rctpcw(3, lambda = 2, Sigma = Sigma, L = 4)
attr(S, "N")
dctpcw(S, lambda = 2, Sigma = Sigma, L = 4, log = TRUE)
S2 <- rcgcw(3, p = 0.6, Sigma = Sigma, L = 4)
dcgcw(S2, p = 0.6, Sigma = Sigma, L = 4, log = TRUE)
```

dg0i *The G0I distribution for SAR intensities*

Description

Density, distribution function, quantile function and random generation for the $\mathcal{G}_I^0(\alpha, \gamma, L)$ distribution, the multiplicative model for multilook SAR intensities with reciprocal gamma backscatter and gamma speckle (Frery et al., 1997).

Usage

```
dg0i(x, alpha, gamma, L, log = FALSE)

pg0i(q, alpha, gamma, L, lower.tail = TRUE, log.p = FALSE)

qg0i(p, alpha, gamma, L, lower.tail = TRUE, log.p = FALSE)

rg0i(n, alpha, gamma, L)
```

Arguments

x, q	Vector of quantiles.
alpha	Roughness parameter ($\alpha < 0$).
gamma	Scale parameter ($\gamma > 0$).
L	Number of looks ($L > 0$).
log, log.p	Logical; if TRUE, probabilities/densities are given as logarithms.
lower.tail	Logical; if TRUE (default) probabilities are $P(X \leq x)$, otherwise $P(X > x)$.
p	Vector of probabilities.
n	Number of observations.

Details

The density is

$$f(z) = \frac{L^L \Gamma(L - \alpha)}{\gamma^\alpha \Gamma(-\alpha)\Gamma(L)} \frac{z^{L-1}}{(\gamma + Lz)^{L-\alpha}}, \quad z > 0,$$

with roughness $\alpha < 0$, scale $\gamma > 0$ and number of looks $L > 0$. If $Z \sim \mathcal{G}_I^0(\alpha, \gamma, L)$ then $-\alpha Z/\gamma$ follows a Snedecor F distribution with $2L$ and -2α degrees of freedom, which is used to compute the distribution and quantile functions exactly.

Value

dg0i() gives the density, pg0i() the distribution function, qg0i() the quantile function and rg0i() generates random deviates.

References

Frery, A. C., Muller, H.-J., Yanasse, C. C. F. and Sant'Anna, S. J. S. (1997). A model for extremely heterogeneous clutter. *IEEE Transactions on Geoscience and Remote Sensing*, 35(3), 648–659.

Nascimento, A. D. C., Cintra, R. J. and Frery, A. C. (2010). Hypothesis testing in speckled data with stochastic distances. *IEEE Transactions on Geoscience and Remote Sensing*, 48(1), 373–385.

See Also

G0I for the R6 interface (estimation and entropy).

Examples

```
set.seed(1)
z <- rg0i(5, alpha = -1.5, gamma = 2, L = 8)
dg0i(z, alpha = -1.5, gamma = 2, L = 8)
pg0i(c(0.5, 1, 2), alpha = -1.5, gamma = 2, L = 8)
qg0i(c(0.1, 0.5, 0.9), alpha = -1.5, gamma = 2, L = 8)
```

---

dg0p *The G0 polarimetric distribution*

Description

Density and random generation for the polarimetric $\mathcal{G}_m^0(\alpha, \Sigma, L)$ law (also known as $\mathcal{G}_P^0$), obtained from the multiplicative model $Z = XY$ with $Y \sim \mathcal{W}_m(\Sigma, L)$ and reciprocal gamma texture $X \sim \Gamma^{-1}(-\alpha, -\alpha - 1)$ with unit mean (Freitas et al., 2005):

$$f(Z) = \frac{L^{mL}\Gamma(mL-\alpha)|Z|^{L-m}}{|\Sigma|^L\Gamma_m(L)\Gamma(-\alpha)(-\alpha-1)^\alpha}\left[L\,\mathrm{tr}(\Sigma^{-1}Z) - \alpha - 1\right]^{\alpha-mL},$$

with $\alpha < -1$. When $\alpha \to -\infty$ it converges to the scaled complex Wishart law.

Usage

```
dg0p(x, alpha, Sigma, L, log = FALSE)

rg0p(n, alpha, Sigma, L, output = c("array", "vec"))
```

Arguments

`x`	PolSAR observations: a complex array `m x m x n`, a single Hermitian matrix or an `n x m(m+1)/2` matrix (see `as_polsar_vec()`).
`alpha`	Roughness parameter ($\alpha < -1$).
`Sigma`	Hermitian positive definite `m x m` matrix (the mean).
`L`	Number of looks ($L > m - 1$).
`log`	Logical; if `TRUE` the log-density is returned.
`n`	Number of observations.
`output`	Format of the generated sample: `"array"` (`m x m x n`) or `"vec"` (`n x m(m+1)/2`).

Value

`dg0p()` returns (log-)densities and `rg0p()` the simulated sample.

References

Freitas, C. C., Frery, A. C. and Correia, A. H. (2005). The polarimetric G distribution for SAR data analysis. *Environmetrics*, 16(1), 13–31.

Nascimento, A. D. C., Ferreira, J. A. and Frery, A. C. (2023). Unsupervised segmentation of PolSAR data with complex Wishart and G0m distributions and Shannon entropy. *Statistics and Computing*, 33, 129. doi:10.1007/s11222023102980

See Also

G0Polarimetric for the R6 interface.

Examples

```
Sigma <- polsar_example_sigma("urban")
set.seed(1)
Z <- rg0p(3, alpha = -3, Sigma = Sigma, L = 4)
dg0p(Z, alpha = -3, Sigma = Sigma, L = 4, log = TRUE)
```

`dgammasar` *The Gamma distribution for SAR intensities*

Description

Density, distribution function, quantile function and random generation for the multilook intensity Gamma law parametrised by its mean μ and number of looks L:

$$f(z) = \frac{L^L}{\mu^L \Gamma(L)} z^{L-1} \exp(-Lz/\mu), \quad z > 0.$$

This is the model for homogeneous areas (fully developed speckle).

Usage

```
dgammasar(x, mu, L, log = FALSE)

pgammasar(q, mu, L, lower.tail = TRUE, log.p = FALSE)

qgammasar(p, mu, L, lower.tail = TRUE, log.p = FALSE)

rgammasar(n, mu, L)
```

Arguments

x, q	Vector of quantiles.
mu	Mean ($\mu > 0$).
L	Number of looks ($L > 0$).
log, log.p	Logical; if TRUE, probabilities/densities are given as logarithms.
lower.tail	Logical; if TRUE (default) probabilities are $P(X \leq x)$, otherwise $P(X > x)$.
p	Vector of probabilities.
n	Number of observations.

Value

Density, distribution function, quantile function or random deviates.

See Also

GammaSAR for the R6 interface.

Examples

```
dgammasar(c(0.5, 1, 2), mu = 1, L = 4)
rgammasar(3, mu = 1, L = 4)
```

---

dggamma *The generalized Gamma distribution for SAR intensities*

Description

Density, distribution function, quantile function and random generation for the generalized Gamma (Stacy) law with power $\alpha > 0$, shape $\phi > 0$ and inverse-scale $\mu > 0$:

$$f(z) = \frac{\alpha \mu^{\alpha\phi}}{\Gamma(\phi)} z^{\alpha\phi - 1} \exp\{-(\mu z)^{\alpha}\}, \quad z > 0.$$

For $\alpha = 1$ it reduces to the Gamma law with shape ϕ and rate μ; in SAR applications ϕ plays the role of the number of looks.

Usage

```
dggamma(x, alpha, phi, mu, log = FALSE)

pggamma(q, alpha, phi, mu, lower.tail = TRUE, log.p = FALSE)

qggamma(p, alpha, phi, mu, lower.tail = TRUE, log.p = FALSE)

rggamma(n, alpha, phi, mu)
```

Arguments

x, q	Vector of quantiles.
alpha	Power parameter ($\alpha > 0$).
phi	Shape parameter ($\phi > 0$).
mu	Inverse-scale parameter ($\mu > 0$).
log, log.p	Logical; if TRUE, probabilities/densities are given as logarithms.
lower.tail	Logical; if TRUE (default) probabilities are $P(X \leq x)$, otherwise $P(X > x)$.
p	Vector of probabilities.
n	Number of observations.

Value

Density, distribution function, quantile function or random deviates.

References

Li, H.-C., Hong, W., Wu, Y.-R. and Fan, P.-Z. (2011). On the empirical- statistical modeling of SAR images with generalized gamma distribution. *IEEE Journal of Selected Topics in Signal Processing*, 5(3), 386–397.

See Also

GGammaSAR for the R6 interface.

Examples

```
dggamma(c(0.5, 1, 2), alpha = 1.5, phi = 2, mu = 1)
rggamma(3, alpha = 1.5, phi = 2, mu = 1)
```

---

distancesg0i *Stochastic distances between G0I distributions*

Description

Computes stochastic distances between two $\mathcal{G}_I^0$ laws by numerical integration (Nascimento, Cintra and Frery, 2010).

Usage

```
distancesg0i(
  par_1,
  par_2,
  p = 1/2,
  type = c("KL", "Renyi", "Bhattacharyya", "Hellinger", "Arithmetic-Geometric",
    "Triangular", "Harmonic-Mean")
)
```

Arguments

par_1, par_2	Parameter vectors c(alpha, gamma, L).
p	Order of the Rényi distance, in (0, 1).
type	One of "KL", "Renyi", "Bhattacharyya", "Hellinger", "Arithmetic-Geometric", "Triangular" or "Harmonic-Mean".

Value

An object of class distancesg0i with the parameters, the type and the distance_value.

References

Nascimento, A. D. C., Cintra, R. J. and Frery, A. C. (2010). Hypothesis testing in speckled data with stochastic distances. *IEEE Transactions on Geoscience and Remote Sensing*, 48(1), 373–385.

Examples

```
distancesg0i(c(-1.5, 2, 4), c(-1.5, 2, 4))
res <- distancesg0i(c(-1.5, 2, 8), c(-3, 2, 4), type = "Hellinger")
res
```

---

distancesg0itable *Table of G0I stochastic distances between two image regions*

Description

Fits the $\mathcal{G}_I^0$ law to each intensity channel of two rectangular regions of an image and computes all the stochastic distances of distancesg0i() between them.

Usage

```
distancesg0itable(image, marks1, marks2, p = 1/2, L = NULL)
```

Arguments

image	A SARImage or an array (nrow x ncol x layers).
marks1, marks2	Region limits as c(left, right, up, down), i.e. first and last column, first and last row.
p	Order of the Rényi distance.
L	Known number of looks; estimated when NULL.

Value

An object of class distancesg0itable with the marks and the table of distances (distancestable, channels in rows).

Examples

```
res <- distancesg0itable(SanFrancisco150, c(10, 30, 10, 30),
                         c(10, 30, 120, 140), L = 4)
res
```

`DivergenceClassifier` *Divergence-based classifier for SAR/PolSAR features (R6)*

Description

Implements the family of divergence-based classifiers of Ferreira, Coelho and Nascimento (2021). Each pixel is represented by the model fitted to the observations of a moving window (see `sar_features()`) and each class by the model fitted to its training pixels. A pixel is assigned to the class that minimises the chosen divergence between the two fitted models:

$$\hat{\imath} = \arg\min_{i \in \{1,\dots,g\}} d(\hat{\theta}_h, \hat{\theta}_i).$$

Two families of features are available:

`"gaussian"` multivariate normal laws for the intensity vector, using the columns `mean_*` and `cov_*` of `sar_features()`. The class covariance combines the average local covariance and the covariance of the local means (law of total variance).

`"wishart"` scaled complex Wishart laws for the full PolSAR covariance matrix, using the columns `c11`, `c12_re`, `c12_im`, ... of `sar_features()`. Since all laws share the number of looks, it only rescales the divergences and does not affect the classification.

The classifier is also available as the 'parsnip' model `divergence_classifier()`, for use in 'tidymodels' workflows.

Value

An R6::R6Class generator. `DivergenceClassifier$new()` returns a new `DivergenceClassifier` object; its methods are described in the Methods section below.

Public fields

`family` `"gaussian"` or `"wishart"`.

`divergence` Divergence used (see `divergence_gaussian()`).

`order` Order of the Rényi divergence.

`kind` Form (1-4) of the Rényi divergence.

`classes` Class labels.

`prototypes` List with the fitted model of each class.

`channels` Channels (Gaussian) or matrix order (Wishart) used.

Methods

Public methods:

- DivergenceClassifier$new()
- DivergenceClassifier$fit()
- DivergenceClassifier$distances()
- DivergenceClassifier$predict()
- DivergenceClassifier$print()
- DivergenceClassifier$clone()

`DivergenceClassifier$new()`: Create a classifier.

Usage:

```
DivergenceClassifier$new(
  family = c("gaussian", "wishart"),
  divergence = c("renyi", "kl", "jeffreys", "bhattacharyya"),
  order = 0.5,
  kind = 1
)
```

Arguments:

`family` `"gaussian"` or `"wishart"`.

`divergence` `"kl"`, `"jeffreys"`, `"bhattacharyya"` or `"renyi"`.

`order` Order of the Rényi divergence, in (0, 1).

`kind` Form of the Rényi divergence (1 to 4).

`DivergenceClassifier$fit()`: Fit the class prototypes.

Usage:

```
DivergenceClassifier$fit(x, y)
```

Arguments:

`x` Data frame of features (see `sar_features()`); other columns are ignored.

`y` Factor (or vector) of class labels.

`DivergenceClassifier$distances()`: Divergence between every row of `x` and every class.

Usage:

```
DivergenceClassifier$distances(x)
```

Arguments:

`x` Data frame of features.

`DivergenceClassifier$predict()`: Predict classes, pseudo-probabilities or distances.

Usage:

```
DivergenceClassifier$predict(x, type = c("class", "prob", "distance"))
```

Arguments:

`x` Data frame of features.

`type` `"class"`, `"prob"` (softmax of minus the divergences) or `"distance"`.

`DivergenceClassifier$print()`: Print the classifier.

Usage:

```
DivergenceClassifier$print(...)
```

Arguments:

`...` Ignored.

`DivergenceClassifier$clone()`: The objects of this class are cloneable with this method.

Usage:

```
DivergenceClassifier$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Ferreira, J. A., Coelho, H. and Nascimento, A. D. C. (2021). A family of divergence-based classifiers for polarimetric synthetic aperture radar (PolSAR) imagery vector and matrix features. *International Journal of Remote Sensing*, 42(4), 1201–1229. doi:10.1080/01431161.2020.1826060

Examples

```
img <- SARImage$new(SanFrancisco150)$crop(1:80, 1:80)
feats <- sar_features(img, window = 5)
feats$class <- factor(as.vector(SanFrancisco150_gt[1:80, 1:80]),
                      labels = c("ocean", "forest", "urban"))
set.seed(1)
train <- feats[sample(nrow(feats), 300), ]

clf <- DivergenceClassifier$new(family = "wishart", divergence = "renyi",
                                order = 0.5, kind = 3)
clf$fit(train, train$class)
pred <- clf$predict(feats)
mean(pred == feats$class)
head(clf$predict(feats, type = "distance"))
```

`divergences` *Divergences between Gaussian and complex Wishart models*

Description

Closed-form divergences between two multivariate normal laws $N_p(\mu_1, \Sigma_1)$ and $N_p(\mu_2, \Sigma_2)$, and between two scaled complex Wishart laws $\mathcal{W}_m(\Sigma_1, L)$ and $\mathcal{W}_m(\Sigma_2, L)$ with a common number of looks. They are the building blocks of the divergence-based classifiers of Ferreira, Coelho and Nascimento (2021).

Usage

```
divergence_gaussian(
  mu1,
  Sigma1,
  mu2,
  Sigma2,
  type = c("kl", "jeffreys", "bhattacharyya", "renyi"),
  order = 0.5,
  kind = 1
)

divergence_wishart(
  Sigma1,
  Sigma2,
  L,
  type = c("kl", "jeffreys", "bhattacharyya", "renyi"),
  order = 0.5,
  kind = 1
)
```

Arguments

`mu1`, `mu2`	Mean vectors.
`Sigma1`, `Sigma2`	Covariance matrices (real symmetric for Gaussian laws, Hermitian for Wishart laws).
`type`	Divergence: `"kl"`, `"jeffreys"`, `"bhattacharyya"` or `"renyi"`.
`order`	Order $\alpha \in (0, 1)$ of the Rényi divergence.
`kind`	Form of the Rényi divergence (1 to 4).
`L`	Number of looks (common to both Wishart laws).

Details

Available measures (`type`):

`"kl"` Kullback-Leibler divergence $D_{KL}(\theta_1 \| \theta_2)$.

`"jeffreys"` Jeffreys divergence $D_{KL}(\theta_1 \| \theta_2) + D_{KL}(\theta_2 \| \theta_1)$.

`"bhattacharyya"` Bhattacharyya distance $-\log \int \sqrt{f_1 f_2}$.

`"renyi"` Rényi divergence of order α (`order`), in one of four forms (`kind`): 1, $D_R^\alpha(\theta_1 \| \theta_2)$; 2, $D_R^\alpha(\theta_2 \| \theta_1)$; 3, the symmetrised mean $[D_R^\alpha(\theta_1 \| \theta_2) + D_R^\alpha(\theta_2 \| \theta_1)]/2$; and 4, $c \log\{[K(\theta_1 \| \theta_2) + K(\theta_2 \| \theta_1)]/2\}$, where $K(\theta_1 \| \theta_2) = \int f_1^\alpha f_2^{1-\alpha}$.

For Gaussian laws the Rényi divergence uses the normalisation of Pardo (2006), $c = 1/[\alpha(\alpha - 1)]$, which gives

$$D_R^\alpha = \frac{1}{2} \Delta^\top M^{-1} \Delta - \frac{1}{2\alpha(\alpha - 1)} \log \frac{|M|}{|\Sigma_1|^{1-\alpha} |\Sigma_2|^\alpha},$$

with $\Delta = \mu_1 - \mu_2$ and $M = \alpha\Sigma_2 + (1-\alpha)\Sigma_1$. For Wishart laws the normalisation of Frery, Nascimento and Cintra (2014), $c = 1/(\alpha - 1)$, is used:

$$D_R^{\alpha} = \frac{L}{\alpha - 1} \log \frac{|(\alpha\Sigma_1^{-1} + (1-\alpha)\Sigma_2^{-1})^{-1}|}{|\Sigma_1|^{\alpha}|\Sigma_2|^{1-\alpha}}.$$

Both converge to the Kullback-Leibler divergence when $\alpha \to 1$.

Value

A single non-negative number.

References

Ferreira, J. A., Coelho, H. and Nascimento, A. D. C. (2021). A family of divergence-based classifiers for polarimetric synthetic aperture radar (PolSAR) imagery vector and matrix features. *International Journal of Remote Sensing*, 42(4), 1201–1229. doi:10.1080/01431161.2020.1826060

Frery, A. C., Nascimento, A. D. C. and Cintra, R. J. (2014). Analytic expressions for stochastic distances between relaxed complex Wishart distributions. *IEEE Transactions on Geoscience and Remote Sensing*, 52(2), 1213–1226.

Pardo, L. (2006). *Statistical Inference Based on Divergence Measures*. Chapman & Hall/CRC.

Examples

```
S1 <- diag(3); S2 <- diag(c(2, 1, 0.5))
divergence_gaussian(c(0, 0, 0), S1, c(1, 0, 0), S2, type = "kl")
divergence_gaussian(c(0, 0, 0), S1, c(1, 0, 0), S2, type = "renyi",
                    order = 0.5, kind = 3)

W1 <- polsar_example_sigma("forest"); W2 <- polsar_example_sigma("urban")
divergence_wishart(W1, W2, L = 4, type = "jeffreys")
divergence_wishart(W1, W2, L = 4, type = "renyi", order = 0.9)
```

`divergence_classifier` *Divergence-based classifier as a 'parsnip' model*

Description

`divergence_classifier()` defines the divergence-based classifier of Ferreira, Coelho and Nascimento (2021) as a 'parsnip' model specification, so it can be used with 'workflows', 'recipes', 'rsample', 'tune' and 'yardstick'. The engine selects the family of features: `"gaussian"` (multivariate normal intensity features, the default) or `"wishart"` (complex Wishart PolSAR matrix features). The predictors must be in the format of `sar_features()`; extra columns such as `row`/`col` are ignored by the engine (it is good practice to give them an `"id"` role in a recipe).

Usage

```
divergence_classifier(
  mode = "classification",
  divergence = NULL,
  order = NULL,
  kind = NULL,
  engine = "gaussian"
)

## S3 method for class 'divergence_classifier'
update(
  object,
  parameters = NULL,
  divergence = NULL,
  order = NULL,
  kind = NULL,
  fresh = FALSE,
  ...
)
```

Arguments

mode	A single character string for the type of model. The only possible value is "classification".
divergence	Divergence: "renyi" (default), "kl", "jeffreys" or "bhattacharyya".
order	Order of the Rényi divergence, in (0, 1). Tunable with renyi_order().
kind	Form of the Rényi divergence (1 to 4); see divergences.
engine	"gaussian" or "wishart".
object	A divergence_classifier model specification.
parameters	A 1-row tibble or named list with the main parameters.
fresh	Replace the arguments instead of updating them?
...	Not used.

Value

A model specification object with classes divergence_classifier and model_spec.

See Also

DivergenceClassifier, sar_features()

Examples

```
spec <- divergence_classifier(divergence = "renyi", order = 0.5) |>
  parsnip::set_engine("wishart")
spec
```

```
feats <- sar_features(SARImage$new(SanFrancisco150)$crop(1:60, 1:60), window = 5)
feats$class <- factor(as.vector(SanFrancisco150_gt[1:60, 1:60]))
fit <- parsnip::fit(spec, class ~ ., data = feats)
head(predict(fit, feats))
head(predict(fit, feats, type = "prob"))
```

`dwishart_sar` *The scaled complex Wishart distribution*

Description

Density and random generation for the scaled complex Wishart law $\mathcal{W}_m(\Sigma, L)$, the reference model for multilook PolSAR covariance (coherency) matrices in homogeneous areas:

$$f(Z) = \frac{L^{mL}|Z|^{L-m}}{|\Sigma|^{L}\Gamma_m(L)} \exp\{-L\,\mathrm{tr}(\Sigma^{-1}Z)\},$$

where $\Gamma_m(L) = \pi^{m(m-1)/2}\prod_{i=0}^{m-1}\Gamma(L-i)$ and $E(Z) = \Sigma$.

Usage

```
dwishart_sar(x, Sigma, L, log = FALSE)

rwishart_sar(n, Sigma, L, output = c("array", "vec"))
```

Arguments

`x`	PolSAR observations: a complex array `m x m x n`, a single Hermitian matrix or an `n x m(m+1)/2` matrix (see `as_polsar_vec()`).
`Sigma`	Hermitian positive definite `m x m` matrix (the mean).
`L`	Number of looks ($L > m - 1$).
`log`	Logical; if `TRUE` the log-density is returned.
`n`	Number of observations.
`output`	Format of the generated sample: `"array"` (`m x m x n`) or `"vec"` (`n x m(m+1)/2`).

Details

Random matrices are generated with the complex Bartlett decomposition, so non-integer numbers of looks $L > m - 1$ are allowed.

Value

`dwishart_sar()` returns a vector of (log-)densities; `rwishart_sar()` returns the simulated sample.

References

Goodman, N. R. (1963). Statistical analysis based on a certain multivariate complex Gaussian distribution. *The Annals of Mathematical Statistics*, 34(1), 152–177.

Lee, J.-S., Grunes, M. R. and Kwok, R. (1994). Classification of multi-look polarimetric SAR imagery based on complex Wishart distribution. *International Journal of Remote Sensing*, 15(11), 2299–2311.

See Also

WishartSAR for the R6 interface.

Examples

```
Sigma <- polsar_example_sigma("forest")
set.seed(1)
Z <- rwishart_sar(3, Sigma, L = 4)
dim(Z)
dwishart_sar(Z, Sigma, L = 4, log = TRUE)
```

---

entropy *Shannon entropies of SAR and PolSAR models*

Description

Closed-form Shannon entropies $H = E\{-\log f(Z)\}$ for the models used in SAR (intensity) and PolSAR (matrix) image analysis. All functions are vectorised in the scalar parameters.

Usage

```
entropy_gamma(mu, L)

entropy_g0i(alpha, gamma, L)

entropy_ggamma(alpha, phi, mu)

entropy_wishart(Sigma, L)

entropy_g0p(alpha, Sigma, L)
```

Arguments

`mu`	Mean of the Gamma law, or inverse scale of the generalized Gamma law.
`L`	Number of looks.
`alpha`	Roughness (G0I, G0 polarimetric) or power (generalized Gamma) parameter.
`gamma`	Scale parameter of the G0I law.
`phi`	Shape parameter of the generalized Gamma law.
`Sigma`	Hermitian positive definite covariance matrix.

Details

Gamma $H = L - \log L + \log \mu + \log \Gamma(L) + (1-L)\psi^{(0)}(L)$.

G0I (Ferreira and Nascimento, 2020) $H = -\log \frac{L^L \Gamma(L-\alpha)}{\Gamma(-\alpha)\gamma^\alpha \Gamma(L)} + (1-L)[\psi^{(0)}(L) - \log L + \log\gamma - \psi^{(0)}(-\alpha)] + (L-\alpha)[\psi^{(0)}(L-\alpha) + \log\gamma - \psi^{(0)}(-\alpha)]$.

Generalized Gamma $H = -\log\alpha + \log\Gamma(\phi) + \phi - (\phi - 1/\alpha)\psi^{(0)}(\phi) - \log\mu$.

Scaled complex Wishart (Nascimento, Ferreira and Frery, 2023) $H = m\log|\Sigma| - m^2\log L + \log\Gamma_m(L) + mL + (m-L)\psi_m^{(0)}(L)$, with $\psi_m^{(0)}(L) = \sum_{i=0}^{m-1}\psi^{(0)}(L-i)$.

G0 polarimetric (Nascimento, Ferreira and Frery, 2023) obtained from $E\log|Z| = m[\log(-\alpha - 1) - \psi^{(0)}(-\alpha)] + \log|\Sigma| - m\log L + \psi_m^{(0)}(L)$ and $E\log[L\,\mathrm{tr}(\Sigma^{-1}Z) - \alpha - 1] = \log(-\alpha - 1) + \psi^{(0)}(mL-\alpha) - \psi^{(0)}(-\alpha)$.

Value

Numeric vector of entropies.

References

Ferreira, J. A. and Nascimento, A. D. C. (2020). Shannon entropy for the G0I model: a new segmentation approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13, 2547–2553. doi:10.1109/JSTARS.2020.2997666

Nascimento, A. D. C., Ferreira, J. A. and Frery, A. C. (2023). Unsupervised segmentation of PolSAR data with complex Wishart and G0m distributions and Shannon entropy. *Statistics and Computing*, 33, 129. doi:10.1007/s11222023102980

Examples

```
entropy_gamma(mu = 1, L = 4)
entropy_g0i(alpha = -3, gamma = 2, L = 4)
entropy_ggamma(alpha = 1, phi = 4, mu = 4) # equals the Gamma entropy
S <- polsar_example_sigma("forest")
entropy_wishart(S, L = 4)
entropy_g0p(alpha = -5, Sigma = S, L = 4)
```

EntropySegmenter *Entropy-based segmentation by (stochastic) EM (R6)*

Description

Unsupervised segmentation of SAR/PolSAR images from maps of estimated Shannon entropies (Ferreira and Nascimento, 2020; Nascimento, Ferreira and Frery, 2023). By the asymptotic normality of the entropy estimators, the entropy vector of a pixel of the g-th region is approximately Gaussian, so the image is modelled as a finite mixture of Gaussian laws (with diagonal covariance matrices) fitted by the expectation-maximisation algorithm (`method = "em"`) or by its stochastic version (`method = "sem"`), in which the E-step is followed by a random allocation of every pixel to a class drawn from its posterior probabilities. Pixels are then labelled by the maximum a posteriori rule.

Value

An R6::R6Class generator. `EntropySegmenter$new()` returns a new `EntropySegmenter` object; its methods are described in the Methods section below.

Public fields

`k` Number of classes.

`method` `"em"` or `"sem"`.

`weights` Mixture weights.

`means` Matrix (`k x features`) of class means.

`variances` Matrix (`k x features`) of class variances.

`posterior` Matrix of posterior probabilities.

`labels` Integer vector of labels (classes ordered by the mean of the first feature).

`loglik` Trace of the log-likelihood.

`iterations` Number of iterations.

`converged` Logical convergence flag.

`dim` Spatial dimensions when fitted to a map (`NULL` otherwise).

Methods

Public methods:

- `EntropySegmenter$new()`
- `EntropySegmenter$fit()`
- `EntropySegmenter$predict()`
- `EntropySegmenter$label_map()`
- `EntropySegmenter$print()`
- `EntropySegmenter$clone()`

`EntropySegmenter$new()`: Create a segmenter.

Usage:

```
EntropySegmenter$new(
  k = 2,
  method = c("em", "sem"),
  max_iter = 200L,
  tol = 1e-08
)
```

Arguments:

`k` Number of classes.

`method` `"em"` or `"sem"`.

`max_iter` Maximum number of iterations.

`tol` Tolerance for the relative change of the log-likelihood.

`EntropySegmenter$fit()`: Fit the mixture.

Usage:

```
EntropySegmenter$fit(x, init = NULL, is_map = TRUE)
```

Arguments:

`x` Entropy features: a matrix (one map, or pixels in rows and features in columns when `is_map = FALSE`), an array `nrow x ncol x features` (several maps), or a numeric vector.

`init` Initial labels (integer vector) or `NULL` for k-means.

`is_map` Treat a matrix `x` as a single map? Default `TRUE`.

`EntropySegmenter$predict()`: Labels of new observations (maximum a posteriori).

Usage:

```
EntropySegmenter$predict(x, is_map = TRUE)
```

Arguments:

`x` Entropy features in the same format used in `$fit()`.

`is_map` Treat a matrix `x` as a single map?

`EntropySegmenter$label_map()`: Labels as a matrix with the spatial dimensions of the fitted map.

Usage:

```
EntropySegmenter$label_map()
```

`EntropySegmenter$print()`: Print a summary.

Usage:

```
EntropySegmenter$print(...)
```

Arguments:

`...` Ignored.

`EntropySegmenter$clone()`: The objects of this class are cloneable with this method.

Usage:

```
EntropySegmenter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Ferreira, J. A. and Nascimento, A. D. C. (2020). Shannon entropy for the G0I model: a new segmentation approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13, 2547–2553. doi:10.1109/JSTARS.2020.2997666

Nascimento, A. D. C., Ferreira, J. A. and Frery, A. C. (2023). Unsupervised segmentation of PolSAR data with complex Wishart and G0m distributions and Shannon entropy. *Statistics and Computing*, 33, 129. doi:10.1007/s11222023102980

Examples

```
H <- entropy_map(SanFrancisco150, model = "wishart", window = 7, L = 4)
set.seed(1)
seg <- EntropySegmenter$new(k = 3)$fit(H)
seg
plot_map(seg$label_map(), main = "Entropy-based segmentation")
kappa_statistics(SanFrancisco150_gt, seg$label_map())
```

`entropy_map` *Shannon entropy maps*

Description

Estimates, at every pixel, the parameters of a SAR/PolSAR model with the observations inside a square moving window and returns the map of Shannon entropies (Ferreira and Nascimento, 2020; Nascimento, Ferreira and Frery, 2023). Optionally, the map of asymptotic variances of the entropy estimators is returned as well.

Usage

```
entropy_map(
  image,
  model = c("gamma", "g0i", "ggamma", "wishart", "g0p"),
  window = 7,
  L = NULL,
  channels = NULL,
  variance = FALSE
)
```

Arguments

`image`	A SARImage or an array accepted by `SARImage$new()`.
`model`	One of `"gamma"`, `"g0i"`, `"ggamma"` (intensity models, applied to each channel) or `"wishart"`, `"g0p"` (matrix models, for PolSAR images).
`window`	Odd side of the moving window.
`L`	Known number of looks. If `NULL` (default) it is estimated in every window.
`channels`	Channels used by intensity models (default: all).
`variance`	Also return the variance map?

Details

For the Gamma and Wishart models the computations are fully vectorised (local sufficient statistics are obtained from integral images), so the maps are computed in a fraction of a second. The G0I, generalized Gamma and G0 polarimetric models require a numerical maximisation per pixel and are therefore slower (use small images or crops to explore them).

Value

A matrix (one channel or matrix model) or an array `nrow x ncol x channels` of entropies. If `variance = TRUE`, a list with elements `entropy`, `variance` and `n` (window sizes).

References

Ferreira, J. A. and Nascimento, A. D. C. (2020). Shannon entropy for the G0I model: a new segmentation approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13, 2547–2553. doi:10.1109/JSTARS.2020.2997666

Nascimento, A. D. C., Ferreira, J. A. and Frery, A. C. (2023). Unsupervised segmentation of PolSAR data with complex Wishart and G0m distributions and Shannon entropy. *Statistics and Computing*, 33, 129. doi:10.1007/s11222023102980

Examples

```
img <- SARImage$new(SanFrancisco150)
H <- entropy_map(img, model = "wishart", window = 7, L = 4)
dim(H)
plot_map(H, main = "Wishart entropy")

Hg <- entropy_map(img, model = "gamma", window = 7, channels = "HV")

small <- img$crop(1:15, 1:15)
Hi <- entropy_map(small, model = "g0i", window = 5, L = 4, channels = 1)
```

---

entropy_test *Entropy-based test for two SAR samples*

Description

Tests whether two samples come from the same distribution by comparing their estimated Shannon entropies. Under the null hypothesis of equal entropies, and by the asymptotic normality of the maximum likelihood estimator of the entropy,

$$S = \frac{(\hat{H}_1 - \hat{H}_2)^2}{\widehat{\mathrm{Var}}(\hat{H}_1) + \widehat{\mathrm{Var}}(\hat{H}_2)}$$

follows asymptotically a χ^2_1 distribution. This statistic is the basis of the entropy-based change detectors of the thesis of J. A. Ferreira (2021).

Usage

```
entropy_test(
  x,
  y,
  model = c("gamma", "g0i", "ggamma", "wishart", "g0p"),
  L = NULL
)
```

Arguments

x, y	Samples: numeric vectors of intensities (intensity models) or PolSAR observations (matrix models).
model	One of "gamma", "g0i", "ggamma", "wishart", "g0p".
L	Known number of looks; estimated when NULL.

Value

An object of class htest.

References

Ferreira, J. A. (2021). *Entropias e somas compostas para matrizes estocásticas aplicadas a sistemas PolSAR*. PhD thesis, Universidade Federal de Pernambuco.

Examples

```
set.seed(1)
x <- rg0i(200, alpha = -3, gamma = 2, L = 4)
y <- rg0i(200, alpha = -3, gamma = 2, L = 4)
z <- rg0i(200, alpha = -1.5, gamma = 1, L = 4)
entropy_test(x, y, model = "g0i", L = 4)
entropy_test(x, z, model = "g0i", L = 4)$p.value
```

G0I *G0I model for SAR intensities (R6)*

Description

R6 class for the $\mathcal{G}_I^0(\alpha, \gamma, L)$ law (see dg0i()) with maximum likelihood estimation (analytic score, quasi-Newton optimisation on a transformed scale with several starting points), closed-form Shannon entropy (Ferreira and Nascimento, 2020) and the asymptotic variance of its estimator.

Value

An R6::R6Class generator. G0I$new() returns a new G0I object; its methods are described in the Methods section below.

Super class

SARModel -> G0I

Methods

Public methods:

- G0I$new()
- G0I$density()
- G0I$random()
- G0I$fit()
- G0I$entropy()
- G0I$clone()

`G0I$new()`: Create a G0I model.

Usage:

```
G0I$new(alpha = -2, gamma = 1, L = 1)
```

Arguments:

`alpha` Roughness (< 0).

`gamma` Scale (> 0).

`L` Number of looks (> 0).

`G0I$density()`: (Log-)density.

Usage:

```
G0I$density(x, log = FALSE)
```

Arguments:

`x` Numeric vector of intensities.

`log` Return log-densities?

`G0I$random()`: Simulate intensities.

Usage:

```
G0I$random(n)
```

Arguments:

`n` Number of observations.

`G0I$fit()`: Fit by maximum likelihood.

Usage:

```
G0I$fit(x, L = NULL, start = NULL)
```

Arguments:

`x` Numeric vector of positive intensities.

`L` Known number of looks. If `NULL` (default) it is estimated.

`start` Optional named vector of starting values (`alpha`, `gamma` and, if estimated, `L`).

`G0I$entropy()`: Shannon entropy.

Usage:

```
G0I$entropy()
```

`G0I$clone()`: The objects of this class are cloneable with this method.

Usage:

```
G0I$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Ferreira, J. A. and Nascimento, A. D. C. (2020). Shannon entropy for the G0I model: a new segmentation approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13, 2547–2553. doi:10.1109/JSTARS.2020.2997666

Examples

```
set.seed(1)
z <- rg0i(1000, alpha = -3, gamma = 2, L = 4)
m <- G0I$new()$fit(z, L = 4)
m
m$entropy_variance()
G0I$new()$fit(z)$params # L estimated
```

`g0iestimation` *Estimation of G0I parameters*

Description

Maximum likelihood estimation of the $\mathcal{G}_I^0$ parameters. This is a convenience wrapper around G0I`$fit()`, kept for compatibility with previous versions of the package.

Usage

```
g0iestimation(Theta, x, L_fixed = FALSE)
```

Arguments

`Theta`	Initial values `c(alpha, gamma, L)`. When `L_fixed = TRUE`, `Theta[3]` is the known number of looks.
`x`	Vector of observations.
`L_fixed`	Logical; if `TRUE`, L is held fixed at `Theta[3]`.

Value

Named vector `c(alpha, gamma, L)` of estimates.

Examples

```
set.seed(1)
y <- rg0i(1000, alpha = -1.5, gamma = 2, L = 8)
g0iestimation(c(-1, 1, 1), y)
g0iestimation(c(-1, 1, 8), y, L_fixed = TRUE)
```

`G0Polarimetric` *G0 polarimetric model for PolSAR data (R6)*

Description

R6 class for the $\mathcal{G}_m^0(\alpha, \Sigma, L)$ law (see `dg0p()`). Following Nascimento, Ferreira and Frery (2023), Σ is estimated by the sample mean (since $E(Z) = \Sigma$) and (α, L) by maximising the likelihood (analytic score). The variance of the entropy estimator combines the inverse observed information of (α, L) with a moment estimate of the variance of $m \log |\hat{\Sigma}|$.

Value

An R6::R6Class generator. `G0Polarimetric$new()` returns a new `G0Polarimetric` object; its methods are described in the Methods section below.

Super class

`SARModel` -> `G0Polarimetric`

Methods

Public methods:

- `G0Polarimetric$new()`
- `G0Polarimetric$density()`
- `G0Polarimetric$random()`
- `G0Polarimetric$fit()`
- `G0Polarimetric$entropy()`
- `G0Polarimetric$clone()`

`G0Polarimetric$new()`: Create a G0 polarimetric model.

Usage:

```
G0Polarimetric$new(alpha = -3, Sigma = diag(3) + (0+0i), L = 4)
```

Arguments:

`alpha` Roughness (< -1).

`Sigma` Hermitian positive definite matrix.

`L` Number of looks.

`G0Polarimetric$density()`: (Log-)density.

Usage:

```
G0Polarimetric$density(x, log = FALSE)
```

Arguments:

`x` PolSAR observations.

`log` Return log-densities?

`G0Polarimetric$random()`: Simulate PolSAR observations.

Usage:

```
G0Polarimetric$random(n, output = "array")
```

Arguments:

`n` Number of observations.

`output` `"array"` or `"vec"`.

`G0Polarimetric$fit()`: Fit by maximum likelihood.

Usage:

```
G0Polarimetric$fit(x, L = NULL)
```

Arguments:

`x` PolSAR observations.

`L` Known number of looks. If `NULL` (default) it is estimated.

`G0Polarimetric$entropy()`: Shannon entropy.

Usage:

```
G0Polarimetric$entropy()
```

`G0Polarimetric$clone()`: The objects of this class are cloneable with this method.

Usage:

```
G0Polarimetric$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
set.seed(1)
Z <- rg0p(400, alpha = -4, Sigma = polsar_example_sigma("urban"), L = 4)
m <- G0Polarimetric$new()$fit(Z, L = 4)
m
m$entropy_variance()
```

`GammaSAR` *Gamma model for SAR intensities (R6)*

Description

R6 class for the multilook Gamma law $\Gamma(\mu, L)$; see `dgammasar()`. The maximum likelihood estimator of μ is the sample mean and L solves $\log L - \psi^{(0)}(L) = \log \bar{z} - \overline{\log z}$.

Value

An R6::R6Class generator. `GammaSAR$new()` returns a new `GammaSAR` object; its methods are described in the Methods section below.

Super class

SARModel -> GammaSAR

Methods

Public methods:

- GammaSAR$new()
- GammaSAR$density()
- GammaSAR$random()
- GammaSAR$fit()
- GammaSAR$entropy()
- GammaSAR$entropy_variance()
- GammaSAR$clone()

GammaSAR$new(): Create a Gamma model.

Usage:

```
GammaSAR$new(mu = 1, L = 1)
```

Arguments:

mu Mean.

L Number of looks.

GammaSAR$density(): (Log-)density.

Usage:

```
GammaSAR$density(x, log = FALSE)
```

Arguments:

x Numeric vector of intensities.

log Return log-densities?

GammaSAR$random(): Simulate intensities.

Usage:

```
GammaSAR$random(n)
```

Arguments:

n Number of observations.

GammaSAR$fit(): Fit by maximum likelihood.

Usage:

```
GammaSAR$fit(x, L = NULL)
```

Arguments:

x Numeric vector of positive intensities.

L Known number of looks. If NULL (default) it is estimated.

GammaSAR$entropy(): Shannon entropy.

Usage:

```
GammaSAR$entropy()
```

`GammaSAR$entropy_variance()`: Asymptotic variance of the entropy estimator (closed form).

Usage:

```
GammaSAR$entropy_variance(n = self$n)
```

Arguments:

`n` Sample size.

`GammaSAR$clone()`: The objects of this class are cloneable with this method.

Usage:

```
GammaSAR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
set.seed(1)
z <- rgammasar(500, mu = 2, L = 4)
m <- GammaSAR$new()$fit(z)
m
m$entropy()
m$entropy_variance()
generics::tidy(m)
```

`GGammaSAR` *Generalized Gamma model for SAR intensities (R6)*

Description

R6 class for the generalized Gamma law (see `dggamma()`). Estimation is carried out by profiling: for a given power α, the maximum likelihood estimates of ϕ and μ are available in closed form (up to a one-dimensional root), so the profile likelihood is maximised over α only.

Value

An R6::R6Class generator. `GGammaSAR$new()` returns a new `GGammaSAR` object; its methods are described in the Methods section below.

Super class

`SARModel` -> `GGammaSAR`

Methods

Public methods:

- GGammaSAR$new()
- GGammaSAR$density()
- GGammaSAR$random()
- GGammaSAR$fit()
- GGammaSAR$entropy()
- GGammaSAR$clone()

`GGammaSAR$new()`: Create a generalized Gamma model.

Usage:

```
GGammaSAR$new(alpha = 1, phi = 1, mu = 1)
```

Arguments:

`alpha` Power (> 0).

`phi` Shape (> 0).

`mu` Inverse scale (> 0).

`GGammaSAR$density()`: (Log-)density.

Usage:

```
GGammaSAR$density(x, log = FALSE)
```

Arguments:

`x` Numeric vector of intensities.

`log` Return log-densities?

`GGammaSAR$random()`: Simulate intensities.

Usage:

```
GGammaSAR$random(n)
```

Arguments:

`n` Number of observations.

`GGammaSAR$fit()`: Fit by (profile) maximum likelihood.

Usage:

```
GGammaSAR$fit(x, phi = NULL)
```

Arguments:

`x` Numeric vector of positive intensities.

`phi` Known shape (number of looks). If `NULL` it is estimated.

`GGammaSAR$entropy()`: Shannon entropy.

Usage:

```
GGammaSAR$entropy()
```

`GGammaSAR$clone()`: The objects of this class are cloneable with this method.

Usage:

```
GGammaSAR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
set.seed(1)
z <- rggamma(1000, alpha = 1.5, phi = 3, mu = 2)
m <- GGammaSAR$new()$fit(z)
m
```

`kappa_statistics` *Kappa coefficient and overall accuracy*

Description

Computes Cohen's kappa coefficient of agreement, its asymptotic variance (Congalton and Green, 2009) and the overall accuracy from a confusion matrix or from two vectors (or matrices) of labels. Unlike `yardstick::kap()`, the variance of the estimator is also returned, which allows comparing classifiers with Z tests.

Usage

```
kappa_statistics(truth, estimate = NULL, match = FALSE)
```

Arguments

`truth`	Either a square confusion matrix (estimates in rows, truth in columns) or a vector/matrix of reference labels.
`estimate`	Vector/matrix of estimated labels (when `truth` is not a confusion matrix).
`match`	Relabel the estimated classes to maximise agreement?

Details

Labels of unsupervised methods are arbitrary: use `match = TRUE` to relabel the estimated classes so that they best agree with the truth (an exhaustive search over permutations for up to 8 classes).

Value

A tibble with `kappa`, `variance`, `std_error`, `overall_accuracy` and the number of observations `n`.

References

Congalton, R. G. and Green, K. (2009). *Assessing the Accuracy of Remotely Sensed Data: Principles and Practices*. CRC Press.

Examples

```
truth <- c(1, 1, 2, 2, 3, 3, 3)
est <- c(1, 2, 2, 2, 3, 3, 1)
kappa_statistics(truth, est)
kappa_statistics(table(est, truth))
```

`plot_map` *Plot a map*

Description

Displays a matrix as an image with the first row on top: continuous maps (e.g. entropy maps) use a sequential palette and integer/factor maps (e.g. segmentation or classification results) use a qualitative one.

Usage

```
plot_map(x, palette = NULL, main = NULL, legend = FALSE)
```

Arguments

`x`	A numeric, integer, logical or factor matrix.
`palette`	Optional vector of colours.
`main`	Title.
`legend`	Add a legend for discrete maps?

Value

Invisibly, `x`.

Examples

```
plot_map(SanFrancisco150_gt, main = "Ground truth", legend = TRUE)
plot_map(matrix(rnorm(100), 10), main = "Noise")
```

polsar-format *Convert between PolSAR storage formats*

Description

PolSAR observations are Hermitian positive definite matrices. `SITSAR` accepts them either as a complex array of dimension `m x m x n` or as an `n x p` complex matrix in which each row stores the $p = m(m+1)/2$ distinct entries in the order $C_{11}, \ldots, C_{mm}$ (the intensities) followed by the upper triangle $C_{12}, C_{13}, \ldots, C_{23}, \ldots$. For $m = 3$ this is the usual layer order `HH, HV, VV, HH.HV*, HH.VV*, HV.VV*` of the SanFrancisco150 image.

Usage

```
as_polsar_array(x)

as_polsar_vec(x)
```

Arguments

`x`	A complex array (`m x m x n`), a single `m x m` matrix, or an `n x p` complex matrix (or vector of length `p`).

Value

`as_polsar_array()` returns a complex array `m x m x n`; `as_polsar_vec()` returns an `n x p` complex matrix.

Examples

```
S <- diag(3) + 0i
S[1, 2] <- 0.2 + 0.1i
S[2, 1] <- Conj(S[1, 2])
v <- as_polsar_vec(S)
v
as_polsar_array(v)[, , 1]
```

polsar_example_sigma *Example PolSAR covariance matrices*

Description

Typical 3×3 covariance matrices (scaled to unit magnitude) for urban, forest and pasture areas, taken from the simulation studies of the authors' dissertation and thesis. They are handy to simulate PolSAR data.

Usage

```
polsar_example_sigma(type = c("urban", "forest", "pasture"))
```

Arguments

type	One of "urban", "forest" or "pasture".

Value

A complex Hermitian positive definite 3×3 matrix.

Examples

```
polsar_example_sigma("forest")
```

---

renyi_order *Tuning parameters for the divergence classifier*

Description

'dials' parameter objects for divergence_classifier(): renyi_order() for the order of the Rényi divergence and divergence_type() for the divergence measure. Both require the 'dials' package.

Usage

```
renyi_order(range = c(0.05, 0.95), trans = NULL)

divergence_type(values = c("renyi", "kl", "jeffreys", "bhattacharyya"))

renyi_kind(range = c(1L, 4L))
```

Arguments

range	Range of the Rényi order.
trans	A transformation object (or NULL).
values	Candidate divergences.

Value

A 'dials' parameter object.

Examples

```
if (requireNamespace("dials", quietly = TRUE)) {
  renyi_order()
  divergence_type()
}
```

`SanFrancisco150` *San Francisco PolSAR image (150 x 150 pixels)*

Description

A 150×150 crop of the fully polarimetric L-band AIRSAR image of the San Francisco Bay area (nominal 4 looks), used in the dissertation and thesis of J. A. Ferreira. The scene contains ocean, vegetation (forest/park) and urban areas; see SanFrancisco150_gt.

Usage

```
SanFrancisco150
```

Format

A complex array of dimension `150 x 150 x 6`. For each pixel the six layers are the distinct entries of the 3×3 Hermitian covariance matrix: $|S_{HH}|^2$, $|S_{HV}|^2$, $|S_{VV}|^2$ (real intensities), $S_{HH}S_{HV}^*$, $S_{HH}S_{VV}^*$ and $S_{HV}S_{VV}^*$ (complex products).

Source

NASA/JPL AIRSAR mission.

Examples

```
dim(SanFrancisco150)
img <- SARImage$new(SanFrancisco150, looks = 4)
img$plot()
```

`SanFrancisco150_gt` *Ground truth of the San Francisco image*

Description

Reference map of the SanFrancisco150 image with three classes.

Usage

```
SanFrancisco150_gt
```

Format

An integer matrix `150 x 150` with values `1` (ocean), `2` (vegetation) and `3` (urban area).

Examples

```
table(SanFrancisco150_gt)
plot_map(SanFrancisco150_gt)
```

`SARImage` *SAR and PolSAR images (R6)*

Description

`SARImage` stores a SAR image as an array of dimension `nrow x ncol x layers`. Two types are supported:

`"polsar"` a complex array whose `p = m(m+1)/2` layers are the distinct entries of the $m \times m$ covariance matrix of each pixel, in the order $C_{11}, \ldots, C_{mm}, C_{12}, C_{13}, \ldots$ (for $m = 3$: `HH, HV, VV, HH.HV*, HH.VV*, HV.VV` as in SanFrancisco150;

`"intensity"` a real (or real-valued complex) array whose layers are intensity channels.

The class offers cropping, Pauli colour composition, plotting, local (moving-window) features for classification, entropy maps and conversion to tibbles.

Value

An R6::R6Class generator. `SARImage$new()` returns a new `SARImage` object; its methods are described in the Methods section below.

Public fields

`data` The image array.

`type` `"polsar"` or `"intensity"`.

`channels` Names of the intensity channels.

`looks` Nominal number of looks (optional, `NULL` if unknown).

Active bindings

`nrow` Number of rows (read-only).

`ncol` Number of columns (read-only).

`m` Number of channels; matrix order for PolSAR images (read-only).

Methods

Public methods:

- SARImage$new()
- SARImage$intensity()
- SARImage$intensities()
- SARImage$crop()
- SARImage$pixel()
- SARImage$window()
- SARImage$pauli()
- SARImage$plot()
- SARImage$features()
- SARImage$entropy_map()
- SARImage$as_tibble()
- SARImage$print()
- SARImage$clone()

`SARImage$new()`: Create a SAR image.

Usage:

```
SARImage$new(data, type = NULL, looks = NULL, channels = NULL)
```

Arguments:

`data` Array `nrow x ncol x layers` (a matrix is treated as a single intensity channel).

`type` `"polsar"` or `"intensity"`. Guessed when `NULL`: complex arrays with `m(m+1)/2` layers (`m >= 2`) and non-zero imaginary parts are PolSAR images.

`looks` Nominal number of looks.

`channels` Names of the intensity channels.

`SARImage$intensity()`: Intensity of one channel.

Usage:

```
SARImage$intensity(channel = 1)
```

Arguments:

`channel` Channel index or name.

`SARImage$intensities()`: All intensity channels as a real array.

Usage:

```
SARImage$intensities()
```

`SARImage$crop()`: Crop the image.

Usage:

```
SARImage$crop(rows, cols)
```

Arguments:

`rows, cols` Row and column indices to keep.

`SARImage$pixel()`: Observation at a pixel: the Hermitian covariance matrix (PolSAR) or the vector of intensities.

Usage:

```
SARImage$pixel(row, col)
```

Arguments:

`row, col` Pixel position.

`SARImage$window()`: Observations inside a square window centred at a pixel (truncated at the borders). Returns an `n x layers` matrix.

Usage:

```
SARImage$window(row, col, window = 3)
```

Arguments:

`row, col` Pixel position.

`window` Odd window side.

`SARImage$pauli()`: Pauli RGB composition scaled to [0, 1] (PolSAR images with three channels); for other images the first three intensities are used.

Usage:

```
SARImage$pauli(stretch = 0.98, gamma = 0.5)
```

Arguments:

`stretch` Upper quantile used for linear stretching.

`gamma` Gamma correction exponent.

`SARImage$plot()`: Plot the image (Pauli composition or one channel in dB).

Usage:

```
SARImage$plot(type = c("pauli", "intensity"), channel = 1, main = NULL, ...)
```

Arguments:

`type` `"pauli"` or `"intensity"`.

`channel` Channel for `type = "intensity"`.

`main` Title.

`...` Passed to `graphics::rasterImage()`.

`SARImage$features()`: Local features for classification; see `sar_features()`.

Usage:

```
SARImage$features(window = 3, type = c("all", "gaussian", "wishart"))
```

Arguments:

`window` Odd window side.

`type` Which features: `"all"`, `"gaussian"` or `"wishart"`.

`SARImage$entropy_map()`: Entropy map; see `entropy_map()`.

Usage:

```
SARImage$entropy_map(...)
```

Arguments:

`...` Passed to `entropy_map()`.

`SARImage$as_tibble()`: Convert to a tibble with one row per pixel.

Usage:

```
SARImage$as_tibble()
```

`SARImage$print()`: Print a summary.

Usage:

```
SARImage$print(...)
```

Arguments:

`...` Ignored.

`SARImage$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SARImage$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
img <- SARImage$new(SanFrancisco150, looks = 4)
img
small <- img$crop(1:40, 1:40)
dim(small$intensities())
small$pixel(1, 1)
head(small$features(window = 3))

img$plot()
```

`SARModel` *Base R6 class for SAR and PolSAR statistical models*

Description

`SARModel` is the abstract parent of all model classes of `SITSAR` (GammaSAR, G0I, GGammaSAR, WishartSAR, G0Polarimetric and CompoundWishart). Every model can evaluate densities, simulate data, be fitted by maximum likelihood (`$fit()`) and report its Shannon entropy and the asymptotic variance of the entropy estimator.

Fitted models work with the `generics` verbs tidy(), glance(), as well as with `stats::logLik()` and `stats::AIC()`.

Value

An R6::R6Class generator. `SARModel$new()` returns a new `SARModel` object; its methods are described in the Methods section below.

Public fields

`name` Model name.

`params` Named list of parameters.

`fixed` Names of parameters held fixed during estimation.

`n` Sample size used in the last fit (`NULL` if not fitted).

`loglik` Maximised log-likelihood (`NULL` if not fitted).

`converged` Logical convergence flag of the last fit.

Methods

Public methods:

- `SARModel$new()`
- `SARModel$density()`
- `SARModel$random()`
- `SARModel$fit()`
- `SARModel$log_likelihood()`
- `SARModel$entropy()`
- `SARModel$entropy_variance()`
- `SARModel$n_parameters()`
- `SARModel$tidy()`
- `SARModel$glance()`
- `SARModel$print()`
- `SARModel$clone()`

`SARModel$new()`: Abstract constructor.

Usage:

```
SARModel$new(...)
```

Arguments:

`...` Ignored.

`SARModel$density()`: (Log-)density of observations.

Usage:

```
SARModel$density(x, log = FALSE)
```

Arguments:

`x` Observations.

`log` Return log-densities?

`SARModel$random()`: Generate random observations.

Usage:

```
SARModel$random(n)
```

Arguments:

`n` Number of observations.

`SARModel$fit()`: Fit the model by maximum likelihood.

Usage:

```
SARModel$fit(x, ...)
```

Arguments:

`x` Observations.

`...` Model specific arguments.

`SARModel$log_likelihood()`: Log-likelihood of a sample.

Usage:

```
SARModel$log_likelihood(x)
```

Arguments:

`x` Observations.

`SARModel$entropy()`: Shannon entropy at the current parameters.

Usage:

```
SARModel$entropy()
```

`SARModel$entropy_variance()`: Asymptotic variance of the entropy estimator (delta method with the inverse observed Fisher information).

Usage:

```
SARModel$entropy_variance(n = self$n)
```

Arguments:

`n` Sample size. Defaults to the size of the fitted sample.

`SARModel$n_parameters()`: Number of free parameters.

Usage:

```
SARModel$n_parameters()
```

`SARModel$tidy()`: Parameters as a tibble.

Usage:

```
SARModel$tidy()
```

`SARModel$glance()`: One-row summary of the fit.

Usage:

```
SARModel$glance()
```

`SARModel$print()`: Print the model.

Usage:

`SARModel$print(...)`

Arguments:

`...` Ignored.

`SARModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SARModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`sarmodel-methods` *Tidy and glance methods for SITSAR models*

Description

Tidy and glance methods for SITSAR models

Usage

```
## S3 method for class 'SARModel'
tidy(x, ...)

## S3 method for class 'SARModel'
glance(x, ...)

## S3 method for class 'SARModel'
logLik(object, ...)
```

Arguments

`x`	A fitted SARModel object.
`...`	Not used.
`object`	A fitted SARModel object.

Value

`tidy()` returns a tibble with one row per scalar parameter; `glance()` a one-row tibble; `logLik()` an object of class `logLik`.

Examples

```
set.seed(1)
m <- G0I$new()$fit(rg0i(300, -2, 1, 4), L = 4)
generics::tidy(m)
generics::glance(m)
logLik(m)
```

`sar_features` *Local features of SAR images for pixel classification*

Description

Computes moving-window statistics around every pixel, in the format used by DivergenceClassifier and the `divergence_classifier()` 'parsnip' model. Window statistics are computed with integral images, so the cost does not depend on the window size. Windows are truncated at the image borders.

Usage

```
sar_features(image, window = 3, type = c("all", "gaussian", "wishart"))
```

Arguments

`image`	A SARImage or an array accepted by `SARImage$new()`.
`window`	Odd window side.
`type`	Which features: `"all"`, `"gaussian"` or `"wishart"`.

Details

Two groups of columns are produced:

Gaussian features `mean_<ch>`: local mean of each intensity channel; `cov_<ch1>_<ch2>`: local (unbiased) covariance between the intensities. Together they define the multivariate normal fitted to the intensity vector in the window.

Wishart features (PolSAR only) `c<i><i>` and `c<i><j>_re`/`c<i><j>_im`: entries of the local mean covariance matrix, i.e. the maximum likelihood estimate of Σ for the scaled complex Wishart law in the window.

Value

A tibble with the columns `row`, `col` and the features.

Examples

```
img <- SARImage$new(SanFrancisco150)$crop(1:20, 1:20)
f <- sar_features(img, window = 3)
names(f)
```

segment_sar *Entropy-based segmentation of a SAR/PolSAR image*

Description

Convenience wrapper that computes the entropy map(s) with `entropy_map()` and segments them with EntropySegmenter.

Usage

```
segment_sar(
  image,
  k = 2,
  model = c("wishart", "g0p", "gamma", "g0i", "ggamma"),
  window = 7,
  L = NULL,
  channels = NULL,
  method = c("em", "sem"),
  ...
)
```

Arguments

`image`	A SARImage or an array accepted by `SARImage$new()`.
`k`	Number of classes.
`model`	One of `"gamma"`, `"g0i"`, `"ggamma"` (intensity models, applied to each channel) or `"wishart"`, `"g0p"` (matrix models, for PolSAR images).
`window`	Odd side of the moving window.
`L`	Known number of looks. If `NULL` (default) it is estimated in every window.
`channels`	Channels used by intensity models (default: all).
`method`	`"em"` or `"sem"`.
`...`	Passed to `EntropySegmenter$new()`.

Value

A fitted EntropySegmenter; the labels are available with `$label_map()`.

Examples

```
set.seed(1)
seg <- segment_sar(SanFrancisco150, k = 3, model = "gamma", window = 7, L = 4)
table(seg$label_map(), SanFrancisco150_gt)
```

WishartSAR *Scaled complex Wishart model for PolSAR data (R6)*

Description

R6 class for the scaled complex Wishart law $\mathcal{W}_m(\Sigma, L)$ (see `dwishart_sar()`). The maximum likelihood estimator of Σ is the sample mean and L solves $m \log L - \psi_m^{(0)}(L) = \log|\bar{Z}| - \overline{\log|Z|}$ (Anfinsen et al., 2009). The entropy variance is available in closed form (Nascimento, Ferreira and Frery, 2023).

Value

An R6::R6Class generator. `WishartSAR$new()` returns a new `WishartSAR` object; its methods are described in the Methods section below.

Super class

`SARModel` -> `WishartSAR`

Methods

Public methods:

- `WishartSAR$new()`
- `WishartSAR$density()`
- `WishartSAR$random()`
- `WishartSAR$fit()`
- `WishartSAR$entropy()`
- `WishartSAR$entropy_variance()`
- `WishartSAR$clone()`

`WishartSAR$new()`: Create a Wishart model.

Usage:

```
WishartSAR$new(Sigma = diag(3) + (0+0i), L = 4)
```

Arguments:

`Sigma` Hermitian positive definite matrix (default identity 3x3).

`L` Number of looks.

`WishartSAR$density()`: (Log-)density.

Usage:

```
WishartSAR$density(x, log = FALSE)
```

Arguments:

`x` PolSAR observations (array `m x m x n` or vec matrix).

`log` Return log-densities?

`WishartSAR$random()`: Simulate PolSAR observations.

Usage:

```
WishartSAR$random(n, output = "array")
```

Arguments:

`n` Number of observations.

`output` `"array"` or `"vec"`.

`WishartSAR$fit()`: Fit by maximum likelihood.

Usage:

```
WishartSAR$fit(x, L = NULL)
```

Arguments:

`x` PolSAR observations (array `m x m x n` or vec matrix).

`L` Known number of looks. If `NULL` (default) it is estimated.

`WishartSAR$entropy()`: Shannon entropy.

Usage:

```
WishartSAR$entropy()
```

`WishartSAR$entropy_variance()`: Asymptotic variance of the entropy estimator (closed form).

Usage:

```
WishartSAR$entropy_variance(n = self$n)
```

Arguments:

`n` Sample size.

`WishartSAR$clone()`: The objects of this class are cloneable with this method.

Usage:

```
WishartSAR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Anfinsen, S. N., Doulgeris, A. P. and Eltoft, T. (2009). Estimation of the equivalent number of looks in polarimetric synthetic aperture radar imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 47(11), 3795–3809.

Examples

```
set.seed(1)
Z <- rwishart_sar(300, polsar_example_sigma("forest"), L = 4)
m <- WishartSAR$new()$fit(Z)
m
m$entropy_variance()
```

Index

∗ **datasets**
SanFrancisco150, 40
SanFrancisco150_gt, 40

as_polsar_array *(*polsar-format*)*, 38
as_polsar_vec *(*polsar-format*)*, 38
as_polsar_vec(), *8*, *10*, *21*

change_detection_metrics, 3
change_map, 4
compound_wishart, 7
CompoundWishart, 5, *8*, *44*

dcgcw *(*compound_wishart*)*, 7
dctpcw *(*compound_wishart*)*, 7
dctpcw(), *5*
dg0i, 9
dg0i(), *28*
dg0p, 10
dg0p(), *31*
dgammasar, 11
dgammasar(), *32*
dggamma, 12
dggamma(), *34*
distancesg0i, 13
distancesg0i(), *14*
distancesg0itable, 14
divergence_classifier, 19
divergence_classifier(), *15*, *39*, *48*
divergence_gaussian *(*divergences*)*, 17
divergence_gaussian(), *15*
divergence_type *(*renyi_order*)*, 39
divergence_wishart *(*divergences*)*, 17
DivergenceClassifier, 15, *20*, *48*
divergences, 17, *20*
dwishart_sar, 21
dwishart_sar(), *50*

entropy, 22
entropy_g0i *(*entropy*)*, 22
entropy_g0p *(*entropy*)*, 22
entropy_gamma *(*entropy*)*, 22
entropy_ggamma *(*entropy*)*, 22
entropy_map, 26
entropy_map(), *43*, *44*, *49*
entropy_test, 27
entropy_test(), *4*
entropy_wishart *(*entropy*)*, 22
EntropySegmenter, 23, *49*

G0I, *10*, 28, *30*, *44*
g0iestimation, 30
G0Polarimetric, *11*, 31, *44*
GammaSAR, *12*, 32, *44*
GGammaSAR, *13*, 34, *44*
glance(), *44*
glance.SARModel *(*sarmodel-methods*)*, 47
graphics::rasterImage(), *43*

kappa_statistics, 36

logLik.SARModel *(*sarmodel-methods*)*, 47

pg0i *(*dg0i*)*, 9
pgammasar *(*dgammasar*)*, 11
pggamma *(*dggamma*)*, 12
plot_map, 37
polsar-format, 38
polsar_example_sigma, 38

qg0i *(*dg0i*)*, 9
qgammasar *(*dgammasar*)*, 11
qggamma *(*dggamma*)*, 12

R6::R6Class, *5*, *15*, *24*, *28*, *31*, *32*, *34*, *41*, *45*, *50*
rcgcw *(*compound_wishart*)*, 7
rctpcw *(*compound_wishart*)*, 7
renyi_kind *(*renyi_order*)*, 39
renyi_order, 39
renyi_order(), *20*

rg0i *(dg0i)*, 9
rg0p *(dg0p)*, 10
rgammasar *(dgammasar)*, 11
rggamma *(dggamma)*, 12
rwishart_sar *(dwishart_sar)*, 21

SanFrancisco150, *38*, *40*, 40, *41*
SanFrancisco150_gt, *40*, 40
sar_features, 48
sar_features(), *15, 16*, *19, 20*, *43*
SARImage, *4*, *14*, *26*, 41, *48, 49*
SARModel, *5*, *28*, *31*, *33, 34*, 44, *47*, *50*
sarmodel-methods, 47
segment_sar, 49
stats::AIC(), *44*
stats::logLik(), *44*

tidy(), *44*
tidy.SARModel *(sarmodel-methods)*, 47

update.divergence_classifier
(divergence_classifier), 19

WishartSAR, *22*, *44*, 50

yardstick::kap(), *36*