# Package 'SpatialVx'

July 4, 2018

**Version** 0.6-3

**Date** 2018-07-03

**Title** Spatial Forecast Verification

**Author** Eric Gilleland <EricG@ucar.edu>

**Maintainer** Eric Gilleland <EricG@ucar.edu>

**Depends** R (>= 2.10.0), spatstat (>= 1.37-0), fields (>= 6.8), smoothie, smatr, turboEM

**Imports** distillery, maps, boot, CircStats, fastcluster, waveslim

**Description** Spatial forecast verification arose from verifying high-resolution forecasts, where coarser-resolution models generally are favored even when a human forecaster finds the higher-resolution model to be considerably better. Most newly proposed methods, which largely come from image analysis, computer vision, and similar, are available, with more on the way.

**License** GPL (>= 2)

**URL** http://www.ral.ucar.edu/projects/icp/SpatialVx

**BugReports** http://www.ral.ucar.edu/staff/ericg

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2018-07-04 17:50:02 UTC

## R topics documented:

---

`SpatialVx-package` *Spatial Forecast Verification*

---

## Description

**SpatialVx** contains functions to perform many spatial forecast verification methods.

## Details

Primary functions include:

0. `make.SpatialVx`: An object that contains the verification sets and pertinent information.

1. Filter Methods:

1a. Neighborhood Methods:

Neighborhood methods generally apply a convolution kernel smoother to one or both of the fields in the verificaiton set, and then apply the traditional scores. Most of the methods reviewed in Ebert (2008, 2009) are included in this package. The main functions are:

`hoods2d`, `pphindcast2d`, `kernel2dsmooth`, and `plot.hoods2d`.

1b. Scale Separation Methods:

Scale separation refers to the idea of applying a band-pass filter (and/or doing a multi-resolution analysis, MRA) to the verification set. Typically, skill is assessed on a scale-by-scale basis. However, other techniques are also applied. For example, denoising the field before applying traditional statistics, using the variogram, or applying a statistical test based on the variogram (these last are less similar to the spirit of the scale separation idea, but are at least somewhat related).

There is functionality to do the wavelet methods proposed in Briggs and Levine (1997). In particular, to simply denoise the fields before applying traditional verification statistics, use

`wavePurifyVx`.

To apply verification statistics to detail fields (in either the wavelet or field space), use:

waverify2d (dyadic fields) or mowaverify2d (non-dyadic or dyadic) fields.

The intensity-scale technique introduced in Casati et al. (2004) and the new developments of the technique proposed in Casati (2010) can be performed with

waveIS.

Although not strictly a "scale separation" method, the structure function (for which the variogram is a special case) is in the same spirit in the sense that it analyzes the field for different separation distances, and these "scales" are separate from each other (i.e., the score does not necessarily improve or decline as the scale increases). This package contains essentially wrapper functions to the vgram.matrix and plot.vgram.matrix functions from the fields package, but there is also a function called

variogram.matrix

that is a modification of vgram.matrix that allows for missing values. The primary function for doing this is called

griddedVgram, which has a plot method function associated with it.

There are also slight modifications of these functions (small modifications of the fields functions) to calculate the structure function of Harris et al. (2001). These functions are called

structurogram (for non-gridded fields) and structurogram.matrix (for gridded fields).

The latter allows for ignoring zero-valued grid points (as detailed in Harris et al., 2001) where the former does not (they must be removed prior to calling the function).

2. Displacement Methods:

In Gilleland et al (2009), this category was broken into two main types as field deformation and features-based. The former lumped together binary image measures/metrics with field deformation techniques because the binary image measures inform about the "similarity" (or dissimilarity) between the spatial extent or pattern of two fields (across the entire field). Here, they are broken down further into those that yield only a single (or small vector of) metric(s) or measure(s) (location measures), and those that have mechanisms for moving grid-point locations to match the fields better spatially (field deformation).

2a. Location Measures:

Included here are the well-known Hausdorff metric, the partial-Hausdorff measure, FQI (Venugopal et al., 2005), Baddeley's delta metric (Baddeley, 1992; Gilleland, 2011; Schwedler et al., 2011), metrV (Zhu et al., 2011), as well as the localization performance measures described in Baddeley, 1992: mean error distance, mean square error distance, and Pratt's Figure of Merit (FOM).

locmeasures2d, metrV, distob, locperf

Image moments can give useful information about location errors, and are used within feature-based methods, particularly MODE, as they give the centroid of an image (or feature), as well as the orientation angle, among other useful properties. See the imomenter function for more details.

2b. Field deformation:

Thanks to Caren Marzban for supplying his optical flow code for this package (it has been modified some). These functions perform the analyses described in Marzban and Sandgathe (2010) and are based on the work of Lucas and Kanade (1981). See the help file for

OF.

Rigid transformations can be estimated using the rigider function. To simply rigidly transform a field (or feature) using specified parameters (x- and y- translations and/or rotations), the

rigidTransform function can be used. For these functions, which may result in transformations that do not perfectly fall onto grid points, the function Fint2d can be used to interpolate from nearest grid points. Interpolation options include "round" (simply take the nearest location value), "bilinear" and "bicubic".

2c. Features-based methods: These methods are also sometimes called object-based methods (the term "features" is used in this package in order to differentiate from an R object), and have many similarities to techniques used in Object-Based Image Analysis (OBIA), a relatively new research area that has emerged primarily as a result of advances in earth observations sensors and GIScience (Blaschke et al., 2008). It is attempted to identify individual features within a field, and subsequently analyze the fields on a feature-by-feature basis. This may involve intensity error information in addition to location-specific error information. Additionally, contingency table verifcation statistics can be found using new definitions for hits, misses and false alarms (correct negatives are more difficult to asses, but can also be done).

Currently, there is functionality for performing the analyses introduced in Davis et al. (2006,2009), including the merge/match algorithm of Gilleland et al (2008), as well as the SAL technique of Wernli et al (2008, 2009). Some functionality for composite analysis (Nachamkin, 2004) is provided by way of placing individual features onto a relative grid so that each shares the same centroid. Shape analysis is partially supported by way of functions to identify boundary points (Micheas et al. 2007; Lack et al. 2010). In particular, see:

Functions to identify features: FeatureFinder

Functions to match/merge features: centmatch, deltamm, minboundmatch

Functions to diagnose features and/or compare matched features:

FeatureAxis, FeatureComps, FeatureMatchAnalyzer, FeatureProps, FeatureTable, interester

See compositer for setting up composited objects, and see hiw (along with distill and summary method functions) for some shape analysis functionality.

The cluster analysis methods of Marzban and Sandgathe (2006; 2008) have been added. The former method was written from scratch by Eric Gilleland

clusterer

and the latter variation was modified from code originally written by Hilary Lyons

CSIsamples.

The contiguous rain area (CRA) utilizes the rigider function, and can be carried out with the function craer.

The Structure, Amplitude and Location (SAL) method can be performed with saller.

2d. Geometrical characterization measures:

Perhaps the measures in this sub-heading are best described as part-and-parcel of 2c. They are certainly useful in that domain, but have been proposed also for entire fields by AghaKouchak et al. (2011); though similar measures have been applied in, e.g., MODE. The measures introduced in AghaKouchak et al. (2011) available here are: connectivity index (Cindex), shape index (Sindex), and area index (Aindex):

Cindex, Sindex, Aindex

3. Statistical inferences for spatial (and/or spatiotemporal) fields:

In addition to the methods categorized in Gilleland et al. (2009), there are also functions for making comparisons between two spatial fields. The field significance approach detailed in Elmore et

al. (2006), which requires a temporal dimension as well, involves using a circular block bootstrap (CBB) algorithm (usually for the mean error) at each grid point (or location) individually to determine grid-point significance (null hypothesis that the mean error is zero), and then a semi-parametric Monte Carlo method viz. Livezey and Chen (1983) to determine field significance.

`spatbiasFS`, `LocSig`, `MCdof`

In addition, the spatial prediction comparison test (SPCT) introduced by Hering and Genton (2011) is included via the functions: `lossdiff`, `empiricalVG` and `flossdiff`. Supporting functions for calculating the loss functions include: absolute error (`abserrloss`), square error (`sqerrloss`) and correlation skill (`corrskill`), as well as the distance map loss function (`distmaploss`) introduced in Gilleland (2013).

4. Other:

The bias corrected TS and ETS (or TS dHdA and ETS dHdA) introduced in Mesinger (2008) are now included within the `vxstats` function.

The 2-d Gaussian Mixture Model (GMM) approach introduced in Lakshmanan and Kain (2010) can be carried out using the

`gmm2d`

function (to estimate the GMM) and the associated `summary` function calculates the parameter comparisons. Also available are `plot` and `predict` method functions, but it can be very slow to run. The `gmm2d` employs an initialization function that takes the K largest object areas (connected components) and uses their centroids as initial estimates for the means, and uses the axes as initial guesses for the standard deviations. However, the user may supply their own initial estimate function.

The S1 score and anomaly correlation (ACC) are available through the functions

`S1` and `ACC`.

See Brown et al. (2012) and Thompson and Carter (1972) for more on these statistics.

Also included is a function to do the geographic box-plot of Willmott et al. (2007). Namely,

`GeoBoxPlot`.

Datasets:

All of the initial Spatial Forecast Verification Inter-Comparison Project (ICP, `http://www.ral.ucar.edu/projects/icp`) data sets used in the special collection of the Weather and Forecasting journal are included. See the help file for

`obs0426`,

which gives information on all of these datasets that are included, as well as two examples for plotting them: one that does not preserve projections, but plots the data without modification, and another that preserves the projections, but possibly with some interpolative smoothing.

Ebert (2008) provides a nice review of these methods. Roberts and Lean (2008) describes one of the methods, as well as the primary boxcar kernel smoothing method used throughout this package. Gilleland et al. (2009, 2010) provides an overview of most of the various recently proposed methods, and Ahijevych et al. (2009) describes the data sets included in this package. Some of these have been applied to the ICP test cases in Ebert (2009).

Additionally, one of the NIMROD cases (as provided by the UK Met Office) analyzed in Casati et al (2004) (case 6) is included along with approximate lon/lat locations. See the help file for UKobs6 more information.

A spatio-temporal verification dataset is also included for testing the method of Elmore et al. (2006). See the help file for

`GFSNAMfcstEx`.

A simulated dataset similar to the one used in Marzban and Sandgathe (2010) is also available and is called

`hump`.

## Author(s)

Eric Gilleland

## References

AghaKouchak, A., Nasrollahi, N. Li, J. Imam, B. and Sorooshian, S. (2011) Geometrical characterization of precipitation patterns. *J. Hydrometeorology*, **12**, 274–285, doi:10.1175/2010JHM1298.1.

Ahijevych, D., Gilleland, E., Brown, B. G. and Ebert, E. E. (2009) Application of spatial verification methods to idealized and NWP gridded precipitation forecasts. *Wea. Forecasting*, **24** (6), 1485–1497.

Baddeley, A. J. (1992) An error metric for binary images. In *Robust Computer Vision Algorithms*, Forstner, W. and Ruwiedel, S. Eds., Wichmann, 59–78.

Blaschke, T., Lang, S. and Hay, G. (Eds.) (2008) Object-Based Image Analysis. Berlin, Germany: Springer-Verlag, 818 pp.

Briggs, W. M. and Levine, R. A. (1997) Wavelets and field forecast verification. *Mon. Wea. Rev.*, **125**, 1329–1341.

Brown, B. G., Gilleland, E. and Ebert, E. E. (2012) Chapter 6: Forecasts of spatial fields. pp. 95–117, In *Forecast Verification: A Practitioner's Guide in Atmospheric Science*, 2nd edition. Edts. Jolliffe, I. T. and Stephenson, D. B., Chichester, West Sussex, U.K.: Wiley, 274 pp.

Casati, B. (2010) New Developments of the Intensity-Scale Technique within the Spatial Verification Methods Inter-Comparison Project. *Wea. Forecasting* **25**, (1), 113–143, doi:10.1175/2009WAF2222257.1.

Casati, B., Ross, G. and Stephenson, D. B. (2004) A new intensity-scale approach for the verification of spatial precipitation forecasts. *Meteorol. Appl.* **11**, 141–154.

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006) Object-based verification of precipitation forecasts, Part I: Methodology and application to mesoscale rain areas. *Mon. Wea. Rev.*, **134**, 1772–1784.

Ebert, E. E. (2008) Fuzzy verification of high resolution gridded forecasts: A review and proposed framework. *Meteorol. Appl.*, **15**, 51–64. DOI: 10.1002/met.25

Ebert, E. E. (2009) Neighborhood verification: A strategy for rewarding close forecasts. *Wea. Forecasting*, **24**, 1498–1510, doi:10.1175/2009WAF2222251.1.

Elmore, K. L., Baldwin, M. E. and Schultz, D. M. (2006) Field significance revisited: Spatial bias errors in forecasts as applied to the Eta model. *Mon. Wea. Rev.*, **134**, 519–531.

Gilleland, E. (2013) Testing competing precipitation forecasts accurately and efficiently: The spatial prediction comparison test. *Mon. Wea. Rev.*, **141**, (1), 340–355.

Gilleland, E. (2011) Spatial forecast verification: Baddeley's delta metric applied to the ICP test cases. *Wea. Forecasting*, **26**, 409–415, doi:10.1175/WAF-D-10-05061.1.

Gilleland, E., Lee, T. C. M., Halley Gotway, J., Bullock, R. G. and Brown, B. G. (2008) Computationally efficient spatial forecast verification using Baddeley's delta image metric. *Mon. Wea. Rev.*, **136**, 1747–1757.

Gilleland, E., Ahijevych, D., Brown, B. G., Casati, B. and Ebert, E. E. (2009) Intercomparison of Spatial Forecast Verification Methods. *Wea. Forecasting*, **24**, 1416–1430, doi:10.1175/2009WAF2222269.1.

Gilleland, E., Ahijevych, D. A., Brown, B. G. and Ebert, E. E. (2010) Verifying Forecasts Spatially. *Bull. Amer. Meteor. Soc.*, October, 1365–1373.

Harris, D., Foufoula-Georgiou, E., Droegemeier, K. K. and Levit, J. J. (2001) Multiscale statistical properties of a high-resolution precipitation forecast. *J. Hydrometeorol.*, **2**, 406–418.

Hering, A. S. and Genton, M. G. (2011) Comparing spatial predictions. *Technometrics* **53**, (4), 414–425.

Lack, S., Limpert, G. L. and Fox, N. I. (2010) An object-oriented multiscale verification scheme. *Wea. Forecasting*, **25**, 79–92, DOI: 10.1175/2009WAF2222245.1

Lakshmanan, V. and Kain, J. S. (2010) A Gaussian Mixture Model Approach to Forecast Verification. *Wea. Forecasting*, **25** (3), 908–920.

Livezey, R. E. and Chen, W. Y. (1983) Statistical field significance and its determination by Monte Carlo techniques. *Mon. Wea. Rev.*, **111**, 46–59.

Lucas, B D. and Kanade, T. (1981) An iterative image registration technique with an application to stereo vision. *Proc. Imaging Understanding Workshop*, DARPA, 121–130.

Marzban, C. and Sandgathe, S. (2006) Cluster analysis for verification of precipitation fields. *Wea. Forecasting*, **21**, 824–838.

Marzban, C. and Sandgathe, S. (2008) Cluster Analysis for Object-Oriented Verification of Fields: A Variation. *Mon. Wea. Rev.*, **136**, (3), 1013–1025.

Marzban, C. and Sandgathe, S. (2009) Verification with variograms. *Wea. Forecasting*, **24** (4), 1102–1120, doi: 10.1175/2009WAF2222122.1

Marzban, C. and Sandgathe, S. (2010) Optical flow for verification. *Wea. Forecasting*, **25**, 1479–1494, doi:10.1175/2010WAF2222351.1.

Mesinger, F. (2008) Bias adjusted precipitation threat scores. *Adv. Geosci.*, **16**, 137–142.

Micheas, A. C., Fox, N. I., Lack, S. A. and Wikle, C. K. (2007) Cell identification and verification of QPF ensembles using shape analysis techniques. *J. of Hydrology*, **343**, 105–116.

Nachamkin, J. E. (2004) Mesoscale verification using meteorological composites. *Mon. Wea. Rev.*, **132**, 941–955.

Roberts, N. M. and Lean, H. W. (2008) Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Mon. Wea. Rev.*, **136**, 78–97. doi:10.1175/2007MWR2123.1.

Schwedler, B. R. J. and Baldwin, M. E. (2011) Diagnosing the sensitivity of binary image measures to bias, location, and event frequency within a forecast verification framework. *Wea. Forecasting*, **26**, 1032–1044, doi:10.1175/WAF-D-11-00032.1.

Thompson, J. C. and Carter, G. M. (1972) On some characteristics of the S1 score. *J. Appl. Meteorol.*, **11**, 1384–1385.

Venugopal, V., Basu, S. and Foufoula-Georgiou, E. (2005) A new metric for comparing precipitation patterns with an application to ensemble forecasts. *J. Geophys. Res.*, **110**, D08111, doi:10.1029/2004JD005395, 11pp.

Wernli, H., Paulat, M., Hagen, M. and Frei, C. (2008) SAL–A novel quality measure for the verification of quantitative precipitation forecasts. *Mon. Wea. Rev.*, **136**, 4470–4487.

Wernli, H., Hofmann, C. and Zimmer, M. (2009) Spatial forecast verification methods intercomparison project: Application of the SAL technique. *Wea. Forecasting*, **24**, 1472–1484, doi:10.1175/2009WAF2222271.1

Willmott, C. J., Robeson, S. M. and Matsuura, K. (2007) Geographic box plots. *Physical Geography*, **28**, 331–344, DOI: 10.2747/0272-3646.28.4.331.

Zhu, M., Lakshmanan, V. Zhang, P. Hong, Y. Cheng, K. and Chen, S. (2011) Spatial verification using a true metric. *Atmos. Res.*, **102**, 408–419, doi:10.1016/j.atmosres.2011.09.004.

## Examples

```
## See help files for above named functions and datasets
## for specific examples.
```

---

`abserrloss` *Loss functions for the spatial prediction comparison test (SPCT)*

---

## Description

Loss functions for applying the spatial prediction comparison test (SPCT) for competing forecasts.

## Usage

```
abserrloss(x, y, ...)
corrskill(x, y, ...)
sqerrloss(x, y, ...)
distmaploss(x, y, threshold = 0, const = Inf, ...)
```

## Arguments

| | |
|---|---|
| `x,y` | m by n numeric matrices against which to calculate the loss (or skill) functions. |
| `threshold` | numeric giving the threshold over which (and including) binary fields are created from `x` and `y` in order to make a distance map. |
| `const` | numeric giving the constant beyond which the differences in distance maps between `x` and `y` are set to zero. If `Inf` (default), then no cut-off is taken. The SPCT is probably not powerful for large values of `const`. |
| `...` | Not used by `abserrloss` or `sqerrloss` (there for consistency only, and in order to work with `lossdiff`). For `corrskill`, these are optional arguments to `sd`. For `distmaploss`, these are optional arguments to the `distmap` function from pacakge **spatstat**. |

## Details

These are simple loss functions that can be used in conjunction with `lossdiff` to carry out the spatial prediction comparison test (SPCT) as introduced in Hering and Genton (2011); see also Gilleland (2013) in particular for details about the distance map loss function.

The distance map loss function does not zero-out well as the other loss functions do. Therefore, `zero.out` should be `FALSE` in the call to `lossdiff`. Further, as pointed out in Gilleland (2013), the distance map loss function can easily be hedged by having a lot of correct negatives. The image warp loss function is probably better for this purpose if, e.g., there are numerous zero-valued grid points in all fields.

## Value

numeric m by n matrices containing the value of the loss (or skill) function at each location i of the original set of locations (or grid of points).

## Author(s)

Eric Gilleland

## References

Gilleland, E. (2013) Testing competing precipitation forecasts accurately and efficiently: The spatial prediction comparison test. *Mon. Wea. Rev.*, **141**, (1), 340–355.

Hering, A. S. and Genton, M. G. (2011) Comparing spatial predictions. *Technometrics* **53**, (4), 414–425.

## See Also

`lossdiff`, `vgram.matrix`, `vgram`

## Examples

```
# See help file for lossdiff for examples.
```

---

`Aindex` *Area Index*

---

## Description

Calculate Area index described in AghaKouchak et al. (2011).

## Usage

```
Aindex(x, thresh = NULL, dx = 1, dy = 1, ...)

## Default S3 method:
Aindex(x, thresh = NULL, dx = 1, dy = 1, ...)

## S3 method for class 'SpatialVx'
Aindex(x, thresh = NULL, dx = 1, dy = 1, ...,
    time.point=1, obs = 1, model=1)
```

## Arguments

| | |
|---|---|
| x | Default: m by n numeric matrix giving the field for which the area index is to be calculated.<br>Aindex.SpatialVx: list object of class "SpatialVx". |
| thresh | Values under this threshold are set to zero. If NULL, it will be set to 1e-8 (a very small value). |
| dx,dy | numeric giving the grid point size in each direction if it is desired to apply such a correction. However, the values are simply canceled out in the index, so these arguments are probably not necessary. If it is desired to only get the area of the non-zero values in the field, or the convex hull, then these make sense. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| ... | Not used. |

## Details

The area index introduced in AghaKouchak et al. (2011) is given by

Aindex = A/Aconvex,

where A is the area of the pattern, and Aconvex the area of its convex hull (area.owin from package spatstat is used to calculate this latter area, and the functions as.im and solutionset from spatstat are also used by this function). Values are between 0 and 1. Values closer to unity indicate a more structured pattern, and values closer to zero indicate higher dispersiveness of the pattern, but note that two highly structured patterns far away from each other may also give a low value (see examples below). Because of this property, this measure is perhaps best applied to individual features in a field.

## Value

numeric vector (or two-row matrix in the case of Aindex.SpatialVx) with named components (columns):

| | |
|---|---|
| Aindex | numeric giving the area index. |
| A,Aconvex | numeric giving the area of th epattern and the convex hull, resp. |
| dx,dy | the values of dx and dy as input to the function. |

## Author(s)

Eric Gilleland

## References

AghaKouchak, A., Nasrollahi, N., Li, J., Imam, B. and Sorooshian, S. (2011) Geometrical characterization of precipitation patterns. *J. Hydrometeorology*, **12**, 274–285, doi:10.1175/2010JHM1298.1.

## See Also

as.im, solutionset, convexhull, Cindex, Sindex

## Examples

```
# Gemetric shape that is highly structured.
data( "geom000" )

Aindex( geom000 )
Aindex( geom000, dx = 4, dy = 4 )

## Not run:
# Two separate areas with highly structured shapes, but far away from each other.
data( "pert000" )
data( "pert006" )
data( "ICPg240Locs" )

hold <- make.SpatialVx(pert000, pert006, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert006" )

plot( hold )

Aindex( hold, thresh = 20, dx = 4, dy = 4 )


## End(Not run)
```

---

# bearing *Bearing from One Spatial Location to Another*

---

## Description

Find the bearing from one spatial location to another.

## Usage

```
bearing(point1, point2, deg = TRUE, aty = "compass")
```

## Arguments

| | |
|---|---|
| point1, point2 | two-column numeric matrices giving lon/lat coordinates for the origin point(s) (point1) and the destination point(s) (point2). |
| deg | logical, should the output be converted from radians to degrees? |
| aty | character stating either "compass" (default) or "radial". The former gives the standard compass bearing angle (0 is north, increase clockwise), and the latter is for polar coordinates (0 is East, increase counter-clockwise). |

## Details

The bearing, beta, of a point B as seen from a point A is given by

beta = atan2(S,T)

where

S = cos(phi_B) * sin(L_A - L_B), and

T = cos(phi_A)*sin(phi_B) - sin(phi_A)*cos(phi_B)*cos(L_A - L_B)

where phi_A (phi_B) is the latitude of point A (B), and L_A (L_B) is the longitude of point A (B).

Note that there is no simple relationship between the bearing of A to B vs. the bearing of B to A. The bearing given here is in the usual R convention for lon/lat information, which gives points east of Greenwich as negative longitude, and south of the equator as negative latitude.

## Value

numeric giving the bearing angle.

## Author(s)

Eric Gilleland and Randy Bullock, bullock "at" ucar.edu

## References

Keay, W. (1995) Land Navigation: Routefinding with Map & Compass, Coventry, UK: Clifford Press Ltd., ISBN 0319008452, 978-0319008454

## See Also

atan2, FeatureAxis, rdist.earth

## Examples

```
# Boulder, Colorado and Wallaroo, Australia.
A <- rbind(c(-105.2833, 40.0167), c(137.65, -33.9333))

# Wallaroo, Australia and Boulder, Colorado.
B <- rbind(c(137.65, -33.9333), c(-105.2833, 40.0167))

bearing(A,B)
```

```
bearing(A,B,aty="radial")

plot(A, type="n", xlab="", ylab="")
points(A[,1], A[,2], pch="*", col="darkblue")

# Boulder, Colorado to Wallaroo, Australia.
arrows(A[1,1], A[1,2], A[2,1], A[2,2], col="red", lwd=1.5)
```

---

`calculate_dFSS` *Calculate dFSS Score Value*

---

## Description

Calculates the value for the dFSS binary distance metric. The dFSS uses the Fraction Skill Score to provide a measure of spatial displacement of precipitation in two precipitation fields.

## Usage

```
calculate_dFSS(fbin1, fbin2)
```

## Arguments

| | |
|---|---|
| `fbin1` | A numeric matrix representing the first binary field. Only values 0 and 1 are allowed in the matrix. |
| `fbin2` | A numeric matrix representing the second binary field. Only values 0 and 1 are allowed in the matrix. The matrix needs to have the same dimensions as `fbin1`. |

## Details

The dFSS uses the Fraction Skill Score to provide a measure of spatial displacement of precipitation in two precipitation fields.

The function requires two binary fields as input. A binary field can only have values of 0 or 1 and can be obtained through a thresholding process of the original continuous precipitation field (e.g., by setting all values below a selected precipitation threshold to zero, and all values above the threshold to one).

The dFSS has a requirement that the frequency bias of precipitation needs to be small in order for the metric to work properly (i.e. the number of non-zero grid points has to be similar in both binary fields). The unbiased fields can be obtained from the original continuous precipitation fields via the use of a frequency (percentile) threshold. For example, instead of using a predefined physical threshold (e.g. 1 mm/h), which might produce binary fields with a different number of non-zero points, a frequency threshold (e.g. 5 %) can be used which guarantees that both fields will have the same number of non-zero grid-points and will thus be unbiased (provided that enough grid points in the domain contain non-zero precipitation). Function `quantile` can be used to determine the value of a physical threshold that corresponds to a prescribed frequency threshold.

If the frequency bias is larger than 1.5 the function will work but produce a warning. If the frequency bias is larger than 2 the function will produce an error. The dFSS value can only be calculated if both fields contain at least one non-zero grid point. For correct interpretation of the results and some

other considerations please look at the "Recipe" in the Conclusions section of Skok and Roberts (2018).

The code utilizes the fast method for computing fractions (Faggian et al., 2015) and the Bisection method to arrive more quickly at the correct displacement. Optionally, a significantly faster R code that requires significantly less memory and uses some embedded C++ code is available upon request from the author.

## Value

The function returns a single numeric value representing the size of the estimated spatial displacement (expressed as a number of grid points - see the example below).

## Author(s)

Gregor Skok (Gregor.Skok@fmf.uni-lj.si)

## References

Skok, G. and Roberts, N. (2018), Estimating the displacement in precipitation forecasts using the Fractions Skill Score. Q.J.R. Meteorol. Soc. doi:10.1002/qj.3212.

Faggian N., Roux B., Steinle P., Ebert B., 2015: Fast calculation of the Fractions Skill Score, MAUSAM, 66 (3), 457-466.

## See Also

hoods2d

## Examples

```
# ------------------------------------------
# A simple example with two 500 x 500 fields
# ------------------------------------------

# generate two empty 500 x 500 fields where all values are 0
fbin1=matrix(0, 500, 500, byrow = FALSE)
fbin2=fbin1

# in the fields define a single 20x20 non-zero region of precipitation
# that is horizontally displaced in the second field by 100 grid points
fbin1[200:220,200:220]=1
fbin2[200:220,300:320]=1

# calulate dFSS value
dFSS=calculate_dFSS(fbin1, fbin2)

# print dFSS value
print(dFSS)

# The example should output 97 which means that the spatial displacement
# estimated by dFSS is 97 grid points.
```

`calculate_FSSvector_from_binary_fields` *Calculate FSS Values*

## Description

Calculates the value of Fraction Skill Score (FSS) for multiple neighborhood sizes.

## Usage

```
calculate_FSSvector_from_binary_fields(fbin1, fbin2, nvector)
```

## Arguments

| | |
|---|---|
| `fbin1` | A numeric matrix representing the first binary field. Only values 0 and 1 are allowed in the matrix. |
| `fbin2` | A numeric matrix representing the second binary field. Only values 0 and 1 are allowed in the matrix. The matrix needs to have the same dimensions as `fbin1`. |
| `nvector` | A numeric vector containing neighborhood sizes for which the FSS values are to be calculated. Only positive odd values are allowed in the vector. A square neighborhood shape is assumed and the specified value represents the length of square side. |

## Details

Fractions Skill Score is a neighborhood-based spatial verification metric frequently used for verifying precipitation (see Roberts and Lean, 2008, for details).

The function requires two binary fields as input. A binary field can only have values of 0 or 1 and can be obtained through a thresholding process of the original continuous precipitation field (e.g., by setting all values below a selected precipitation threshold to zero, and all values above the threshold to one). Either a predefined physical threshold (e.g. 1 mm/h) or a frequency threshold (e.g. 5 %) can be used to produce the binary fields from the original continuous precipitation fields. If a frequency threshold is used the binary fields will be unbiased and the FSS value will asymptote to 1 at large neighborhoods. Function `quantile` can be used to determine the value of a physical threshold that corresponds to a prescribed frequency threshold.

The code utilizes the fast method for computing fractions (Faggian et al., 2015) that enables fast computation of FSS values at multiple neighborhood sizes. Optionally, a significantly faster R code that requires significantly less memory and uses some embedded C++ code is available upon request from the author.

## Value

A numeric vector of the same dimension as `nvector` that contains the FSS values at corresponding neighborhood sizes.

## Author(s)

Gregor Skok (Gregor.Skok@fmf.uni-lj.si)

## References

Roberts, N.M., Lean, H.W., 2008. Scale-Selective Verification of Rainfall Accumulations from High-Resolution Forecasts of Convective Events. Mon. Wea. Rev. 136, 78-97.

Faggian N., Roux B., Steinle P., Ebert B., 2015: Fast calculation of the Fractions Skill Score, MAUSAM, 66 (3), 457-466.

## See Also

hoods2d

## Examples

```
# ------------------------------------------
# A simple example with two 500 x 500 fields
# ------------------------------------------

# generate two empty 500 x 500 binary fields where all values are 0
fbin1=matrix(0, 500, 500, byrow = FALSE)
fbin2=fbin1

# in the fields define a single 20x20 non-zero region of precipitation that
# is horizontally displaced in the second field by 100 grid points
fbin1[200:220,200:220]=1
fbin2[200:220,300:320]=1

# specify a vector of neighborhood sizes for which the FSS values are to be calculated
nvector = c(1,51,101,201,301,601,901,1501)

# calulate FSS values
FSSvector=calculate_FSSvector_from_binary_fields(fbin1, fbin2, nvector)

# print FSS values
print(FSSvector)

# The example should output:
#  0.00000000 0.00000000 0.04271484 0.52057596 0.68363656 0.99432823 1.00000000 1.00000000
```

---

`calculate_FSSwind` *Calculate FSSwind Score Value*

---

## Description

Calculates the value for the FSSwind metric that can be used for spatial verification of 2D wind fields.

## Usage

```
calculate_FSSwind(findex1, findex2, nvector)
```

## Arguments

| | |
|---|---|
| findex1 | A numeric matrix representing the first wind class index field. Only integer values larger than 0 are allowed in the matrix. Each integer value corresponds to a certain wind class. |
| findex2 | A numeric matrix representing the second wind class index field. Only integer values larger than 0 are allowed in the matrix. Each integer value corresponds to a certain wind class. The matrix needs to have the same dimensions as findex1. |
| nvector | A numeric vector containing neighborhood sizes for which the FSSwind score value is to be calculated. Only positive odd values are allowed in the vector. |

## Details

The FSSwind is based on the idea of the Fractions Skill Score, a neighborhood-based spatial verification metric frequently used for verifying precipitation. The FSSwind avoids some of the problems of traditional non-spatial verification metrics (the "double penalty" problem and the failure to distinguish between a "near miss" and much poorer forecasts) and can distinguish forecasts even when the spatial displacement of wind patterns is large. Moreover, the time-averaged score value in combination with a statistical significance test enables different wind forecasts to be ranked by their performance (see Skok and Hladnik, 2018, for details).

The score can be used to spatially compare two 2D wind vector fields. In order to calculate the FSSwind value, wind classes have first to be defined. The choice of classes will define how the score behaves and influence the results. The definition of classes should reflect what a user wants to verify. The score will evaluate the spatial matching of the areas of the wind classes. The class definitions should cover the whole phase space of possible wind values (i.e. to make sure every wind vector can be assigned a wind class). It does not make sense choosing a class definition with an overly large number of classes or a definition where a single class would totally dominate over all the other classes. Once the class definition is chosen, the wind vector fields need to be converted to wind class index fields with each class assigned an unique integer value. These wind class index fields serve as input to the calculate_FSSwind function. The FSSwind can have values between 0 and 1 with 0 indicating the worst possible forecast and 1 indicating a perfect forecast. For guidance on correct interpretation of the results and other important considerations please refer to Skok and Hladnik (2018).

The code utilizes the fast method for computing fractions (Faggian et al., 2015). Optionally, a significantly faster R code that requires significantly less memory and uses some embedded C++ code is available upon request from the author.

## Value

A numeric vector of the same dimension as nvector that contains the FSSwind values at corresponding neighborhood sizes.

## Author(s)

Gregor Skok (Gregor.Skok@fmf.uni-lj.si)

## References

Skok, G. and V. Hladnik, 2018: Verification of Gridded Wind Forecasts in Complex Alpine Terrain: A New Wind Verification Methodology Based on the Neighborhood Approach. Mon. Wea. Rev., 146, 63-75, https://doi.org/10.1175/MWR-D-16-0471.1

Faggian N., Roux B., Steinle P., Ebert B., 2015: Fast calculation of the Fractions Skill Score, MAUSAM, 66 (3), 457-466.

## See Also

hoods2d

## Examples

```
# ------------------------------------------
# A simple example with two 500 x 500 fields
# ------------------------------------------

# generate two 500 x 500 wind class index fields where all values are 1 (wind class 1)
findex1=matrix(1, 500, 500, byrow = FALSE)
findex2=findex1

# in the fields generate some rectangular areas with other wind classes (classes 2,3 and 4)
findex1[001:220,200:220]=2
findex1[100:220,300:220]=3
findex1[300:500,100:200]=4

findex2[050:220,100:220]=2
findex2[200:320,300:220]=3
findex2[300:500,300:500]=4

# specify a vector of neighborhood sizes for which the FSSwind values are to be calculated
nvector = c(1,51,101,201,301,601,901,1501)

# calulate FSSwind values
FSSwindvector=calculate_FSSwind(findex1, findex2, nvector)

# print FSSwind values
print(FSSwindvector)

# The example should output:
# 0.6199600 0.6580598 0.7056385 0.8029494 0.8838075 0.9700274 0.9754587 0.9756134
```

---

censqdelta *Centered on Square Domain Baddeley's Delta Metric*

---

## Description

Baddeley's delta metric is sensitive to the position of non-zero grid points within the domain, as well as to the size of the domain. In order to obtain consistent values of the metric across cases, it is recommended to first position the sets to be compared so that they are centered with respect to one another on a square domain; where the square domain is the same for all comparison sets.

## Usage

```
censqdelta(x, y, N, const = Inf, p = 2, ...)
```

## Arguments

| | |
|---|---|
| `x, y` | Matrices representing binary images to be compared. If they are not binary, then they will be forced to binary by setting anything above zero to one. |
| `N` | The size of the square domain. If missing, it will be the size of the largest side, and if it is even, one will be added to it. |
| `const` | single numeric giving the `c` argument to `deltametric`, which is the constant value over which the distance map is reduced to this value. |
| `p` | single numeric giving the `p` argument to `deltametric`, which specifies the type of Lp norm used to calculate the delta maetric. |
| `...` | Not used. |

## Details

Baddeley's delta metric (Baddeley, 1992a,b) is the L_p norm over the absolute difference of distance maps for two binary images, A and B. A concave function (e.g., f(t) = min(t, constant)) may first be applied ot each distance map before taking their absolute differences, which makes the result less sensitive to small changes in one or both images than other similar metrics. The metric is sensitive to size, shape and location differences, which make it very practical for comparing forecasts to observations in terms of the position, area extent, and area shape errors. However, its sensitivity to domain size and position within the domain are undesirable, but are easily fixed by calculating the metric over a consistent, square domain with the combined verification set centerd on that domain. See the example section below to see the issue.

This function essentially takes a window of size N by N and moves so that the centroid of each pair of sets, A and B, is the center of the window before calculating the metric.

Centering and squaring is recommended for carrying out a procedure such as that proposed in Gilleland et al. (2008). Centering on a square domain alleviates the problems discovered by Schwedler and Baldwin (2011) who suggested using a small value of the constant in f(t) = min(t, constant) applied to the distance maps. This solution is not very appealing because of the sensitivity in choice of the constant that generally diminishes as it approaches the domain size (Gilleland, 2011).

After centering the sets on a square domain, the function `deltametric` from package **spatstat** is used to calculate the metric.

## Value

A single numeric value is returned.

## Author(s)

Eric Gilleland

## References

Baddeley, A. (1992a) An error metric for binary images. In *Robust Computer Vision Algorithms*, W. Forstner and S. Ruwiedel, Eds., Wichmann, 59–78.

Baddeley, A. (1992b) Errors in binary images and an Lp version of the Hausdorff metric. *Nieuw Arch. Wiskunde*, **10**, 157–183.

Gilleland, E. (2011) Spatial Forecast Verification: Baddeley's Delta Metric Applied to the ICP Test Cases. *Weather Forecast.*, **26** (3), 409–415.

Gilleland, E. (2017) A new characterization in the spatial verification framework for false alarms, misses, and overall patterns. *Weather Forecast.*, **32** (1), 187–198, DOI: 10.1175/WAF-D-16-0134.1.

Gilleland, E., Lee, T. C. M., Halley Gotway, J., Bullock, R. G. and Brown, B. G. (2008) Computationally efficient spatial forecast verification using Baddeley's delta image metric. *Mon. Wea. Rev.*, **136**, 1747–1757.

Schwedler, B. R. J. and Baldwin, M. E. (2011) Diagnosing the sensitivity of binary image measures to bias, location, and event frequency within a forecast verification framework. *Weather Forecast.*, **26**, 1032–1044.

## See Also

deltametric, locmeasures2d

## Examples

```
x <- y <- matrix( 0, 100, 200 )
x[ 45, 10 ] <- 1
x <- kernel2dsmooth( x, kernel.type = "disk", r = 4 )

y[ 50, 60 ] <- 1
y <- kernel2dsmooth( y, kernel.type = "disk", r = 10 )

censqdelta( x, y )

## Not run:
# Example form Gilleland (2017).
#
# I1 = circle with radius = 20 centered at 100, 100
# I2 = circle with radius = 20 centered at 140, 100
# I3 = circle with radius = 20 centered at 180, 100
# I4 = circle with radius = 20 centered at 140, 140

I1 <- I2 <- I3 <- I4 <- matrix( 0, 200, 200 )

I1[ 100, 100 ] <- 1
I1 <- kernel2dsmooth( I1, kernel.type = "disk", r = 20 )
I1[ I1 > 0 ] <- 1
if( any( I1 < 0 ) ) I1[ I1 < 0 ] <- 0
```

```
I2[ 140, 100 ] <- 1
I2 <- kernel2dsmooth( I2, kernel.type = "disk", r = 20 )
I2[ I2 > 0 ] <- 1
if( any( I2 < 0 ) ) I2[ I2 < 0 ] <- 0

I3[ 180, 100 ] <- 1
I3 <- kernel2dsmooth( I3, kernel.type = "disk", r = 20 )
I3[ I3 > 0 ] <- 1
if( any( I3 < 0 ) ) I3[ I3 < 0 ] <- 0

I4[ 140, 140 ] <- 1
I4 <- kernel2dsmooth( I4, kernel.type = "disk", r = 20 )
I4[ I4 > 0 ] <- 1
if( any( I4 < 0 ) ) I4[ I4 < 0 ] <- 0

image( I1, col = c("white", "darkblue") )
contour( I2, add = TRUE )
contour( I3, add = TRUE )
contour( I4, add = TRUE )

# Each circle is the same size and shape, and the domain is square.
# I1 and I2, I2 and I3, and I2 and I4 are all the same distance
# away from each other.  I1 and I4 and I3 and I4 are also the same distance
# from each other.  I3 touches the edge of the domain.
#

# First, calculate the Baddeley delta metric on each
# comparison.

I1im <- as.im( I1 )
I2im <- as.im( I2 )
I3im <- as.im( I3 )
I4im <- as.im( I4 )

I1im <- solutionset( I1im > 0 )
I2im <- solutionset( I2im > 0 )
I3im <- solutionset( I3im > 0 )
I4im <- solutionset( I4im > 0 )

deltametric( I1im, I2im )
deltametric( I2im, I3im )
deltametric( I2im, I4im )

# Above are all different values.
# Below, they are all 28.84478.
censqdelta( I1, I2 )
censqdelta( I2, I3 )
censqdelta( I2, I4 )

# Similarly for I1 and I4 vs I3 and I4.
deltametric( I1im, I4im )
deltametric( I3im, I4im )
```

```
censqdelta( I1, I4 )
censqdelta( I3, I4 )

# To see why this problem exists.
dm1 <- distmap( I1im )
dm1 <- as.matrix( dm1 )
dm2 <- distmap( I2im )
dm2 <- as.matrix( dm2 )

par( mfrow = c( 2, 2 ) )
image.plot( dm1 )
contour( I1, add = TRUE, col = "white" )
image.plot( dm2 )
contour( I2, add = TRUE, col = "white" )

image.plot( abs( dm1 ) - abs( dm2 ) )
contour( I1, add = TRUE, col = "white" )
contour( I2, add = TRUE, col = "white" )


## End(Not run)
```

---

`centdist` *Centroid Distance Between Two Identified Objects*

---

## Description

Find the centroid distance between two identified objects/features.

## Usage

```
centdist(x, y, distfun = "rdist", loc = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x,y` | objects of class "owin" (package **spatstat**) containing binary images of features of interest. |
| `distfun` | character string naming a distance function that should take arguments `x1` and `x2` as 1 by 2 matrices, and return a single numeric value. Default uses the **fields** function, `rdist`, where the fields function `rdist.earth` is an obvious alternative. |
| `loc` | two-column matrix giving the location values for which to calculate the centroids. If NULL, indices according to the dimension of the fields are used. |
| `...` | optional arguments to `distfun`. |

## Details

This is a simple function that calculates the centroid for each of x and y (to get their centroids), and then finds the distance between them according to `distfun`. The centroids are calculated using `FeatureProps`.

## Value

numeric giving the centroid distance.

## Author(s)

Eric Gilleland

## See Also

`FeatureProps`, `as.im`, `solutionset`, `FeatureMatchAnalyzer`, `FeatureComps`

## Examples

```
x <- y <- matrix(0, 10, 12)
x[2:3,c(3:6, 8:10)] <- 1
y[c(4:7, 9:10),c(7:9, 11:12)] <- 1

x <- as.im(x)
x <- solutionset(x>0)
y <- as.im(y)
y <- solutionset(y>0)
centdist(x,y)
```

---

# Cindex *Connectivity Index*

---

## Description

Calculate the connectivity index of an image.

## Usage

```
Cindex(x, thresh = NULL, connect.method = "C", ...)

## Default S3 method:
Cindex(x, thresh = NULL, connect.method = "C", ...)

## S3 method for class 'SpatialVx'
Cindex(x, thresh = NULL, connect.method = "C", ...,
    time.point = 1, obs = 1, model = 1)
```

## Arguments

| | |
|---|---|
| x | Default: m by n numeric matrix giving the field for which the shape index is to be calculated.<br>Sindex.SpatialVx: list object of class "SpatialVx". |
| thresh | Set values under (strictly less than) this threshold to zero, and calculate the connectivity index for the resulting image. If NULL, no threshold is applied. |
| connect.method | character string giving the method argument for the connected function of package **spatstat**. This must be one of "C" or "interpreted". See the help file for connected for more details. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| ... | Not used. |

## Details

The connectivity index is introduced in AghaKouchak et al. (2011), and is designed to automatically determine how connected an image is. It is defined by

Cindex = 1 - (NC - 1)/(sqrt(NP) + NC),

where 0 <= Cindex <= 1 is the connectivity index (values close to zero are less connected, and values close to 1 are more connected), NP is the number of nonzero pixels, and NC is the number of isolated clusters.

The function connected from package **spatstat** is used to identify the number of isolated clusters.

## Value

numeric giving the connectivity index.

## Author(s)

Eric Gilleland

## References

AghaKouchak, A., Nasrollahi, N., Li, J., Imam, B. and Sorooshian, S. (2011) Geometrical characterization of precipitation patterns. *J. Hydrometerology*, **12**, 274–285, doi:10.1175/2010JHM1298.1.

## See Also

connected, as.im, Sindex, Aindex

## Examples

```
data( "geom000" )
Cindex( geom000 )

data( "pert000" )
Cindex( pert000 )

## Not run:
# Two separate areas with highly structured shapes, but far away from each other.
data( "pert006" )
data( "ICPg240Locs" )

hold <- make.SpatialVx(pert000, pert006, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert006" )

plot( hold )

Cindex( hold )

## End(Not run)
```

---

clusterer *Cluster Analysis Verification*

---

## Description

Perform Cluster Analysis (CA) verifcation per Marzban and Sandgathe (2006).

## Usage

```
clusterer(X, Y = NULL, ...)

## Default S3 method:
clusterer(X, Y = NULL, ..., xloc = NULL, xyp = TRUE, threshold = 1e-08,
    linkage.method = "complete", stand = TRUE, trans = "identity",
    a = NULL, verbose = FALSE)

## S3 method for class 'SpatialVx'
clusterer(X, Y = NULL, ..., time.point = 1, obs = 1, model = 1, xyp = TRUE,
    threshold = 1e-08, linkage.method = "complete", stand = TRUE,
    trans = "identity", verbose = FALSE)

## S3 method for class 'clusterer'
plot(x, ..., mfrow = c(1, 2), col = c("gray", tim.colors(64)),
```

```
    horizontal = FALSE)

## S3 method for class 'summary.clusterer'
plot(x, ...)

## S3 method for class 'clusterer'
print(x, ...)

## S3 method for class 'clusterer'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| X,Y | clusterer default method, these are m by n matrices giving the verification and forecast fields, resp.<br>"SpatialVx" method function, X is an object of class "SpatialVx" and Y is not used (a warning is given if it is not missing and not NULL). |
| object,x | list object of class "clusterer" as returned by clusterer (or summary.clusterer in the case of plot.summary.clusterer). |
| xloc | (optional) numeric mn by 2 matrix giving the gridpoint locations. If NULL, this will be created using 1:m and 1:n. |
| xyp | logical, should the cluster analysis be performed on the locations and intensities (TRUE) or only the locations (FALSE)? |
| threshold | numeric of length one or two giving the threshold to apply to each field (>=). If length is two, the first value corresponds to the threshold for the verification field, and the second to the foreast field. |
| linkage.method | character naming a valid linkage method accepted by hclust. |
| stand | logical, should the data matrices consisting of xloc and each field first be standardized before performing cluster analysis? |
| trans | character naming a function to be applied to the field intensities before performing the CA. Only used if xyp is TRUE. Default applies no transformation. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| a | (optional) list giving object attributes associated with a "SpatialVx" class object. The clusterer method for "SpatialVx" objects calls the default method function, and uses this argument to pass the attributes through to the final returned object, as well as to grab location information. |
| mfrow | mfrow parameter (see help file for par). If NULL, then the parameter is not re-set. |
| col | color vector for image plots of fields after applying the threshold(s). |
| horizontal | logical, should the image plot color legend be placed horizontally or vertically? Only for image plot sof the fields. |
| verbose | logical, should progress information be printed to the screen? |

`...` optional arguments to the `hclust` function. In the case of the `summary` method function, `z` and/or `sigma` giving a numeric value used to find the cut-off given by median + z*sigma for detemining matched objects (see Marzban and Sandgathe 2006) where defaults of 1 and the standard deviation of minimum inter-cluster distances are used, and `silent` (logical should information be printed to the screen (FALSE) or not (TRUE); default is to print to the screen. In the case of the `plot` method functions, these are optional arguments to the `summary` method function.

## Details

This function performs cluster analysis (CA) on positive values from each of two fields in a verification set using the hclust function from package fastcluster. Inter-cluster distances are computed between each cluster of each field at every level of the CA. The function clusterer performs CA on both fields, and finds the inter-cluster distances across fields for every possible combination of objects at each iteration of each CA. The summary method function finishes the analysis by determining hits, misses and false alarms as well as the numbers of clusters. It also computes CSI for each number of cluster combinations. This is the verification approach described in Marzban and Sandgathe (2006).

The `plot` method function creates a 4 by 2 panel of plots. The top two plots give image plots of the verification and forecast fields with grid points below the threshold(s) showing zero. The next two plots are dendrograms as performed by the plot method function for `hclust` (`dendrogram`) objects. The next row gives a histogram of the minimum inter-cluster distances, then box plots showing the hits, misses and false alarms for every possible combination of levels of each CA. Finally, the bottom two plots show, for each combination of CA level (i.e., numbers of clusters), the CSI and average error (inter-cluster distance) for all matched objects. These last three plots are the ones made by the plot method for values returned from the `summary` method function.

`print` is currently not very useful here, but it prevents printing a big mess to the screen.

## Value

A list object of class "clusterer" is returned with components:

`linkage.method` character vector of length one or two giving the linkage method as passed into the function. The length is two only if the McQuitty method is chosen in which case this method is used for the CA, but not for the inter-cluster differencs across fields (average is used for that instead).

`trans` character naming the transformation function applied to the intensities.

`N` numeric giving the size of the fields.

`threshold` numeric of length two giving the threshold applied to each field.

`NCo,NCf` numeric vectors giving the number of clusters at each iteration of the CA for the verification and forecast fields, resp.

`cluster.identifiers`
a list with components X and Y giving lists of lists identifying specific CA components at each level of the CA for both fields.

`idX,idY` logical vectors describing which grid points were included in the CA for each field (i.e., which grid points were >= threshold and had non-missing values).

`cluster.objects`
: a list with components X and Y giving the objects returned by hclust for each field.

`inter.cluster.dist`
: a list of list objects with NCf by NCo matrix components giving the inter-cluster distances (between verification and forecast fields) for each iteration of CA for each field.

`min.intercluster.dists`
: numeric vector givng the minimum values inter.cluster.dist at each iteration. Used to determine the cut-off for matched objects.

The summary method function returns a list with the same components as above, but also the components:

`cutoff` The cut-off value used for determining matches.

`csi,AvgErr` NCo by NCf numeric matrix giving the critical success index (CSI) and average intercluster error (distance) based on matched/un-matched objects.

`HMF` NCo by NCf by 3 array giving the hits, misses and false alarms based on matched/un-matched objects.

If the argument a is not NULL, then these are returned as attributes of the returned object. In the case of "SpatialVx" objects, the attributes are preserved.

plot and print methods do not return anything.

## Warning

Although some effort has been put into making the functions in this package as computationally efficient as possible, there is a lot of bookeeping involved with this approach, and the current functions are probably not as efficient as they could be. In any case, they will likely be slow for large data sets. The function can work quickly on large fields if an adequately high threshold is used (e.g., if threshold=10 is replaced for 16 in the not run example below, the function is VERY slow). Performing the actual cluster analysis on each field is fast because the hclust function from the fastcluster package is used, which works very well. However, bookeeping after the CA is done employs a lot of loops within loops, which possibly can be made more efficient (and maybe someday will be), but for now...

If it is desired to simply look at the CA for the two fields, the function hclust from fastcluster can be used, which essentially replaces the hclust function from the stats package with a faster version, but otherwise operates the same as far as what is returned, etc., and the same method functions can be employed.

## Note

Contact Caren Marzban, marzban "at" u.washington.edu, for questions about the method, and Eric Gilleland, ericg "at" ucar.edu, for problems with the code.

## Author(s)

Eric Gilleland

## References

Marzban, C. and Sandgathe, S. (2006) Cluster analysis for verification of precipitation fields. *Wea. Forecasting*, **21**, 824–838.

## See Also

hclust, hclust, as.dendrogram, cutree, make.SpatialVx, CSIsamples

## Examples

```
data( "UKobs6" )
data( "UKfcst6" )
look <- clusterer(X=UKobs6, Y=UKfcst6, threshold=16, trans="log", verbose=TRUE)
plot( look )

## Not run:
data( "UKloc" )

# Now, do the same thing, but using a "SpatialVx" object.
hold <- make.SpatialVx( UKobs6, UKfcst6, loc = UKloc, map = TRUE,
    field.type = "Rainfall", units = "mm/h",
    data.name = "Nimrod", obs.name = "obs 6", model.name = "fcst 6" )

look2 <- clusterer(hold, threshold=16, trans="log", verbose=TRUE)
plot( look2 )
# Note that values differ because now we're using the
# actual locations instead of integer indicators of
# positions.

## End(Not run)
```

---

# combiner *Combine Features/Matched Objects*

---

## Description

Combine two or more features or matched class objects into one object for aggregation purposes.

## Usage

```
combiner(...)
```

## Arguments

| | |
|---|---|
| ... | Two or more objects of class "features" or "matched" (can also be a list of these objects). |

### Details

Useful for functions such as `compositer` and/or (coming soon) aggregating results for feature-based methods.

### Value

A list object of class "combined" with the same components as the input arguments, but where some components (namely, X, Xhat, X.labeled, Y.labeled) are now arrays containing these values from each combined object. The lists of lists contained in the X.feats and Y.feats include one long list of lists containing all of the individual features from each object.

### Author(s)

Eric Gilleland

### See Also

Functions that create objects of class "features" and "matched": `FeatureFinder`, `centmatch`, and `deltamm`.

### Examples

```
# TO DO
```

---

## compositer *Create Composite Features*

---

### Description

After identifying features in a verification set, re-grid them so that their centroids are all the same, and the new grid is as small as possible to completely contain all of the features in the verification set.

### Usage

```
compositer(x, level = 0, verbose = FALSE, ...)

## S3 method for class 'features'
compositer(x, level = 0, verbose = FALSE, ...)

## S3 method for class 'matched'
compositer(x, level = 0, verbose = FALSE, ...)

## S3 method for class 'combined'
compositer(x, level = 0, verbose = FALSE, ...)

## S3 method for class 'composited'
plot(x, ..., type = c("all", "X", "Xhat", "X|Xhat", "Xhat|X"),
    dist.crit = 100, FUN = "mean", col = c("gray", tim.colors(64)))
```

## Arguments

| | |
|---|---|
| x | compositer: an object of class "features", "matched", or "combined".<br>plot: an object of class "composited". |
| type | character, stating which composite features should be plotted. Default makes a two by two panel of plots with all of the choices. |
| dist.crit | maximum value beyond which any minimum centroid distances are considered too far for features to be "present" in the area. |
| FUN | name of a function to be applied to the composites in order to give the distributional summary. |
| col | color pallette to be used. |
| level | numeric used in shrinking the grid to a smaller size. |
| verbose | logical, should information be printed to the screen? |
| ... | Not used by compositer.<br>plot: Optional arguments to image.plot (only for the scale legend), such as legend.only, legend.lab, legend.mar, horizontal, etc. |

## Details

This is functionality for performing an analysis similar to the composite verification method of Nachamkin (2004). See also Nachamkin et al. (2005) and Nachamkin (2009). The main difference is that this function centers all features to the same point, then re-sizes the grid to the smallest possible size to contain all features. The "existence" of a feature at the same time point is determined by the centroid distance (because, here, the compositing is done for a large field rather than a small area), but it does not allow for having half of the feature in the domain in order to be considered.

compositer takes an object of class "features" or "matched" and centers all of the identified features onto the same point so that all features have the same centroid. It also then re-grids the composited features so that they are contained on the smallest possible domain that includes all of the features in the verification set.

Generally, because the composite approach is distributional in nature, it makes sense to look at features across multiple time points. The function combiner allows for combining features from more than one object of class "features" or "matched" in order to subsequently run with compositer.

plot takes the composite features and adds them together creating a density of the composite features, then, Depending on the type argument, the verification (type = "X"), model (type = "Xhat"), verification conditioned on the model (type = "X|Xhat"), or the model conditioned on the verification composite features are plotted. In the case of type = "all", then a panel of four plots are made with all of these choices. In the case of the conditional plots, the sum of composites for one field are masked out so that only the density of the other field is plotted where composited features from the first field exist.

## Value

A list object of class "composited" is returned with all of the same components and attributes as the x argument, but with additional components:

`distances` List with components X and Xhat giving the minimum centroid distances from each feature in X (Xhat) to a feature in the other field (used for determining the conditional distributions; i.e., a feature is present if its centroid distance is less than some pre-specified amount).

`Xcentered, Ycentered` list of "owin" objects containing each feature similar to X.feats and Y.feats, but centered on the same spot and re-gridded

## Note

centroids are rounded to the nearest whole number so that interpolation is not necessary. This may introduce a slight bias in results, but it should not be a major issue.

## Author(s)

Eric Gilleland

## References

Nachamkin, J. E. (2004) Mesoscale verification using meteorological composites. *Mon. Wea. Rev.*, **132**, 941–955.

Nachamkin, J. E. (2009) Application of the Composite Method to the Spatial Forecast Verification Methods Intercomparison Dataset. *Wea. Forecasting*, **24** (5), 1390–1400, DOI: 10.1175/2009WAF2222225.1.

Nachamkin, J. E., Chen, S. and Schmidt, J. S. (2005) Evaluation of heavy precipitation forecasts using composite-based methods: A distributions-oriented approach. *Mon. Wea. Rev.*, **133**, 2163–2177.

## See Also

Identifying features: `FeatureFinder`

## Examples

```
x <- y <- matrix(0, 100, 100)
x[2:3,c(3:6, 8:10)] <- 1
y[c(4:7, 9:10),c(7:9, 11:12)] <- 1

x[30:50,45:65] <- 1
y[c(22:24, 99:100),c(50:52, 99:100)] <- 1

hold <- make.SpatialVx( x, y, field.type = "contrived", units = "none",
    data.name = "Example", obs.name = "x", model.name = "y" )

look <- FeatureFinder(hold, smoothpar=0.5)

look2 <- compositer(look)
plot(look2, horizontal = TRUE)

## Not run:
data( "pert000" )
```

```
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004,
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000", model.name = "pert004" )

look <- FeatureFinder(hold, smoothpar=10.5, thresh = 5)
plot(look)

look2 <- compositer(look)
plot(look2, horizontal = TRUE)

# TO DO: show examples with combiner.

## End(Not run)
```

---

craer *Contiguous Rain Area*

---

## Description

Calculate the contiguous rain area (CRA) for matched features.

## Usage

```
craer(x, type = c("regular", "fast"), rotate = FALSE, loss, loss.args = NULL,
    interp = "bicubic", method = "BFGS", stages = TRUE, verbose = FALSE, ...)

## S3 method for class 'craered'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | craer: A list object of class "matched".<br>print: Object returned by craer. |
| type | character string that must be one of "regular" or "fast". See details section for more information. |
| rotate | logical, should the features also be rotated into alignement (in addition to their being translated)? |
| loss | character string naming a function to calculate the loss function when using type "regular". Default uses QlossRigid, which is based on an assumption of iid normally distributed errors (only important if you try to obtain CIs for the rigid transformations, which is currently not readily available), but is similar to squared error loss. See QcorrRigid for another option based on correlations. This argument is ignored if type is "fast". |

| | |
|---|---|
| loss.args | list object with optional arguments for loss. Only used if type is "fast". |
| interp | character string naming the interpolation method used in calls to Fint2d. Must be one of "round", "bilinear" or "bicubic". Only used if type is "fast". |
| method | character string naming the numerical optimization method to use (see optim for choices). Only used if type is "fast". |
| stages | logical, if rotate is TRUE, then should the optimal translation be found first, and then further try to find the optimal translation and rotation using the results from the initial optimization? The idea is to help the optimization, but not clear that it matters. |
| verbose | logical, should progress information be printed to the screen? |
| ... | Optional arguments to optim besides method. Not used by print. |

## Details

The contiguous rain area (CRA) spatial verification method was first introduced by Ebert and McBride (2000). After identifying features and matching them across fields using your favorite feature identification and feature matching functions, the method attempts to find an optimal rigid transformation between the matched features, and then applies the mean square error (MSE) breakdown given by:

MSE( total ) = MSE( displacement ) + MSE( volume ) + MSE( pattern ),

where MSE( displacement ) = MSE( total ) - MSE( shifted ), and MSE( shifted ) is the MSE between the rigidly transformed forecast field and the observed one. MSE( volume ) is the squared bias (differences in means) between the rigidly transformed forecast and the observed field. MSE( pattern ) is the difference between MSE( shift ) and MSE( volume ). In the case of rotations with stages = FALSE, these values are the same, but the displacement also incorporates the rotation. It is more complicated if this argument is TRUE. In this case, the values are as with its being FALSE, but some additional MSE breakdowns are given for the translation only situation, but MSE( volume ) and MSE( pattern ) are based on the translation and rotation together.

Finding the optimal rigid transformation between two features can be difficult. Two options are supplied here. The first (type = "regular") attempts to find the optimal rigid transformation according to a given loss function via numerical optimization of the loss function. The second (type = "fast") uses the image moments to find the distance between feature centroids (as the optimal translation) and difference between feature major axis angles (as the optimal rotation).

In general, Ebert and McBride (2000) only considered translations (not rotations), and it is unclear whether or not the addition of rotations is worthwhile.

## Value

Returns a matrix of class "craered" with named columns specifying the results for each feature match in the order they were supplied via the "matched" list object.

## Note

The rigid transformation is performed by the rigider function, which is experimental at this stage in the game.

## Author(s)

Eric Gilleland

## References

Ebert, E. E. and McBride, J. L. (2000) Verification of precipitation in weather systems: determination of systematic errors. *J. Hydrology*, **239**, 179–202.

## See Also

optim, imomenter

Feature identification function: FeatureFinder

For some feature matching functions, see:

centmatch, deltamm, minboundmatch

For finding the optimal rigid transformations:

rigider, imomenter

For rigidly transforming a field: rigidTransform

Two-dimensional interpolation (useful because the transformations typically do not map exactly to a grid square):

Fint2d

## Examples

```
x <- y <- matrix(0, 100, 100)
x[2:3,c(3:6, 8:10)] <- 1
y[c(4:7, 9:10),c(7:9, 11:12)] <- 1

x[30:50,45:65] <- 1
y[c(22:24, 99:100),c(50:52, 99:100)] <- 1

hold <- make.SpatialVx( x, y, field.type = "contrived", units = "none",
    data.name = "Example", obs.name = "x", model.name = "y" )

look <- FeatureFinder(hold, smoothpar=0.5)

look2 <- minboundmatch( look, type = "multiple", mindist = 20 )
look2 <- MergeForce( look2 )

craer( look2, type = "fast", verbose = TRUE)

## Not run:
data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004,
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
```

```
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- FeatureFinder(hold, smoothpar=10.5, thresh = 5)
plot(look)

look2 <- minboundmatch(look, verbose = TRUE)
plot(look2)

craer( look2 )

## End(Not run)
```

---

CSIsamples *Forecast Verification with Cluster Analysis: The Variation*

---

## Description

A variation on cluster analysis for forecast verification as proposed by Marzban and Sandgathe (2008).

## Usage

```
CSIsamples(x, ...)

## Default S3 method:
CSIsamples(x, ..., xhat, nbr.csi.samples = 100, threshold = 20,
    k = 100, width = 25, stand = TRUE, z.mult = 0, hit.threshold = 0.1,
    max.csi.clust = 100, diss.metric = "euclidean", linkage.method = "average",
    verbose = FALSE)

## S3 method for class 'SpatialVx'
CSIsamples(x, ..., time.point = 1, obs = 1, model = 1, nbr.csi.samples = 100,
    threshold = 20, k = 100, width = 25, stand = TRUE, z.mult = 0,
    hit.threshold = 0.1, max.csi.clust = 100, diss.metric = "euclidean",
    linkage.method = "average", verbose = FALSE)

## S3 method for class 'CSIsamples'
summary(object, ...)

## S3 method for class 'CSIsamples'
plot(x, ...)

## S3 method for class 'summary.CSIsamples'
plot(x, ...)

## S3 method for class 'CSIsamples'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x,xhat | default method: matrices giving the verification and forecast fields, resp.<br>"SpatialVx" method: x is an object of class "SpatialVx".<br>plot, print methods: list object of class "CSIsamples" or "summary.CSIsamples" (in the case of plot). |
| object | list object of class "CSIsamples". |
| nbr.csi.samples | integer giving the number of samples to take at each level of the CA. |
| threshold | numeric giving a value over which is to be considered an event. |
| k | numeric giving the value for centers in the call to kmeans. |
| width | numeric giving the size of the samples for each cluster sample. |
| stand | logical, should the data first be standardized before applying CA? |
| z.mult | numeric giving a value by which to multiply the z- component. If zero, then the CA is performed on locations only. Can be used to give more or less weight to the actual values at these locations. |
| hit.threshold | numeric between zero and one giving the threshold for the proportion of a cluster that is from the verification field vs the forecast field used for determining whether the cluster consitutes a hit (vs false alarm or miss depending). |
| max.csi.clust | integer giving the maximum number of clusters allowed. |
| diss.metric | character giving which method to use in the call to dist (which dissimilarity metric should be used?). |
| linkage.method | character giving the name of a linkage method acceptable to the method argument from the hclust function of package **fastcluster**. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| verbose | logical, should progress information be printed to the screen? |
| ... | Not used by CSIsamples method functions.<br>summary method function: the argument silent may be specified, which is a logical stating whether to print the information to the screen (FALSE) or not (TRUE). If not given, the summary information will be printed to the screen.<br>Not used by the plot method function. |

## Details

This function carries out the procedure described in Marzban and Sandgathe (2008) for verifying forecasts. Effectively, it combines the verification and forecast fields (keeping track of which values belong to which field) and applies CA to the combined field. Clusters identified with a proportion of values belonging to the verification field within a certain range (defined by the hit.threshold argument) are determined to be hits, misses or false alarms. From this information, the CSI (at each number of clusters; scale) is calculated. A sampling scheme is used to speed up the process.

The plot and summary functions all give the same information, but in different formats: i.e., CSI by number of clusters (scale).

## Value

A list is returned by CSIsamples with components:

| | |
|---|---|
| `data.name` | character vector giving the names of the verification and forecast fields analyzed, resp. |
| `call` | an object of class "call" giving the function call. |
| `results` | max.csi.clust by nbr.csi.samples matrix giving the caluclated CSI for each sample and iteration of CA. |

The summary method function invisibly returns the same list, but with the additional component:

| | |
|---|---|
| `csi` | vector of length max.csi.clust giving the sample average CSI for each iteration of CA. |

The plot method functions do not return anything. Plots are created.

## Note

Special thanks to Caren Marzban, marzban "at" u.washington.edu, for making the CSIsamples (originally called csi.samples) function available for use with this package.

## Author(s)

Hillary Lyons, h.lyons "at" comcast.net, and modified by Eric Gilleland

## References

Marzban, C., Sandgathe, S. (2008) Cluster Analysis for Object-Oriented Verification of Fields: A Variation. *Mon. Wea. Rev.*, **136**, (3), 1013–1025.

## See Also

hclust, hclust, kmeans, clusterer

## Examples

```
## Not run:
grid<- list( x= seq( 0,5,,100), y= seq(0,5,,100))
obj<-Exp.image.cov( grid=grid, theta=.5, setup=TRUE)
look<- sim.rf( obj)
look2 <- sim.rf( obj)

res <- CSIsamples(x=look, xhat=look2, 10, threshold=0, k=100,
                  width=2, z.mult=0, hit.threshold=0.25, max.csi.clust=75)
plot(res)
y <- summary(res)
plot(y)

## End(Not run)
## Not run:
data(UKfcst6)
```

```
data(UKobs6)
data(UKloc)

hold <- make.SpatialVx(UKobs6, UKfcst6, thresholds=0,
    loc=UKloc, map=TRUE, field.type="Rainfall", units="mm/h",
    data.name = "Nimrod", obs.name = "obs 6", model.name = "fcst 6" )

res <- CSIsamples( hold, threshold = 0, k = 200, z.mult = 0.3, hit.threshold = 0.2,
                  max.csi.clust = 150, verbose = TRUE)
plot( res )

summary( res )

y <- summary( res )

plot( y )

## End(Not run)
```

---

`deltamm` *Merge and/or Match Identified Features Within Two Fields*

---

## Description

Merge and/or match identified features within two fields using the delta metric method described in Gilleland et al. (2008), or the matching only method of Davis et al. (2006a).

## Usage

```
deltamm(x, p = 2, max.delta = Inf, const = Inf, type = c( "sqcen", "original" ),
    N = NULL, verbose = FALSE, ...)

centmatch(x, criteria = 1, const = 14, distfun = "rdist", areafac = 1,
    verbose = FALSE, ...)

## S3 method for class 'matched'
plot(x, mfrow = c(1, 2), ...)

## S3 method for class 'matched'
print(x, ...)

## S3 method for class 'matched'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| `x` | For `deltamm` and `centmatch`, an object of class "features" with components `X.labeled`, `Y.labeled` (matrices with numbered features), as well as, `X.feats` |

and `Y.feats`, each of which are list objects containing numbered components within which are objects of class "owin" containing logical matrices that define features for the forecast (Y) and verification (X) fields, resp. For example, as returned by the `FeatureFinder` function. For `plot` an object of class "matched". Argument `y` is used if it is not NULL, otherwise argument `x` is used (but only one of `x` or `y` is used). If `x` and `y` are missing, but not `object`, then `object` will be used, in which case it must be of class "features".

`object` list object of class "matched".

`p` Baddeley delta metric parameter. A value of 1 gives arithmetic averages, Inf gives the Hausdorff metric and -Inf gives a minimum. The default of 2 is most common.

`max.delta` single numeric giving a cut-off value for delta that disallows two features to be merged or matched if the delta between them is larger than this value.

`const` `deltamm`: a constant value over which the shortest distances in a distance map are set. See Gilleland (2011) and Schwedler and Baldwin (2011) for more information about this parameter.

`type` character specifying whether Baddeley's delta metric should be calculated after centering object pairs on a new square grid (default) or performed in their original positions on the original grid.

`N` If `type` is "sqcen", then `N` is the argument to `censqdelta` that specifies the common grid size. If not specified the maximum side of the original domain is used (possibly adding one first to make it an odd number). It is possible that values could be pushed off the new grid, and making `N` larger might alleviate the issue. `centmatch` numeric giving the number of grid squares whereby if the centroid distance (D) is less than this value, a match is declared (only used if `criteria` is 3.

`criteria` 1, 2 or 3 telling which criteria for determining a match based on centroid distance, D, to use. The first (1) is a match if D is less than the sum of the sizes of the two features in question (size is the square root of the area of the feature). The second is a match if D is less than the average size of the two features in question. The third is a match if D is less than a constant given by the argument `const`.

`distfun` character string naming a distance function. Default uses `rdist` from the **fields** package.

`areafac` single numeric used to multiply by grid-space based area in order to at least approximate the correct distance (e.g., using the ICP test cases, 4 would make the areas approximately square km instead of grid points). This should not be used unless `distfun` is "rdist.earth" in which case it will use the locations given in the call to `make.SpatialVx`, which are assumed to be lat/lon coordinates.

`mfrow` mfrow parameter (see help file for `par`). If NULL, then the parameter is not re-set.

`verbose` logical, should progress information be printed to the screen?

`...` For `deltamm`: additional optional arguments to the `distmap` function from package **spatstat**. For `centmatch`: optional arguments to `distfun`. For `plot.matched`, additional arguments to `image.plot` concerning the color legend bar only.

## Details

`deltamm`:

Gilleland et al. (2008) describe a method for automatically merging, and simultaneously, matching identified features within two fields (a verification set). The method was proposed with the general method for spatial forecast verification introduced by Davis et al. (2006 a,b) in mind. It relies heavily on use of a binary image metric introduced by Baddeley (1992a,b) for comparing binary images; henceforth referred to as the delta metric, or just delta.

The procedure is as follows. Suppose there are m identified forecast features and n identified verification features.

1. Compute delta for each feature identified in the forecast field against each feature identified in the verification field. Store these values in an m by n matrix, Upsilon.

2. For each of the m rows of Upsilon, rank the values of delta to identify the features, j_1, ..., j_n that provide the lowest (best) to highest (worst) value, and do the same for each of the n columns to find the forecast features i1, ...,i_m that yield the lowest to highest values for each verification feature.

3. Create a new m by n matrix, Psi, whose columns contain delta computed between each of the individual features in the forecast and (first column) the corresponding j_1 feature from the verification field, and each successive column, k, has delta between the i-th forecast feature and the union of j_1, j_2, ..., j_k.

4. Create a similar m by n matrix, Ksi, that has delta computed between each individual feature in the verification field and the successively bigger unions i_1, ..., i_l for the l-th column.

5. Let Q=[Upsilon, Psi, Ksi], and merge and match features based on the rankings of delta in Q. That is, find the smallest delta in Q, and determine which mergings (if any) and matchings correspond to this value. Remove the appropriate row(s) and column(s) of Q corresponding to the already determined matchings and/or mergings. Repeat this until all features in at least one field have been exhausted.

The above algorithm suffers from two deficiencies. First, features that are merged in one field cannot be matched to merged features in another field. One possible remedy for this is to run this algorithm twice, though this is not a universally good solution. Second, features can be merged and/or matched to features that are very different from each other. A possible remedy for this is to use the cut-off argument, max.delta, to disallow mergings or matchings between features whose delta value is not <= this cut-off. In practice, these two deficiencies are not likely very problematic.

`centmatch`:

This function works similarly as `deltamm`, though it does not merge features. It is based on the method proposed by Davis et al. (2006a). It is possible for more than one object to be matched to the same object in another field. As a result, when plotting, it might appear that features have been merged, but they have not been. For informational purposes, the criteria, appelled `criteria.values` (as determined by the `criteria` argument), along with the centroid distance matrix, appelled `centroid.distances`, are returned.

`plot`: The plot method function for matched features plots matched features across fields in the same color using `rainbow`. Unmatched features in either field are all colored gray. Zero values are colored white. The function `MergeForce` must first be called, however, in order to organize the object into a format that allows the `plot` method function to determine the correct color coding.

The `print` method function will tell you which features matched between fields, so one can plot the originally derived features (e.g., from `FeatureFinder`) to identify matched features.

`summary`:

The summary method function so far simply reverts the class back to "features" and calls that summary function.

## Value

A list object of class "matched" is returned by both centmatch and deltamm containing several components added to the value of x or y passed in, and possibly with attributes inhereted from object.

`match.message` A character string stating how features were matched.

`match.type` character string naming the matching function used.

`matches` two-column matrix with forecast object numbers in the first column and corresponding matched observed features in the second column. If no matches, this will have value integer(0) for each column giving a matrix with dimension 0 by 2.

`unmatched` list with components X and Xhat giving the unmatched object numbers, if any, from the observed and forecast fields, resp. If none, the value will be integer(0).

`Q` (deltamm only) an array of dimension n by m by 3 giving all of the delta values that were computed in determining the mergings and matchings.

`criteria` (centmatch only) 1, 2, or 3 as given by the criteria argument.

`criteria.values, centroid.distances`
(centmatch only) matrices giving the forecast by observed object criteria and centroid distances.

`implicit.merges`
(centmatch only) list displaying multiple matches for each field (this could define potential merges). Each component of the list is a unified set of matched features in the form of two-column matrices analogous to the matches component. If there are no implicit mergings or no matched features, this component will be named, but also NULL. Note: such implicit mergings may or may not make physical sense, and are not considered to be merged generally, but will show up as having been merged/clustered when plotted.

If the argument 'object' is passed in, then the list object will also contain nearly the same attributes, with the data.name attribute possibly changed to reflect the specific model used. It will also contain a time.point and model attribute.

## Author(s)

Eric Gilleland

## References

Baddeley, A. (1992a) An error metric for binary images. In *Robust Computer Vision Algorithms*, W. Forstner and S. Ruwiedel, Eds., Wichmann, 59–78.

Baddeley, A. (1992b) Errors in binary images and an Lp version of the Hausdorff metric. *Nieuw Arch. Wiskunde*, **10**, 157–183.

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006a) Object-based verification of precipitation forecasts, Part I: Methodology and application to mesoscale rain areas. *Mon. Wea. Rev.*, **134**, 1772–1784.

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006b) Object-based verification of precipitation forecasts, Part II: Application to convective rain systems. *Mon. Wea. Rev.*, **134**, 1785–1795.

Gilleland, E. (2011) Spatial Forecast Verification: Baddeley's Delta Metric Applied to the ICP Test Cases. *Wea. Forecasting*, **26** (3), 409–415.

Gilleland, E., Lee, T. C. M., Halley Gotway, J., Bullock, R. G. and Brown, B. G. (2008) Computationally efficient spatial forecast verification using Baddeley's delta image metric. *Mon. Wea. Rev.*, **136**, 1747–1757.

Schwedler, B. R. J. and Baldwin, M. E. (2011) Diagnosing the sensitivity of binary image measures to bias, location, and event frequency within a forecast verification framework. *Wea. Forecasting*, **26**, 1032–1044.

## See Also

To identify features: `FeatureFinder`

`minboundmatch` is another feature matching function.

To force merges (implicit or otherwise): `MergeForce`,

Other functions used to identify features within the above mentioned functions (all from **spatstat**):

`disjointer`, `deltametric`, `owin`, `tess`, `tiles`, `connected`

## Examples

```
## Not run:
x <- y <- matrix(0, 100, 100)
x[2:3,c(3:6, 8:10)] <- 1
y[c(4:7, 9:10),c(7:9, 11:12)] <- 1

x[30:50,45:65] <- 1
y[c(22:24, 99:100),c(50:52, 99:100)] <- 1

hold <- make.SpatialVx( x, y, field.type = "contrived", units = "none",
    data.name = "Example", obs.name = "x", model.name = "y" )

look <- FeatureFinder( hold, smoothpar = 0.5 )

# The next line fails because the centering pushes one object out of the new domain.
# look2 <- deltamm( look )
# Setting N larger fixes the problem.
look2 <- deltamm( look, N = 300 )
look2 <- MergeForce( look2 )

look2

plot( look2 )

FeatureTable(look2)
```

```
look3 <- centmatch(look)

FeatureTable(look3)

look3 <- MergeForce( look3 )

plot( look3 )

## End(Not run)
## Not run:
data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004,
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004")

look <- FeatureFinder(hold, smoothpar = 10.5, thresh = 5 )

plot( look )

look2 <- deltamm( look, N = 701, verbose = TRUE )

look2

look2 <- MergeForce( look2 )

look2

plot( look2 )

# No false alarm objects, one missed object.
FeatureTable( look2 )


## End(Not run)
```

---

`disjointer` *Identify Disjoint Sets of Connected Components*

---

## Description

Identify disjoint sets of contiguous events in a binary field. In many areas of research, this function finds connected components.

## Usage

```
disjointer(x, method = "C")
```

## Arguments

| | |
|---|---|
| x | A numeric matrix or other object that as.im from package **spatstat** works on. |
| method | Same argument as that in connected from package spatstat. |

## Details

disjointer essentially follows the help file for connected to produce a list object where each component is an image describing one set of connected components (or blobs). It is essentially a wrapper function to connected. This function is mainly used internally by FeatureFinder and similar, but could be of use outside such functions.

## Value

An unnamed list object where each component is an image describing one set of connected components (or blobs).

## Author(s)

Eric Gilleland

## References

Park, J.-M., Looney, C.G. and Chen, H.-C. (2000) Fast connected component labeling algorithm using a divide and conquer technique. Pages 373-376 in S.Y. Shin (ed) *Computers and Their Applications:* Proceedings of the ISCA 15th International Conference on Computers and Their Applications, March 29–31, 2000, New Orleans, Louisiana USA. ISCA 2000, ISBN 1-880843-32-3.

Rosenfeld, A. and Pfalz, J.L. (1966) Sequential operations in digital processing. *Journal of the Association for Computing Machinery* **13** 471–494.

## See Also

connected, FeatureFinder

## Examples

```
##
## For examples, see FeatureFinder
##
```

EBS *Elmore, Baldwin and Schultz Method for Field Significance for Spatial Bias Errors*

## Description

Apply the method of Elmore, Baldwin and Schultz (2006) for calculating field significance of spatial bias errors.

## Usage

```
EBS(object, model = 1, block.length = NULL, alpha.boot = 0.05,
    field.sig = 0.05, bootR = 1000, ntrials = 1000,
    verbose = FALSE)

## S3 method for class 'EBS'
plot(x, ..., mfrow = c(1, 2), col, horizontal)
```

## Arguments

| | |
|---|---|
| object | list object of class "SpatialVx". |
| x | object of class "EBS" as returned by EBS. |
| model | number or character describing which model (if more than one in the "SpatialVx" object) to compare. |
| block.length | numeric giving the block length to be used n the block bootstrap algorithm. If NULL, floor(sqrt(n)) is used. |
| alpha.boot | numeric between 0 and 1 giving the confidence level desired for the bootstrap algorithm. |
| field.sig | numeric between 0 and 1 giving the desired field significance level. |
| bootR | numeric integer giving the number of bootstrap replications to use. |
| ntrials | numeric integer giving the number of Monte Carol iterations to use. |
| mfrow | mfrow parameter (see help file for par). If NULL, then the parameter is not re-set. |
| col, horizontal | optional arguments to image.plot from **fields**. |
| verbose | logical, should progress information be printed to the screen? |
| ... | optional arguments to image.plot from **fields**. |

## Details

this is a wrapper function for the spatbiasFS function utilizing the "SpatialVx" object class to simplify the arguments.

## Value

A list object of class "EBS" with the same attributes as the input object and additional attribute (called "arguments")that is a named vector giving information provided by the user. Components of the list include:

`block.boot.results`
: object of class "LocSig".

`sig.results`
: list object containing information about the significance of the results.

## Author(s)

Eric Gilleland

## References

Elmore, K. L., Baldwin, M. E. and Schultz, D. M. (2006) Field significance revisited: Spatial bias errors in forecasts as applied to the Eta model. *Mon. Wea. Rev.*, **134**, 519–531.

## See Also

boot, tsboot, spatbiasFS, LocSig, poly.image, image.plot, make.SpatialVx

## Examples

```
data( "GFSNAMfcstEx" )
data( "GFSNAMobsEx" )
data( "GFSNAMlocEx" )

id <- GFSNAMlocEx[,"Lon"] >=-95
id <- id & GFSNAMlocEx[,"Lon"] <= -75
id <- id & GFSNAMlocEx[,"Lat"] <= 32

##
## This next step is a bit awkward, but these data
## are not in the format of the SpatialVx class.
## These are being set up with arbitrarily chosen
## dimensions (49 X 48) for the spatial part.  It
## won't matter to the analyses or plots.
##
Vx <- GFSNAMobsEx
Fcst <- GFSNAMfcstEx
Ref <- array(t(Vx), dim=c(49, 48, 361))
Mod <- array(t(Fcst), dim=c(49, 48, 361))

hold <- make.SpatialVx(Ref, Mod, loc=GFSNAMlocEx,
    projection=TRUE, map=TRUE, loc.byrow = TRUE, subset=id,
    field.type="Precipitation", units="mm",
    data.name = "GFS/NAM", obs.name = "Reference", model.name = "Model" )

look <- EBS(hold, bootR=500, ntrials=500, verbose=TRUE)
plot( look )
```

```
## Not run:
# Same as above, but now we'll do it for all points.
# A little slower, but not terribly bad.

hold <- make.SpatialVx(Ref, Mod, loc = GFSNAMlocEx,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", reg.grid = FALSE, units = "mm",
    data.name = "GFS/NAM", obs.name = "Reference", model.name = "Model" )

look <- EBS(hold, bootR=500, ntrials=500, verbose=TRUE)
plot( look )

## End(Not run)
```

---

`ExampleSpatialVxSet` *Simulated Spatial Verification Set*

---

## Description

A simulated spatial verification set for use by various examples for this package.

## Usage

```
data(ExampleSpatialVxSet)
```

## Format

The format is: List of 2 $ vx : num [1:50, 1:50] 0 0 0 0 0 0 0 0 0 0 ... $ fcst: num [1:50, 1:50] 0.0141 0 0 0 0 ...

## Details

The data here were generated using the `sim.rf` function from **fields** (Furrer et al., 2012):

x <- y <- matrix(0, 10, 12) x[2:3,c(3:6, 8:10)] <- 1 y[c(1:2, 9:10),c(3:6)] <- 1

grid <- list(x=seq(0,5,,50), y=seq(0,5,,50)) obj <- Exp.image.cov(grid=grid, theta=0.5, setup=TRUE) x <- sim.rf(obj) x[x < 0] <- 0 x <- zapsmall(x)

y <- sim.rf(obj) y[y < 0] <- 0 y <- zapsmall(y)

## References

Reinhard Furrer, Douglas Nychka and Stephen Sain (2012). fields: Tools for spatial data. R package version 6.6.3. http://CRAN.R-project.org/package=fields

## Examples

```
data( "ExampleSpatialVxSet" )
x <- ExampleSpatialVxSet$vx
xhat <- ExampleSpatialVxSet$fcst
par(mfrow=c(1,2))
image.plot(x, col=c("gray",tim.colors(64)))
image.plot(xhat, col=c("gray",tim.colors(64)))
```

---

# expvg *Exponential Variogram*

---

## Description

Compute the exponential variogram.

## Usage

```
expvg(p, vg, ...)

## S3 method for class 'flossdiff.expvg'
predict(object, newdata, ...)

## S3 method for class 'flossdiff.expvg'
print(x, ...)
```

## Arguments

| | |
|---|---|
| p | numeric vector of length two. Each component should be positively valued. The first component is the nugget and the second is the range parameter. |
| vg | A list object with component d giving a numeric vector of distances over which the variogram is to be calculated. |
| object, x | A list object returned by flossdiff using expvg as the variogram model. |
| newdata | Numeric giving the distances over which to use the fitted exponential variogram model to make predictions. The default is to go from zero to the maximum lag distance for a given data set, which is not the usual convention for the generic predict, which usually defaults to operate on the lags used in performing the fit. |
| ... | Not used. |

## Details

A very simple function used mainly internally by flossdiff when fitting the exponential variogram to the empirical one, and by the predict, print and summary method functions for lossdiff objects. For those wishing to use a different variogram model than the exponential, use this function and its method functions as a template. Be sure to create predict and print method functions to operate on objects of class "flossdiff.XXX" where "XXX" is the name of the variogram function you write (so, "expvg" in the current example).

## Value

Numeric vector of length equal to that of the d component of vg giving the corresponding exponential variogram values with nugget and range defined by p.

## Author(s)

Eric Gilleland

## References

Cressie, N. A. (2015) Statistics for Spatial Data. Wiley-Interscience; Revised Edition edition (July 27, 2015), ISBN-10: 1119114616, ISBN-13: 978-1119114611, 928 pp.

## See Also

lossdiff, flossdiff

## Examples

```
##
## For examples, see lossdiff and flossdiff
##
```

---

# expvgram *Exponential Variogram*

---

## Description

Calculates the empirical variogram for use with function spct.

## Usage

```
expvgram(p, h, ...)
```

## Arguments

| | |
|---|---|
| p | numeric vector of length two giving the nugget and range parameter values, resp. |
| h | numeric vector of separation distances. |
| ... | Not used. |

## Details

Simple function to work with spct to calculate the exponential variogram for given parameters and separation distances. The exponential variogram employed here is parameterized by

gamma(h) = sigma * ( 1 - exp( - h * theta ) )

where p is c( sigma, theta ).

## Value

A numeric vector of variogram values for each separation distance in h.

## Author(s)

Eric Gilleland

## See Also

spct

## Examples

```
# See help file for spct for examples.
```

---

FeatureAxis *Major and Minor Axes of a Feature*

---

## Description

Calculate the major and minor axes of a feature and various other properties such as the aspect ratio.

## Usage

```
FeatureAxis(x, fac = 1, flipit = FALSE, twixt = FALSE)

## S3 method for class 'FeatureAxis'
plot(x, ..., zoom = FALSE)

## S3 method for class 'FeatureAxis'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x | For FeatureAxis this is an object of class "owin" containing a binary image matrix defining the feature. In the case of plot.FeatureAxis, this is the value returned from FeatureAxis. |
| object | list object of class "FeatureAxis" as returned by FeatureAxis. |
| fac | numeric, in determining the lengths of the axes, they are multiplied by a factor of fac (e.g., if the grid points are k by k km each, then one could set this to k so that the resulting lengths are in terms of km rather than grid points. |
| flipit | logical, should the objects be flipped over x and y? The disjointer function results in images that are flipped, this would flip them back. |
| twixt | logical, should the major axis angle be forced to be between +/- 90 degrees? |

| | |
|---|---|
| zoom | logical, should the object be plotted on its bounding box (TRUE) or on the original grid (FALSE, default)? Useful if the feature is too small to be seen well on the original gid. |
| ... | For plot.FeatureAxis these are additional arguments to the plot function. Not used by summary.FeatureAxis. |

## Details

This function attempts to identify the major and minor axes for a pre-defined feature (sometimes referred to as an object). This function relies heavily on the **spatstat** and **smatr** packages. First, the convex hull of the feature is determined using the convexhull function from the **spatstat** package. The major axis is then found using the sma function from package **smatr**, which is then converted into a psp object (see as.psp from **spatstat**) from which the axis angle and length are found (using angles.psp and lengths.psp, resp., from **spatstat**).

The minor axis anlge is easily found after rotating the major axis 90 degrees using rotate.psp from **spatstat**. The length of the minor axis is more difficult. Here, it is found by rotating the convex hull of the feature by the major axis angle (so that it is upright) using rotate.owin from **spatstat**, and then computing the bounding box (using boundingbox from **spatstat**). The differnce is then taken between the range of x- coordinates of the bounding box. This seems to give a reasonable value for the length of the minor axis. A psp object is then created using the mid point of the major axis (which should be close to the centroid of the feature) using as.psp and midpoints.psp from **spatstat** along with the length and angle already found for the minor axis.

See the help files for the above mentioned functions for references, etc.

## Value

FeatureAxis: A list object of class "FeatureAxis" is returned with components:

| | |
|---|---|
| z | same as the argument x passed in. |
| MajorAxis,MinorAxis | a psp object with one segment that is the major (minor) axis. |
| OrientationAngle | list with two components: MajorAxis (the angle in degrees of the major axis wrt the abscissa), MinorAxis (the angle in degrees wrt the abscissa). |
| aspect.ratio | numeric giving the ratio of the length of the minor axis to that of the major axis (always between 0 and 1). |
| MidPoint | an object of class "ppp" giving the mid point of the major (minor) axis. |
| lengths | list object with components: MajorAxis giving the length (possibly multiplied by a factor) of the major axis, and MinorAxis same as MajorAxis but for the minor axis. |
| sma.fit | The fitted object returned by the sma function. This is useful, e.g., if confidence intervals for the axis are desired. See the sma help file for more details. |

No value is returned from the plot or summary method functions.

## Author(s)

Eric Gilleland

## See Also

owin, convexhull, sma, as.psp, angles.psp, rotate.owin, rotate.psp, boundingbox, midpoints.psp, lengths.psp, infline, clip.infline, deltamm, FeatureFinder, disjointer, connected, tiles, tess, solutionset

## Examples

```
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx

look <- disk2dsmooth(x,5)
u <- quantile(look,0.99)
sIx <- matrix(0, 100, 100)
sIx[ look > u] <- 1
look2 <- disjointer(sIx)[[1]]
look2 <- flipxy(look2)
tmp <- FeatureAxis(look2)
plot(tmp)
summary(tmp)

## Not run:
data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004,
    loc = ICPg240Locs, projection = TRUE, map = TRUE,
    loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Perturbed ICP Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- FeatureFinder(hold, smoothpar=10.5)
par(mfrow=c(1,2))
plot(look)

par(mfrow=c(2,2))
image.plot(look$X.labeled)
image.plot(look$Y.labeled)

# The next line will likely be very slow.
look2 <- deltamm(x=look, verbose=TRUE)
image.plot(look2$X.labeled)
image.plot(look2$Y.labeled)

look2$mm.new.labels # the first seven features are matched.

ang1 <- FeatureAxis(look2$X.feats[[1]])
ang2 <- FeatureAxis(look2$Y.feats[[1]])
plot(ang1)
plot(ang2)
```

```
summary(ang1)
summary(ang2)

ang3 <- FeatureAxis(look2$X.feats[[4]])
ang4 <- FeatureAxis(look2$Y.feats[[4]])
plot(ang3)
plot(ang4)
summary(ang3)
summary(ang4)

## End(Not run)
```

---

FeatureFinder *Threshold-based Feature Finder*

---

## Description

Identify spatial features within a verification set using a threshold-based method.

## Usage

```
FeatureFinder(object, smoothfun = "disk2dsmooth", do.smooth = TRUE,
    smoothpar = 1, smoothfunargs = NULL, thresh = 1e-08, idfun = "disjointer",
    min.size = 1, max.size = Inf, fac = 1, zero.down = FALSE, time.point = 1,
    obs = 1, model = 1, ...)

## S3 method for class 'features'
plot(x, ..., type = c("both", "obs", "model"))

## S3 method for class 'features'
print(x, ...)

## S3 method for class 'features'
summary(object, ...)

## S3 method for class 'summary.features'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| object | An object of class "SpatialVx". |
| x | list object of class "features" as returned by FeatureFinder. |
| smoothfun | character naming a 2-d smoothing function from package **smoothie**. Not used if do.smooth is FALSE. |

| | |
|---|---|
| do.smooth | logical, should the field first be smoothed before trying to identify features (resulting field will not be smoothed, this is just for identifying features). Default is to do convolution smoothing using a disc kernel as is recommended by Davis et al (2006a). |
| smoothpar | numeric of length one or two giving the smoothing parameter for smoothfun. If length is two, the first value is applied to the forecast field and the second to the verification field. The default smooth function (smoothfun argument) is the disk kernel smoother, so this argument gives the radius of the disk. See the help file for hoods2dsmooth from package **smoothie** for other smoother choices; this argument corresponds to the lambda argument of these functions. |
| smoothfunargs | list object with named additional arguments to smoothfun. |
| thresh | numeric vector of length one or two giving the threshold over which (inclusive) features should be identified. If different thresholds are used for the forecast and verification fields, then the first element is the threshold for the forecast, and the second for the verification field. |
| idfun | character naming the function used to identify (and label) individual features in the thresholded, and possibly smoothed, fields. Must take an argument 'x', the thresholded, and possibly smoothed, field. |
| min.size | numeric of length one or two giving the minimum number of contiguous grid points exceeding the threshold in order to be included as a feature (can be used to exclude any small features). Default does not exclude any features. If length is two, first value applies to the forecast and second to the verification field. |
| max.size | numeric of length one or two giving the maximum number of contiguous grid points exceeding the threshold in order to be included as a feature (can be used to exclude large features, if the need be). Default does not exclude any features. If length is two, then the first value applies ot the forecast field, and the second to the verification. |
| fac | numeric of length one or two giving a factor by which to multiply the R quantile in determining the threshold from the fields. For example, ~ 1/15 is suggested in Wernli et al (2008, 2009). If length is two, then the first value applies to the threshold of the forecast and the second to that of the verification field. |
| zero.down | logical, should negative values and relatively very small values be set to zero after smoothing the fields? For thresholds larger than such values, this argument is moot. 'zapsmall' is used to set the very small positive values to zero. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| type | character string stating which features to plot (observed, forecast or both). If both, a panel of two plots will be made side-by-side. |
| ... | FeatureFinder: additional arguments to idfun.<br>plot: optional arguments to image.plot for the color legend bar only.<br>Not used by the print or summary functions.<br>The 'summary' method function can take the argument: 'silent'-logical, should information be printed to the screen (FALSE) or not (TRUE). |

## Details

FeatureFinder applies for finding features based on three proposed methods from different papers; and also allows for combinations of the methods. The methods include: the convolution-threshold approach of Davis et al. (2006a,b), which uses a disc kernel convolution smoother to first smooth the fields, then applies a threshold to remove low-intensity areas. Feautres are identified by groups of contiguous "events" (or connected components in the computer vision/image analysis literature) using idfun. Nachamkin (2009) and Lack et al (2010) further require that features have at least min.size connected components in order to be considered a feature (in order to remove very small areas of threshold excesses). Wernli et al. (2009) modify the threshold by a factor (see the fac argument).

In addition to the above options, it is also possible to remove features that are too large, as for some purposes, it is the small-scale features that are of interest, and sometimes the larger features can cause problems when merging and matching features across fields.

## Value

FeatureFinder returns a list object of class "features" with comopnents:

data.name — character vector naming the verification and forecast (R object) fields, resp.

X.feats,Y.feats — The identified features for the verification and forecast fields as returned by the idfun function.

X.labeled,Y.labeled — matrices of same dimension as the forecast and verification fields giving the images of the convolved and thresholded verification and forecast fields, but with each individually identified object labeled 1 to the number of objects in each field.

identifier.function,identifier.label — character strings naming the function and giving the long name (for use with plot method function).

An additional attribute, named "call", is given. This attribute shows the original function call, and is used mainly by the print function..

The plot method functions do not return anything.

The summary method function for objects of class "features" returns a list with components:

X,Y — matrices whose rows are objects and columns are properties: centroidX and centroidY (the x- and y- coordinates for the feature centroids), area (the area of each feature in squared grid points), the orientation angle for the fitted major axis, the aspect ratio, Intensity0.25 and Intensity0.9 (the lower quartile and 0.9 quantile of intensity values for each feature).

## Note

This function replaces the now deprecated functions: convthresh, threshsizer and threshfac.

## Author(s)

Eric Gilleland

## References

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006a) Object-based verification of precipitation forecasts, Part I: Methodology and application to mesoscale rain areas. _Mon. Wea. Rev._, *134*, 1772-1784.

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006b) Object-based verification of precipitation forecasts, Part II: Application to convective rain systems. _Mon. Wea. Rev._, *134*, 1785-1795.

Lack, S. A., Limpert, G. L. and Fox, N. I. (2010) An object-oriented multiscale verification scheme. _Wea. Forecasting_, *25*, 79-92, doi:10.1175/2009WAF2222245.1.

Nachamkin, J. E. (2009) Application of the composite method to the spatial forecast verification methods intercomparison dataset. _Wea. Forecasting_, *24*, 1390-1400, doi:10.1175/2009WAF2222225.1.

Wernli, H., Paulat, M. Hagen, M. and Frei, C. (2008) SAL-A novel quality measure for the verification of quantitative precipitation forecasts. _Mon. Wea. Rev._, *136*, 4470-4487.

Wernli, H., Hofmann, C. and Zimmer, M. (2009) Spatial forecast verification methods intercomparison project: Application of the SAL technique. _Wea. Forecasting_, *24*, 1472-1484, doi:10.1175/2009WAF2222271.1.

## See Also

Functions used in identifying the features (mostly from package **spatstat**:

`connected`, `as.im`, `tess`, `tiles`, `owin`, `make.SpatialVx`, `disjointer`

Functions that work on the resulting "features" objects for merging and/or matching features within/across fields:

`centmatch`, `deltamm`, `minboundmatch`

To force merges (implicit or otherwise; recommended): `MergeForce`

## Examples

```
## Not run:
x <- y <- matrix(0, 100, 100)
x[2:3,c(3:6, 8:10)] <- 1
y[c(4:7, 9:10),c(7:9, 11:12)] <- 1

x[30:50,45:65] <- 1
y[c(22:24, 99:100),c(50:52, 99:100)] <- 1

hold <- make.SpatialVx( x, y, field.type = "contrived", units = "none",
         data.name = "Example", obs.name = "x", model.name = "y" )

look <- FeatureFinder(hold, smoothpar=0.5)

par( mfrow=c(1,2))
image.plot(look$X.labeled)
image.plot(look$Y.labeled)

look2 <- centmatch(look)

FeatureTable(look2)
```

```
look3 <- deltamm( look, N = 201, verbose = TRUE )
FeatureTable( look3 )


data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004,
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- FeatureFinder(hold, smoothpar=10.5, thresh = 5)
plot(look)

look2 <- deltamm( look, N = 701, verbose = TRUE )

look2 <- MergeForce( look2 )

plot(look2)

summary( look2 )

# Now remove smallest features ( those with fewer than 700 grid squares).

look <- FeatureFinder( hold, smoothpar = 10.5, thresh = 5, min.size = 700 )

look # Now only two features.

plot( look )

# Now remove the largest features (those with more than 1000 grid squares).

look <- FeatureFinder( hold, smoothpar = 10.5, thresh = 5, max.size = 1000 )

look

plot( look )

# Remove any features smaller than 700 and larger than 2000 grid squares).

look <- FeatureFinder( hold, smoothpar = 10.5, thresh = 5,
    min.size = 700, max.size = 2000 )

look

plot( look )

# Find features according to Wernli et al. (2008).
look <- FeatureFinder( hold, thresh = 5, do.smooth = FALSE, fac = 1 / 15 )
```

```
look

plot( look )

# Now do a mix of the two types of methods.
look <- FeatureFinder( hold, smoothpar = 10.5, thresh = 5, fac = 1 / 15 )

look

plot( look )


## End(Not run)
```

---

`FeatureMatchAnalyzer` *Analyze Features of a Verification Set*

---

## Description

Analyze matched features of a verification set.

## Usage

```
FeatureMatchAnalyzer(x, which.comps=c("cent.dist", "angle.diff", "area.ratio", "int.area",
                     "bdelta", "haus", "ph", "med", "msd", "fom", "minsep",
    "bearing"), sizefac=1, alpha=0.1, k=4, p=2, c=Inf,
    distfun="distmapfun", ...)

## S3 method for class 'matched.centmatch'
FeatureMatchAnalyzer(x, which.comps=c("cent.dist", "angle.diff",
    "area.ratio", "int.area", "bdelta", "haus", "ph", "med",
    "msd", "fom", "minsep", "bearing"), sizefac=1, alpha=0.1, k=4, p=2,
    c=Inf, distfun="distmapfun", ...)

## S3 method for class 'matched.deltamm'
FeatureMatchAnalyzer(x, which.comps = c("cent.dist", "angle.diff",
    "area.ratio", "int.area", "bdelta", "haus", "ph", "med", "msd",
    "fom", "minsep", "bearing"), sizefac = 1, alpha = 0.1, k = 4, p = 2,
    c = Inf, distfun = "distmapfun", ..., y = NULL, matches = NULL,
    object = NULL)

## S3 method for class 'FeatureMatchAnalyzer'
summary(object, ...)

## S3 method for class 'FeatureMatchAnalyzer'
plot(x, ..., type = c("all", "ph", "med", "msd",
    "fom", "minsep", "cent.dist", "angle.diff", "area.ratio",
```

```
    "int.area", "bearing", "bdelta", "haus"))

## S3 method for class 'FeatureMatchAnalyzer'
print(x, ...)

FeatureComps(Y, X, which.comps=c("cent.dist", "angle.diff", "area.ratio", "int.area",
    "bdelta", "haus", "ph", "med", "msd", "fom", "minsep", "bearing"),
    sizefac=1, alpha=0.1, k=4, p=2, c=Inf, distfun="distmapfun", deg = TRUE,
    aty = "compass", loc = NULL, ...)

## S3 method for class 'FeatureComps'
distill(x, ...)
```

## Arguments

| | |
|---|---|
| x,y,matches | x, y and matches are list objects with components as output by deltamm or similar function. Only one is used, and it first checks for matches, then y, and finally x. It expects a component named mm.new.labels that gives the number of matched objects. In the case of the plot and print method functions, x is a list object as returned by FeatureMatchAnalyzer.<br>distill: output from FeatureComps. |
| X,Y | list object giving a pixel image as output from solutionset from package **spatstat** for the verification and forecast fields, resp. These arguments are passed directly to the locperf function. |
| object | list object returned of class "FeatureMatchAnalyzer", this is the returned value from the self-same function. |
| which.comps,type | character vector indicating which properties of the features are to be analyzed (which.comps) or plotted (type). |
| sizefac | single numeric by which area calculations should be multiplied in order to get the desired units. If unity (default) results are in terms of grid squares. |
| alpha | numeric value for the FOM measure (see the help file for locperf. |
| k | numeric indicating which quantile to use if the partial Hausdorff measure is to be used. |
| p | numeric giving the value of the parameter p for the Baddeley metric. |
| c | numeric giving the cut-off value for the Baddeley metric. |
| distfun | character naming a distance functions to use in calculating the various binary image measures. Default is Euclidean distance. |
| deg, aty | optional arguments to the bearing function. |
| loc | two-column matrix giving location coordinates for centroid distance. If NULL, uses an indices based on the dimension of the field. |
| ... | Additional arguments to deltametric from package **spatstat**. In case of the summary method function, additional optional arguments may be passed, which include silent (logical, should the information be printed to the screen or not?), interest (numeric vector defining an interest value for calculating total interest |

for each matched object, if NULL, this is not performed), con (name of function that takes three arguments, the first two are matrices whose rows are objects and columns are matched feature properties, where the former is a matrix of matched feature property values (e.g., angle difference) and the latter is a matrix of interest values determined by the interest argument (whereby each row is identical), the third argument to con must be called which.comps, and it gives the short-form feature property names (i.e., same as which.comps argument); see details section). In the case of the plot method function, these are optional arguments to the function barplot.

Not used by distill.

## Details

FeatureMatchAnalyzer operates on objects of class "matched". It is set up to calculate the values discussed in sec. 4 of Davis et al. (2006) for a single verification set (i.e., mean and standard deviation are not computed because it is only a single case). If criteria is 1, then features separated by a distance D < the sum of the sizes of the two features (size of a feature is defined as the square root of its area) are considered a match. If criteria is 2, then a match is made if D < the average of the sizes of the two features. Finally, criteria 3 decides a match as being anything less than a pre-determined constant.

FeatureComps is the primary function called by FeatureMatchAnalyzer, and is designed as a more stand-alone type of function. Several of the measures that can be calculated are simply the binary image measures/metrics available via, e.g., locperf. It calculates comparisons between two matched features (i.e., between the verification and forecast fields).

distill reduces a "FeatureComps" list object to a named numeric vector containing (in this order) the components that exist from "cent.dist", "angle.diff", "area.ratio", "int.area", "bdelta", "haus", "ph", "med", "msd", "fom", and "minsep". This is used, for example, by interester, which is why the order is important.

The summary method function for FeatureMatchAnalyzer allows for passing a function, con, to determine confidence for each interest value. The idea being to set the interest to zero when the particular interest value does not make sense. For example, angle difference makes no sense if both objects are circles. Currently, no functions are included in this package for actually doing this, and so the functionality itself has not been tested.

The print method function for FeatureMatchAnalyzer first converts the object to a simple named matrix, then prints the matrix out. The resulting matrix is returned invisibly.

## Value

FeatureMatchAnalyzer returns a list of list objects. The specific components depend on the 'which.comps' argument, and are the same as those returned by FeatureComps. These can be any of the following.

| | |
|---|---|
| cent.dist | numeric giving the centroid (Euclidean) distance. |
| angle.diff | numeric giving the orientation (major axis) angle difference. |
| area.ratio | numeric giving the area ratio, which is always between 0 and 1 because this is defined by Davis et al. (2006) to be the area of the smaller feature divided by that of the larger feature regardless of which field the feature belongs to. |
| int.area | numeric giving the intersection area of the features. |

`bdelta` numeric giving Baddeley's delta metric between the two features.

`haus, ph, med, msd, fom, minsep` numeric, see locperf for specific information.

`bearing` numeric giving the bearing from the forecast object centroid to the observed object centroid.

The summary method for FeatureMatchAnalyzer invisibly returns a matrix with the same information, but where each matched object is a row and each column is the specific statistic. Or, if optional interest argument is passed, a list with components:

`print` returns a named vector invisibly.

## Author(s)

Eric Gilleland

## References

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006) Object-based verification of precipitation forecasts, Part I: Methodology and application to mesoscale rain areas. *Mon. Wea. Rev.*, **134**, 1772–1784.

## See Also

Functions to identify features: `FeatureFinder`

Functions to merge and/or match objects: `deltamm`, `centmatch`, `MergeForce`

Functions to compute feature properties: `locperf`, `deltametric`, `bearing`

Function to calculate fuzzy logic interest values: `interester`

## Examples

```
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx
xhat <- ExampleSpatialVxSet$fcst

hold <- make.SpatialVx( x, xhat, field.type="Example",
    units = "units", data.name = "Example",
    obs.name = "x", model.name = "xhat" )

look <- FeatureFinder(hold, smoothpar=1.5)
look2 <- centmatch(look)

tmp <- FeatureMatchAnalyzer(look2)
tmp
summary(tmp)
plot(tmp)


## Not run:
data( "pert000" )
```

```
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Perturbed ICP Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- FeatureFinder(hold, smoothpar=10.5)
look2 <- centmatch(look)
tmp <- FeatureMatchAnalyzer(look2)
summary(tmp)
plot(tmp)

## End(Not run)
```

---

FeatureProps *Single Feature Properties*

---

## Description

Calculate properties for an identified feature.

## Usage

```
FeatureProps(x, Im = NULL, which.props = c("centroid", "area", "axis", "intensity"),
    areafac = 1, q = c(0.25, 0.9), loc = NULL, ...)
```

## Arguments

| | |
|---|---|
| x | object of class "owin" containing a binary image matrix defining the feature. |
| Im | Matrix giving the original values of the field from which the feature was extracted. Only needed if the feature intensity is desired. |
| which.props | character vector giving one or more of "centroid", "area", "axis" and "intensity". If "axis" is given, then a call to FeatureAxis is made. |
| areafac | numeric, in determining the lengths of the axes, they are multiplied by a factor of fac (e.g., if the grid points are k by k km each, then one could set this to k so that the resulting lengths are in terms of km rather than grid points. |
| q | numeric vector of values between 0 and 1 inclusive giving the quantiles for determining the intensity of the feature. |
| loc | optional argument giving a two-column matrix of grid locations for finding the centroid. If NULL, indices based on the dimension of x are used. |
| ... | additional arguments to FeatureAxis. |

## Details

This function takes an owin image and returns several property values for that image, including: centroid, spatial area, major and minor axis angle/length, as well as the overall intensity of the field (cf., Davis et al., 2006a, b).

## Value

list object with components depending on the which.props argument. One or more of:

| | |
|---|---|
| centroid | list with components x and y giving the centroid of the object. |
| area | numeric giving the area of the feature. |
| axis | list object of class FeatureAxis as returned by the same-named function. |

## Author(s)

Eric Gilleland

## References

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006a) Object-based verification of precipitation forecasts, Part I: Methodology and application to mesoscale rain areas. *Mon. Wea. Rev.*, **134**, 1772–1784.

Davis, C. A., Brown, B. G. and Bullock, R. G. (2006b) Object-based verification of precipitation forecasts, Part II: Application to convective rain systems. *Mon. Wea. Rev.*, **134**, 1785–1795.

## See Also

FeatureAxis, owin, convexhull, sma, as.psp, angles.psp, rotate.owin, rotate.psp, boundingbox, midpoints.psp, lengths.psp, infline, clip.infline, deltamm, FeatureFinder, disjointer, connected, tiles, tess, solutionset

## Examples

```
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx

look <- disk2dsmooth(x,5)
u <- quantile(look,0.99)
sIx <- matrix(0, 100, 100)
sIx[ look > u] <- 1
look2 <- disjointer(sIx)[[1]]
look2 <- flipxy(look2)

FeatureProps(look2,
    which.props=c("centroid", "area", "axis"))
```

FeatureTable *Feature-Based Contingency Table*

## Description

Create a feature-based contingency table from a matched object and calculate some summary scores with their standard errors.

## Usage

```
FeatureTable(x, fudge = 1e-08, hits.random = NULL, correct.negatives = NULL, fA = 0.05)

## S3 method for class 'FeatureTable'
ci(x, alpha = 0.05, ...)

## S3 method for class 'FeatureTable'
print(x, ...)

## S3 method for class 'FeatureTable'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x | FeatureTable: An object of class "matched" (e.g., from deltamm or centmatch.<br>print: An object of class "FeatureTable". |
| object | An object of class "FeatureTable". |
| fudge | value added to denominators of scores to ensure no division by zero. Set to zero if this practice is not desired. |
| hits.random | If a different value for random hits in the caluclation of GSS than provided is desired, it can be given here. Default uses Eq (3) from Davis et al (2009). |
| correct.negatives | If a different value for correct negatives than provided is desired, it can be given here. Default uses Eq (4) from Davis et al (2009). |
| fA | numeric between zero and 1 giving the fraction of area occupied for the purpose of matching as in Davis et al (2009). |
| alpha | numeric between zero and one giving the (1 - alpha) * 100 percent confidence level. |
| ... | Additional arguments to ci. |

## Details

This function takes an object of class "matched" and calculates a contingency table based on matched and unmatched objects. If no value for correct negatives is given, then it will also determine them based on Eq (3) from Davis et al (2009). The following contingency table scores

and their standard errors (based on their usual traditional version) are returned. It should be noted that the standard errors may not be entirely meaningful because they do not capture the uncertainty associated with identiying, merging and matching features within the fields. Neverhteless, they are calculated here for investigative purposes. Note that hits are determined by number of matched objects, which for some matching algorithms can mean that features are matched more than once (e.g., if using `centmatch`). In essence, this fact may artificially increase thenumber of hits. On the other hand, situations exist where such handling may be more appropriate than not having duplicate matches.

hits are determined by the total number of matched features.

false alarms are the total number of unmatched forecast features.

misses are the total number of unmatched observed features.

correct negatives are less obviously defined. If the user does not supply a value, then these are calculated according to Eq (4) in Davis et al (2009).

GSS: Gilbert skill score (aka Equitable Threat Score) based on Eq (2) of Davis et al (2009).

POD: probability of detecting an event (aka the hit rate).

false alarm rate: (aka probability of false detection) is the ratio of false alarms to the number of false alarms and correct negtives.

FAR: the false alarm ratio is the ratio of false alarms to the total forecast events (in this case, the total number of forecast features in the field).

HSS: Heidke skill score

The `print` method function simply calls `summary`, which prints the feature-based contingency table in addition to calling `ci`. The confidence intervals are based on the normal approximation method using the estimated standard errors, which themselves are suspicious. In any case, the intervals can give a feel for some of the uncertainty associated with the scores, but should not be considered as solid.

## Value

A list with inherited attributes from x and components:

| | |
|---|---|
| `estimates` | named numeric vector giving the estimated scores. |
| `se` | named numeric vector giving the estimated standard errors of the scores. |
| `feature.contingency.table` | named numeric vector giving the feature-based contingency table. |

## Note

Standard error estimates are based on the univariate equivalent formulations, which do not account for uncertainties introduced in the feature identification, merging/clustering and matching. They should not be considered as legitimate, and resulting confidence intervals should be mistrusted.

## Author(s)

Eric Gilleland

## References

Davis, C. A., Brown, B. G., Bullock, R. G. and Halley Gotway, J. (2009) The Method for Object-based Diagnostic Evaluation (MODE) applied to numerical forecasts from the 2005 NSSL/SPC Spring Program. *Wea. Forecsting*, **24**, 1252–1267, DOI: 10.1175/2009WAF2222241.1.

## See Also

To identify features in the fields: `FeatureFinder`

To match (and merge) features: `centmatch`, `deltamm`

## Examples

```
##
## See help file for 'deltamm' for examples.
##
```

---

# Fint2d *2-d Interpolation*

---

## Description

Interpolate a function of two variables by rounding (i.e. taking the nearest value), bilinear or bicubic interpolation.

## Usage

```
Fint2d(X, Ws, s, method = c("round", "bilinear", "bicubic"), derivs = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `X` | A numeric n by m matrix giving the value of the function at the old coordinates. |
| `Ws` | A numeric k by 2 matrix of new grid coordinates where k <= m * n. |
| `s` | A numeric k by 2 matrix of old grid coordinates where k <= m * n. |
| `method` | character naming one of “round” (default), “bilinear”, or “bicubic” giving the specific interpolation method to use. |
| `derivs` | logical, should the gradient interpolatants be returned? |
| `...` | Not used. |

## Details

Method round simply returns the values at each grid point that correspond to the nearest points in the old grid.

Interpolation of a function, say H, is achieved by the following formula (cf. Gilleland et al 2010, sec. 3), where r and s represent the fractional part of their respective coordinate. that is, r = x - g( x ) and s = y - g( y ), where g( x ) is the greatest integer less than x.

sum_k sum_l b_k( r ) * b_l( s ) * H(g( x ) + l, g( y ) + k).

The specific choices for the values of b_l and b_k and their ranges depends on the type of interpolation. For bilinear interpolation, they both range from 0 to 1, and are given by: b_0( x ) = 1 - x and b_1( x ) = x. for bicubic interpolation, they both range from -1 to 2 and are given by:

b_-1( t ) = (2 * t^2 - t^3 - t) / 2

b_0( t ) = (3 * t^3 - 5 * t^2 + 2) / 2

b_1( t ) = (4 * t^2 - 3 * t^3 + t) / 2

b_2( t ) = ((t - 1) * t^2) / 2.

## Value

If deriv is FALSE, then a matrix is returned whose values correspond to the new coordinates. Otherwise a list is returned with components:

| | |
|---|---|
| `xy` | matrix whose values correspond to the new coordinates. |
| `dx, dy` | matrices giving the x and y direction gradients of the interpolation. |

## Author(s)

Eric Gilleland

## References

Gilleland and co-authors (2010) Spatial forecast verification: Image warping. *NCAR Technical Note*, NCAR/TN-482+STR, DOI: 10.5065/D62805JJ.

## See Also

`rigider`, `rigidTransform`

## Examples

```
# see rigider for an example.
```

FQI *Forecast Quality Index*

## Description

Functions for calculating the Forecast Quality Index (FQI) and its components.

## Usage

```
FQI(object, surr = NULL, k = 4, time.point = 1, obs = 1, model = 1, ...)

UIQI(X, Xhat, ...)

ampstats(X, Xhat, only.nonzero = FALSE)

## S3 method for class 'fqi'
print(x, ...)

## S3 method for class 'fqi'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| object | list object of class "SpatialVx". In the case of the summary method, object is the list object returned by FQI. |
| X,Xhat | numeric matrices giving the fields for the verification set. |
| x | list object of class "fqi" as returned by FQI. |
| surr | three-dimesnional array containing surrogate fields for X, e.g. as returned by surrogater2d. If NULL, these will be calculated using surrogater2d. |
| only.nonzero | logical, should the means and variances of only the non-zero values of the fields be calculated (if so, the covariance is returned as NA)? |
| k | numeric vector for use with the partial Hausdorff distance. For k that are whole numerics or integers >= 1, then the k-th highest value is returned by locmeasures2d. If 0 <= k < 1, then the corresponding quantile is returned. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which forecast model to select for the analysis. |
| ... | In the case of FQI, additional arguments to surrogater2d. Only used if surr is NULL. In the case of UIQI, additional arguments to ampstats. In the case of summary.fqi, these are not used. |

## Details

The FQI was proposed as a spatial verification metric (a true metric in the mathematical sense) by Venugopal et al. (2005) to combine amplitude and displacement error information in a single summary statistic. It is given by

FQI = (PHD_k(X, Xhat)/mean( PHD_k(X, surr_i); i in 1 to number of surrogates)) / (brightness * distortion)

where the numerator is a normalized partial Hausdorff distance (see help file for locperf), brightness (also called bias) is given by 2*(mu1*mu2)/(mu1^2+mu2^2), where mu1 (mu2) is the mean value of X (Xhat), and the distortion term is given by 2*(sig1*sig2)/(sig1^2+sig2^2), where sig1^2 (sig2^2) is the variance of X (Xhat) values. The denominator is a modified UIQI (Universal Image Quality Index; Wang and Bovik, 2002), which itself is given by

UIQI = cor(X,Xhat)*brightness*distortion.

Note that if `only.nonzero` is `TRUE` in the call to `UIQI`, then the modified UIQI used in the FQI formulation is returned (i.e., without multiplying by the correlation term).

The `print` method so far just calls the `summary` method.

## Value

FQI returns a list with with the following components:

| | |
|---|---|
| `phd.norm` | matrix of normalized partial Hausdorff distances for each value of k (rows) and each threshold (columns). |
| `uiqi.norm` | numeric vector of modified UIQI values for each threshold. |
| `fqi` | matrix of FQI values for each value of k (rows) and each threshold (columns). |

It will also have the same attributes as the "SpatialVx" object with additional attributes defining the arguments specific to parameters used by the function.

UIQI returns a list with components:

| | |
|---|---|
| `data.name` | character vector giving the names of the two fields. |
| `cor` | single numeric giving the correlation between the two fields. |
| `brightness.bias` | single numeric giving the brightness (bias) value. |
| `distortion.variability` | single numeric giving the distortion (variability) value. |
| `UIQI` | single numeric giving the UIQI (or modified UIQI if only.nonzero is set to TRUE) value. |

ampstats returns a list object with components:

| | |
|---|---|
| `mean.fcst,mean.vx` | single numerics giving the mean of Xhat and X, resp. |
| `var.fcst,var.vx` | single numerics giving the variance of Xhat and X, resp. |
| `cov` | single numeric giving the covariance between Xhat and X (if only.nonzero is TRUE, this will be NA). |

## Author(s)

Eric Gilleland

## References

Venugopal, V., Basu, S. and Foufoula-Georgiou, E. (2005) A new metric for comparing precipitation patterns with an application to ensemble forecasts. *J. Geophys. Res.*, **110**, D08111, 11 pp., doi:10.1029/2004JD005395.

Wang, Z. and Bovik, A. C. (2002) A universal image quality index. *IEEE Signal Process. Lett.*, **9**, 81–84.

## See Also

locperf, surrogater2d, locmeasures2d

## Examples

```
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx
xhat <- ExampleSpatialVxSet$fcst

# Now, find surrogates of the simulated field.
z <- surrogater2d(x, zero.down=TRUE, n=10)

u <- list( X = cbind( quantile( c(x), c(0.75, 0.9)) ),
           Xhat = cbind( quantile( c(xhat), c(0.75, 0.9) ) ) )

hold <- make.SpatialVx(x, xhat, thresholds = u,
    field.type = "Example", units = "none",
    data.name = "ExampleSpatialVxSet",
    obs.name = "X", model.name = "Xhat" )

FQI(hold, surr = z, k = c(4, 0.75) )
```

---

# fss2dfun *Various Verification Statistics on Possibly Neighborhood-Smoothed Fields.*

---

## Description

Functions to calculate various verification statistics on possibly neighborhood smoothed fields. Used by hoods2d, but can be called on their own.

## Usage

```
fss2dfun(sPy, sPx, subset = NULL, verbose = FALSE)

fuzzyjoint2dfun(sPy, sPx, subset = NULL)

MinCvg2dfun(sIy, sIx, subset = NULL)

multicon2dfun(sIy, Ix, subset = NULL)

pragmatic2dfun(sPy, Ix, mIx = NULL, subset = NULL)

upscale2dfun(sYy, sYx, threshold = NULL, which.stats = c("rmse",
                "bias", "ts", "ets"), rule = ">=", subset = NULL)
```

## Arguments

| | |
|---|---|
| sPy | n by m matrix giving a smoothed binary forecast field. |
| sPx | n by m matrix giving a smoothed binary observed field. |
| sIy | n by m matrix giving a binary forecast field. |
| sIx | n by m matrix giving a binary observed field (the s indicates that the binary field is obtained from a smoothed field). |
| Ix | n by m matrix giving a binary observed field. |
| mIx | (optional) single numeric giving the base rate. If NULL, this will be calculated by the function. Simply a computation saving step if this has already been calculated. |
| sYy | n by m matrix giving a smoothed forecast field. |
| sYx | n by m matrix giving a smoothed observed field. |
| threshold | (optional) numeric vector of length 2 giving the threshold over which to calculate the verification statistics: bias, ts and ets. If NULL, only the rmse will be calculated. |
| which.stats | character vector naming which statistic(s) should be caluclated for upscale2dfun. |
| subset | (optional) numeric indicating over which points the summary scores should be calculated. If NULL, all of the points are used. |
| rule | character string giving the sort of thresholding process desired. See the help file for thresholder for more information. |
| verbose | logical, should progress information be printed to the screen? |

## Details

These are modular functions that calculate the neighborhood smoothing method statistics in spatial forecast verification (see, e.g., Ebert, 2008, 2009; Gilleland et al., 2009, 2010; Roberts and Lean,2008). These functions take fields that have already had the neighborhood smoothing applied (e.g., using kernele2d) when appropriate. They are called by hoods2d, so need not be called by the user, but they can be.

## Value

In the case of `fss2dfun`, a single numeric giving the FSS value is returned. In the other cases, list objects are returned with one or more of the following components, depending on the particular function.

| | |
|---|---|
| `fuzzy` | `fuzzyjoint2dfun` returns a list with this list as one component. The list component fuzzy has the components: pod, far and ets. |
| `joint` | `fuzzyjoint2dfun` returns a list with this list as one component. The list component joint has the components: pod, far and ets. |
| `pod` | numeric giving the probability of detection, or hit rate. |
| `far` | numeric giving the false alarm ratio. |
| `ets` | numeric giving the equitable threat score, or Gilbert Skill Score. |
| `f` | numeric giving the false alarm rate. |
| `hk` | numeric giving the Hanssen-Kuipers statistic. |
| `bs` | Brier Score |
| `bss` | Brier Skill Score. The `pragmatic2dfun` returns the bs and bss values. The Brier Skill Score here uses the mean square error between the base rate and the Ix field as the reference forecast. |
| `ts` | numeric giving the threat score. |
| `bias` | numeric giving the frequency bias. |

## Author(s)

Eric Gilleland

## References

Ebert, E. E. (2008) Fuzzy verification of high resolution gridded forecasts: A review and proposed framework. *Meteorol. Appl.*, **15**, 51–64. doi:10.1002/met.25

Ebert, E. E. (2009) Neighborhood verification: A strategy for rewarding close forecasts. *Wea. Forecasting*, **24**, 1498–1510, doi:10.1175/2009WAF2222251.1.

Gilleland, E., Ahijevych, D., Brown, B. G., Casati, B. and Ebert, E. E. (2009) Intercomparison of Spatial Forecast Verification Methods. *Wea. Forecasting*, **24**, 1416–1430, doi:10.1175/2009WAF2222269.1.

Gilleland, E., Ahijevych, D. A., Brown, B. G. and Ebert, E. E. (2010) Verifying Forecasts Spatially. *Bull. Amer. Meteor. Soc.*, October, 1365–1373.

Roberts, N. M. and Lean, H. W. (2008) Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Mon. Wea. Rev.*, **136**, 78–97. doi:10.1175/2007MWR2123.1.

## See Also

`hoods2d`,`kernel2dsmooth`,`vxstats`, `thresholder`

## Examples

```
x <- y <- matrix( 0, 100, 100)
x[ sample(1:100, 10), sample(1:100, 10)] <- 1
y[ sample(1:100, 20), sample(1:100, 20)] <- 1
Px <- kernel2dsmooth( x, kernel.type="boxcar", n=9, xdim=c(100, 100))
Py <- kernel2dsmooth( y, kernel.type="boxcar", n=9, xdim=c(100, 100))
par( mfrow=c(2,2))
image( x, col=c("grey", "darkblue"), main="Simulated Observed Events")
image( y, col=c("grey", "darkblue"), main="Simulated Forecast Events")
image( Px, col=c("grey", tim.colors(256)), main="Forecast Event Frequencies (9 nearest neighbors)")
image( Py, col=c("grey", tim.colors(256)), main="Smoothed Observed Events (9 nearest neighbors)")
fss2dfun( Py, Px)

## Not run:
data( "pert004" )
data( "pert000" )

fbin <- obin <- matrix(0, 601, 501)

fbin[ pert004 >= 12] <- 1
obin[ pert000 >= 12] <- 1

Pf <- kernel2dsmooth( fbin, kernel.type="boxcar", n=33, xdim=c(601, 501))
Po <- kernel2dsmooth( obin, kernel.type="boxcar", n=33, xdim=c(601, 501))

fss2dfun(Pf, Po)

fuzzyjoint2dfun(Pf, Po)

Pe <- 1/(33^2) # At least one event in the neighborhood.

MinCvg2dfun(Pf >= Pe, Po >= Pe)

multicon2dfun(Pf >= Pe, obin)

pragmatic2dfun(Pf, obin, mIx=mean( obin, na.rm=TRUE))

Sf <- kernel2dsmooth( pert004, kernel.type="boxcar", n=33, xdim=c(601, 501))
So <- kernel2dsmooth( pert000, kernel.type="boxcar", n=33, xdim=c(601, 501))

upscale2dfun( Sf, So, threshold=12)

## End(Not run)
```

---

`fss2dPlot` *Create Several Graphics for List Objects Returned from hoods2d*

---

## Description

Creates several graphics for list objects returned from hoods2d. Mostly quilt and matrix plots for displaying results of smoothing fields over different neighborhood lengths and thresholds.

## Usage

```
fss2dPlot(x, ..., matplotcol = 1:6, mfrow = c(1, 2), add.text = FALSE)

upscale2dPlot(object, args, ..., type = c("all",
                "gss", "ts", "bias", "rmse"))
```

## Arguments

| | |
|---|---|
| x | list object with components fss, fss.random and fss.uniform. Effectively, it does the same thing as hoods2dPlot, but adds the fss.random and fss.uniform horizontal lines to the matrix plot. |
| object | list object with named components: rmse (numeric vector), ets, ts and bias all matrices whose rows represent neighborhood lengths, and whose columns represent thresholds. |
| args | list object passed to hoods2dPlot, see its help file for more details. |
| mfrow | mfrow parameter (see help file for par). If NULL, then the parameter is not re-set. |
| add.text | logical, if TRUE, FSS values will be added to the quilt plot as text (in addition to the color). |
| type | character string stating which plots to make (default is "all"). |
| ... | Optional arguments to image and image.plot for fss2dPlot, and optional arguments to hoods2dPlot for upscale2dPlot |
| matplotcol | col argument to function matplot. |

## Details

makes quilt and matrix plots for output from hoods2d.

## Value

No value is returned. A series of plots are created. It may be useful to use this function in conjunction with pdf in order to view all of the plots. See the help file for hoods2dPlot to plot individual results.

## Author(s)

Eric Gilleland

## See Also

hoods2dPlot, matplot, image, image.plot, hoods2d, pdf

## Examples

```
##
## This is effectively an internal function, so the example is commented out
## in order for R's check to run faster.
##
## Not run:
data( "geom001" )
data( "geom000" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( geom000, geom001, thresholds = c(0.01,50.01),
    loc = ICPg240Locs, map = TRUE, projection = TRUE, loc.byrow = TRUE,
    units = "mm/h", data.name = "Geometric", obs.name = "observation",
    model.name = "case 1" )

look <- hoods2d(hold, levels=c(1, 3, 5, 33, 65),
    verbose=TRUE)
plot( look)

## End(Not run)
```

---

# GeoBoxPlot *Geographic Box Plot*

---

## Description

Make a geographic box plot as detailed in Willmott et al. (2007).

## Usage

```
GeoBoxPlot(x, areas, ...)
```

## Arguments

| | |
|---|---|
| x | numeric giving the values to be box-plotted. |
| areas | numeric of same length as x giving the associated areas for each value. |
| ... | optional arguments to the boxplot function of R. The argument plot is not allowed. |

## Details

This function makes the geographic box plots described in Willmott et al. (2007) that calculates the five statistics in such a way as to account for the associated areas (e.g., over a grid where each grid box may have differing areas).

Missing values are not handled, and ideally should be handled before calling ths routine.

In future, this function may allow other options for x than currently, but for now, only numeric vectors are allowed.

## Value

List with the same components as returned by `boxplot`.

## Author(s)

Eric Gilleland

## References

Willmott, C. J., Robeson, S. M. and Matsuura, K. (2007) Geographic box plots. *Physical Geography*, **28**, 331–344, doi:10.2747/0272-3646.28.4.331.

## See Also

`boxplot`

## Examples

```
##
## Reproduce the boxplots of Fig. 1 in Willmott et al. (2007).
##
x <- c(4,9,1,3,10,6,7)
a <- c(rep(1,4),2,1,3)
boxplot( x, at=1, xlim=c(0,3))
GeoBoxPlot(x, a, at=2, add=TRUE)
axis( 1, at=c(1,2), labels=c("Traditional", "Geographic"))
```

---

# GFSNAMfcstEx *Example Verification Set*

---

## Description

Example verification set of accumulated precipitation (mm) with 361 time points in addition to 2352 spatial locations on a grid. Taken from a real, but unknown, weather model and observation analysis (one of GFS or NAM). Accumulation is either 3-h or 24-h.

## Usage

```
data(GFSNAMfcstEx)
data(GFSNAMobsEx)
data(GFSNAMlocEx)
```

### Format

The format is: num [1:2352, 1:361] 0 0 0 0 0 ...

The format is: num [1:2352, 1:361] 0 0 0 0 0 ...

The format is: A data frame with 2352 observations on the following 2 variables.

`Lat` a numeric vector of latitude coordinates for GFS/NAM example verification set.

`Lon` a numeric vector of longitude coordinates for GFS/NAM example verification set.

### Details

Example verification set with 2352 spatial locations over the United States, and 361 time points. For both the forecast (`GFSNAMfcstEx`) and verification (`GFSNAMobsEx`), these are numeric matrices whose rows represent time, and columns represent space. The associated lon/lat coordinates are provided by `GFSNAMlocEx` (2352 by 2 data frame with named components giving the lon and lat values).

Note that the available spatial locations are a subset of the original 70 X 100 60-km grid where each time point had no missing observations. This example set is included with the package simply toi demonstrate some functionality that involves both space and time; though this is mostly a spatial-only package.

### Examples

```
data( "GFSNAMfcstEx" )
data( "GFSNAMobsEx" )
data( "GFSNAMlocEx" )

x <- colMeans(GFSNAMfcstEx, na.rm=TRUE)
y <- colMeans(GFSNAMobsEx, na.rm=TRUE)
look <- as.image(x - y, x=GFSNAMlocEx)
image.plot(look)
```

---

## gmm2d *2-d Gaussian Mixture Models Verification*

---

### Description

Use 2-d Gaussian Mixture Models (GMM) to assess forecast performance.

### Usage

```
gmm2d(x, ...)

## Default S3 method:
gmm2d(x, ..., xhat, K = 3, gamma = 1, threshold = NULL,
    initFUN = "initGMM", verbose = FALSE)

## S3 method for class 'SpatialVx'
```

```
gmm2d(x, ..., time.point = 1, obs = 1, model = 1, K = 3, gamma = 1,
    threshold = NULL, initFUN = "initGMM", verbose = FALSE)

## S3 method for class 'gmm2d'
plot(x, ..., col = c("gray", tim.colors(64)),
    zlim = c(0, 1), horizontal = TRUE)

## S3 method for class 'gmm2d'
predict(object, ..., x)

## S3 method for class 'gmm2d'
print(x, ...)

## S3 method for class 'gmm2d'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x,xhat | Default: m by n numeric matrices giving the verification and forecast fields, resp.<br>gmm2d.SpatialVx: object of class "SpatialVx".<br>plot and print: object returned by gmm2d.<br>predict: k by 2 matrix of lon/lat coordinates on which to predict the model. |
| object | output from gmm2d. |
| K | single numeric giving the number of mixture components to use. |
| gamma | Value of the gamma parameter from Eq (11) of Lakshmanan and Kain (2010). This affects the number of times a location is repeated. |
| threshold | numeric giving a threshold over which (and including) the GMM is to be fit (zero-valued grid points are not included in the estimation here for speed). If NULL, no thresholding is applied. |
| initFUN | character naming a function to provide initial estimates for the GMM. Must take an m by n matrix as input, and return a dataframe a component called ind that is a vector indicating the order of the rows for which the first K will be used, a third column giving the x-coordinates of the initial estimate of the mean for the x direction, fourth column giving the initial estimate for the mean of the y-direction, and fifth and sixth columns giving initial estimates for the standard deviations of the x- and y-directions. The default identifies all connected components using the disjointer function, then uses their centroids as the initial estimates of the means, and their axes as initial estimates for the standard deviations. The ind component gives the order of the object areas from largest to smallest so that the K largest objects are used to provide initial estimates. Note that this differs from the initial estimates in Lakshmanan and Kain (2010) where they break the field into different areas first. |
| verbose | logical, should progress information be printed to the screen? |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |

`obs, model` numeric indicating which observation/forecast model to select for the analysis.

`col, zlim, horizontal`
optional arguments to **fields** function(s) `poly.image`, `image.plot`.

`...` In the case of `gmm2d`: optional arguments to `initFUN`. In the case of `plot`: not used. In the case of `predict`: N by 2 matrix of grid point locations on which to predict the probability from the 2-d GMM model. In the case of `summary`: this can include the arguments: `silent`, logical stating whether to print summaries to the screen (FALSE) or not (TURE), `e1`, `e2`, ..., `e5`, giving alternative weights in calculating the overall error (Eq 15 in Lakshmanan and Kain, 2010, but see details section below).

## Details

These functions carry out the spatial verification approach described in Lakshmanan and Kain (2010), which fits a 2-d Gaussian Mixture Model (GMM) to the locations for each field in the verification set, and makes comparisons using the estimated parameters. In fitting the GMMs, first an initial estimate is provided by using the initFUN argument, which is a function. The default function is relatively fast (it might seem slow, but for what it does, it is very fast!), but is typically the slowest part of the process. Although the EM algorithm is a fairly computationally intensive procedure, acceleration algorithms are employed (via the turboem function of the turboEM package) so that once initial estimates are found, the procedure is very fast.

Because the fit is to the locations only, Lakshmanan and Kain (2010) suggest two ways to incorporate intensity information. The first is to repeat points with higher intensities, and the second is to multiply the results by the total intensities over the fields. The points are repeated M times according to the formula (Eq 11 in Lakshmanan and Kain, 2010):

M = 1 + gamma * round( CFD(I_xy)/frequency(I_MODE)),

where CFD is the cumulative *frequency* distribution (here estimated from the histogram using the 'hist' function), I_xy is intensity at grid point (x,y), I_MODE is the mode of intensity values, and gamma is a user-supplied parameter controlling how much to repeat points where higher numbers will result in larger repetitions of high intensity values.

The function `gmm2d` fits the 2-d GMM to both fields, `plot.gmm2d` first uses `predict.gmm2d` to obtain probabilities for each grid point, and then makes a plot similar to those in Lakshmanan and Kain (2010) Figs. 3, 4 and 5, but giving the probabilities instead of the probabilities times A. Note that `predict.gmm2d` can be very slow to compute so that `plot.gmm2d` can also be very slow. Less effort was put into speeding these functions up because they are not necessary for obtaining results via the parameters. However, they can give the user an idea of how good the fit is.

The 2-d GMM is given by

G(x,y) = A*sum(lambda*f(x,y))

where lambda and f(x,y) are numeric vectors of length K, lambda components describe the mixing, and f(x,y) is the bivariate normal distribution with mean (mu.x, mu.y) and covariance function. 'A' is the total sum of intensities over the field.

Comparisons between forecast and observed fields are carried out finally by the summary method function. In particular, the translation error

e.tr = sqrt((mu.xf - mu.xo)^2 + (mu.yf - mu.yo)^2),

where f means forecast and o verification fields, resp., and mu .x is the mean in the x- direction, and mu.y in the y- direction. The rotation error is given by

e.rot = (180/pi)*acos(theta),

where theta is the dot product between the first eigenvectors of the covariance matrices for the verification and forecast fields. The scaling error is given by

e.sc = Af*lambda.f/Ao*lambda.o,

where lambda is the mixture component and Af/Ao is the forecast/observed total intensity.

The overall error (Eq 15 of Lakshmanana and Kain, 2010) is given by

e.overall = e1 * min(e.tr/e2, 1) + e3*min(e.rot,180 - e.rot)/e4 + e5*(max(e.sc,1/e.sc)-1),

where e1 to e5 can be supplied by the user, but the defaults are those given by Lakshmanan and Kain (2010). Namely, e1 = 0.3, e2 = 100, e3=0.2, e4 = 90, and e5=0.5.

## Value

For gmm2d, a list object of class "gmm2d" is returned with components:

`fitX,fitY` list objects returned by the `turboem` function from the **turboEM** package that describe the EM estimates of the 2-d GMM parameters for the verification and forecast fields, resp.

`initX,initY` numeric vectors giving the initial estimates used in the EM algorithm for the verification and forecast fields, resp. The first 2*K values are the initial mean estiamtes for the x- and y- directions, resp. The next 4*K values are the initial estiamtes of the covariances (note that the cross-covariance terms are zero regardless of initialization function employed (maybe this will be improved in the future). The final K values are the initial estimates for lambda.

`sX,sY` N by 2 matrix giving the repeated coordinates calculated per M as described in the details section for the verification and forecast fields, resp.

`k` single numeric giving the value of K

`Ax,Ay` single numerics giving the value of A (the total sum of intensities over the field) for the verifiaction and forecast fields, resp.

For plot.gmm2d no value is returned. A plot is created.

For predict.gmm2d, a list is returned with components:

`predX,predY` numeric vectors giving the GMM predicted values for the verification and forecast fields, resp.

For summary.gmm2d, a list is returned invisibly (if silent is FALSE, information is printed to the screen) with components:

`meanX,meanY` Estimated mean vectors for each GMM component for the verification and forecast fields, resp.

`covX,covY` Estimated covariances for each GMM component for the verification and forecast fields, resp.

`lambdasX,lambdasY`
Estimated mixture components for each GMM component for the verification and forecast fields, resp.

e.tr,e.rot,e.sc,e.overall
: K by K matrices giving the errors between each GMM component in the verification field (rows) to each GMM component in the forecast field (columns). The errors are: translation (e.tr), rotation (e.rot), scaling (e.sc), and overall (e.overall).

## Author(s)

Eric Gilleland

## References

Lakshmanan, V. and Kain, J. S. (2010) A Gaussian Mixture Model Approach to Forecast Verification. *Wea. Forecasting*, **25** (3), 908–920.

## See Also

turboem, disjointer, connected

## Examples

```
## Not run:
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx
xhat <- ExampleSpatialVxSet$fcst

u <- min(quantile(c(x[x > 0]), probs = 0.75),
    quantile(c(xhat[xhat > 0]), probs = 0.75))

look <- gmm2d(x, xhat=xhat, threshold=u, verbose=TRUE)
summary(look)
plot(look)

## End(Not run)
## Not run:
# Alternative method to skin the cat.
hold <- make.SpatialVx( x, xhat, field.type = "MV Gaussian w/ Exp. Cov.",
    units = "units", data.name = "Example", obs.name = "x",
    model.name = "xhat" )

look2 <- gmm2d( hold, threshold = u, verbose = TRUE)
summary(look2)
plot(look2)

data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004, loc = ICPg240Locs,
    map = TRUE, projection = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
```

```
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- gmm2d(hold, threshold=10, verbose=TRUE)
plot(look) # This will take a long time!
summary(look)

## End(Not run)
```

---

`griddedVgram` *Variograms for a Gridded Verification Set*

---

## Description

Find (and plot) variograms for each field in a gridded verification set.

## Usage

```
griddedVgram(object, zero.in = TRUE, zero.out = TRUE, time.point = 1,
    obs = 1, model = 1, ...)

## S3 method for class 'griddedVgram'
plot(x, ... )
```

## Arguments

| | |
|---|---|
| `object` | list object of class "SpatialVx" containing information on the verification set. |
| `zero.in,zero.out` | logical, should the variogram be calculated over the entire field (zero.in), and/or over only the non-zero values (zero.out)? |
| `x` | list object as returned by `griddedVgram`. |
| `time.point` | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| `obs, model` | numeric indicating which observation/forecast model to select for the analysis. |
| `...` | In the case of `griddedVgram`, these are optional arguments to the `vgram.matrix` function from package `fields`. In the case of `plot.griddedVgram`, these are optional arguments to `plot.vgram.matrix`, which in turn are optional arguments to `image.plot`. |

## Details

Here, the terms semi-variogram and variogram are used interchangeably.

This is a simple wrapper function to `vgram.matrix` (entire field) from **fields** and/or `variogram.matrix` (non-zero grid points only) for finding the variogram between two gridded fields. It calls this function for each of two fields in a verification set. This function allows one to do the diagnostic analysis proposed in Marzban and Sangathe (2009).

## Value

A list object containing the entire list passed in by the object argument, and components:

`Vx.cgram.matrix, Fcst.vgram.matrix`
: list objects as returned by vgram.matrix containing the variogram information for each field.

No value is returned by plot.griddedVgram, plots are created showing the empirical variogram (circles), along with directional empirical variograms (dots), and the variogram by direction (image plot).

## Author(s)

Eric Gilleland

## References

Marzban, C. and Sandgathe, S. (2009) Verification with variograms. *Wea. Forecasting*, **24** (4), 1102–1120, doi:10.1175/2009WAF2222122.1.

## See Also

vgram.matrix, make.SpatialVx

## Examples

```
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx
xhat <- ExampleSpatialVxSet$fcst

hold <- make.SpatialVx( x, xhat, field.type = "contrived",
    units="none", data.name = "Example", obs.name = "x",
    model.name = "xhat" )

res <- griddedVgram( hold, R = 8 )
plot( res )

## Not run:
data( "pert004" )
data( "pert000" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Perturbed ICP Cases", obs.name = "pert000",
    model.name = "pert004" )

res <- griddedVgram(hold, R=8)
plot( res )
```

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

---

`hiw` *Spatial Forecast Verification Shape Analysis*

---

## Description

Shape analysis for spatial forecast verification (hiw is OE for shape; yields MnE hue).

## Usage

```
hiw(x, simplify = 0, A = pi * c(0, 1/16, 1/8, 1/6, 1/4, 1/2, 9/16, 5/8, 2/3, 3/4),
    verbose = FALSE, ...)

## S3 method for class 'hiw'
distill(x, ...)

## S3 method for class 'hiw'
plot(x, ..., which = c("X", "Xhat"), ftr.num = 1, zoom = TRUE,
    seg.col = "darkblue")

## S3 method for class 'hiw'
print(x, ...)

## S3 method for class 'hiw'
summary(object, ..., silent = FALSE)
```

## Arguments

| | |
|---|---|
| `x,object` | `hiw`: object of class "features".<br>`distill`, `plot`, `summary`: object of class "hiw". |
| `simplify` | `dmin` argument in call to `simplify.owin` from **spatstat**. If 0 (default), then no call is made to `simplify.owin`. |
| `A` | numeric vector of angles for which to apply shape analysis. Note that this vector will be rounded to 6 digits. If values are less than that, might be prudent to add 1e-6 to them. |
| `verbose` | logical, should progress information be printed to the screen? |
| `which` | character string naming whether to plot a feature from the obsevation field (default) or the forecast field. |
| `ftr.num` | integer stating which feature number to plot. |
| `zoom` | logical, should the feature be plotted within its original domain, or a blow-up of the feature (default)? |
| `seg.col` | color for the line segments. |
| `silent` | logical, should the summary information be printed to the screen? |
| `...` | Not used by `hiw`, `distill`, `summary`. Optional arguments to `plot`. |

## Details

This function is an attempt to approximate the technique described first in Micheas et al. (2007) and as modified in Lack et al. (2010). It will only find the centroids, rays extending from them to the boundaries, and the boundary points. Use `distill` to convert this output into an object readable by, for example, `procGPA` from package **shapes**.

First, identified features (which may be identified by any feature identification function that yields an object of class "features") are taken, the centroid is found (the centroid is found via `centroid.owin` so that the x- and y- coordinates are fliped from what you might expect) and very long line segments are found radiating out in both directions from the center. They are then clipped by where they cross the boundaries of the features.

The **spatstat** package is used heavily by this function. In particular, the function `as.polygonal` is applied to the `owin` objects (possible after first calling `simplify.owin`). Line segments are created using the feature centroids, as found by `centroid.owin`, and the user-supplied angles, along with a very long length (equal to the domain size). Boundary crossings are found using `crossing.psp`, and new line segment patterns are created using the centroids and boundary crossing information (extra points along line segments are subsequently removed through a painstaking process, and `as.psp` is called again, and any missing line segments are subsequently accounted for, for later calculations). Additionally, lengths of line segments are found via `lengths.psp`. Angles must also be re-determined and corresponded to the originally passed angles. Therefore, it is necessary to round the angles to 6 digits, or "equal" angles may not be considered equal, which will cause problems.

The `hiw` function merely does the above step, as well as finds the lengths of the resulting line segments. For non-convex objects, the longest line segment is returned, and if the boundary crossings do not lie on opposite sides of the centroid, then the negative of the shortest segment is returned for that particular value. Also returned are the mean, min and maximum intensities for each feature, as well as the final angles returned. It is possible to have missing values for some of these components.

The `summary` function computes SSloc, SSavg, SSmin, and SSmax between each pair of features between fields. `distill` may be used to create an object that can be further analyzed (for shape) using the **shapes** package.

While any feature identification function may be used, it is recommended to throw out small sized features as the results may be misleading (e.g., comparisons between features consisting of single points, etc.).

## Value

A list object of class "hiw" is returned with components the same as in the original "features" class object, as well as:

`radial.segments`
: a list with components X and Xhat each giving lists of the "psp" class (i.e., line segment) object for each feature containing the radial segments from the feature centroids to the boundaries.

`centers`
: list with components X and Xhat giving two-column matrices containing the x- and y- coordinate centroids for each feature (as determined by centroid.owin).

`intensities`
: list with components X and Xhat giving three-column matrices that contain the mean, min and max intensities for each feature.

`angles,lengths` list with components X and Xhat each giving lists containing the lengths of the line segments and their respective angles. Missing values are possible here.

distill returns an array whose dimensions are the number of landmarks (i.e., boundary points) by two by the number of observed and forecast features. An attribute called "field.identifier" is also given that is a character vector containing repeated "X" and "Xhat" values identiying which of the third dimension are associated with the observed field (X) and those identified with the forecast field (Xhat). Note that missing values may be present, which may need to be dealt with (by the user) before using functions from the shapes package.

summary invisibly returns a list object with components:

`X,Xhat` matrices whose rows represent features and whose columns give their centroids (note that x refers to the columns and y to the rows), as well as the average, min and max intensities.

`SS` matrix with four rows and columns equal to the number of possible combinations of feature matchings between fields. Gives the sum of square translation/location error (i.e., squared centroid distance), as well as the average, min and max squared differences between each combination of features.

`ind` two-column matrix whose rows indicate the feature numbers from each field being compared; corresponding to the columns of SS above.

## Author(s)

Eric Gilleland

## References

Lack, S. A., Limpert, G. L. and Fox, N. I. (2010) An object-oriented multiscale verification scheme. *Wea. Forecasting*, **25**, 79–92.

Micheas, A. C., Fox, N. I., Lack, S. A., and Wikle, C. K. (2007) Cell identification and verification of QPF ensembles using shape analysis techniques. *J. Hydrology*, **343**, 105–116.

## See Also

To indentify features and create objects of class "features", see, for example: `FeatureFinder`

`centroid.owin`, `as.psp`, `psp`, `crossing.psp`, `lengths.psp`, `angles.psp`

## Examples

```
data( "geom000" )
data( "geom001" )
data( "geom004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( geom000, geom001, thresholds = c(0.01, 50.01),
    projection = TRUE, map = TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Geometric Objects Pretending to be Precipitation",
    units = "mm/h", data.name = "ICP Geometric Cases", obs.name = "geom000",
```

```
    model.name = "geom001" )

look <- FeatureFinder(hold, do.smooth = FALSE, thresh = 2, min.size = 200)

look <- hiw(look)

distill.hiw(look)

# Actually, you just need to type:
# distill(look)

summary(look)

# Note: procGPA will not allow missing values.

par(mfrow=c(1,2))
plot(look)
plot(look, which = "Xhat")

## Not run:
hold <- make.SpatialVx( geom000, geom004, thresholds = c(0.01, 50.01),
    projection = TRUE, map = TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Geometric Objects Pretending to be Precipitation",
    units = "mm/h", data.name = "ICP Geometric Cases", obs.name = "geom000",
    model.name = "geom004" )

look <- FeatureFinder(hold, do.smooth = FALSE)

look <- hiw(look)

par(mfrow=c(1,2))
plot(look)
plot(look, which = "Xhat")

## End(Not run)
```

---

hoods2d *Neighborhood Verification Statistics for a Gridded Verification Set.*

---

## Description

Calculates most of the various neighborhood verification statistics for a gridded verification set as reviewed in Ebert (2008).

## Usage

```
hoods2d(object, which.methods = c("mincvr", "multi.event",
                 "fuzzy", "joint", "fss", "pragmatic"), time.point = 1,
                 obs = 1, model = 1, Pe = NULL, levels = NULL, max.n =
                 NULL, smooth.fun = "hoods2dsmooth", smooth.params =
```

```
                  NULL, rule = ">=", verbose = FALSE)

## S3 method for class 'hoods2d'
plot(x, ..., add.text = FALSE)

## S3 method for class 'hoods2d'
print(x, ...)
```

## Arguments

| | |
|---|---|
| object | list object of class "SpatialVx". |
| which.methods | character vector giving the names of the methods. Default is for the entire list to be executed. See Details section for specific option information. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| Pe | (optional) numeric vector of length q >= 1 to be applied to the fields sPy and possibly sPx (see details section). If NULL, then it is taken to be the most relaxed requirement (i.e., that an event occurs at least once in a neighborhood) of Pe=1/(nlen^2), where nlen is the length of the neighborhood. |
| levels | numeric vector giving the successive values of the smoothing parameter. For example, for the default method, these are the neighborhood lengths over which the levels^2 nearest neighbors are averaged for each point. Values should make sense for the specific smoothing function. For example, for the default method, these should be odd integers. |
| max.n | (optional) single numeric giving the maximum neighborhood length to use. Only used if levels are NULL. |
| smooth.fun | character giving the name of a smoothing function to be applied. Default is an average over the n^2 nearest neighbors, where n is taken to be each value of the `levels` argument. |
| smooth.params | list object containing any optional arguments to `smooth.fun`. Use NULL if none. |
| rule | character string giving the threshold rule to be applied. See help file for `thresholder` function for more information. |
| verbose | logical, should progress information be printed to the screen? Will also give the amount of time (in hours, minutes, or seconds) that the function took to run. |
| x | list object output from `hoods2d`. |
| add.text | logical, should the text values of FSS be added to its quilt plot? |
| ... | not used. |

## Details

`hoods2d` uses an object of class "SpatialVx" that includes some information utilized by this function, including the thresholds to be used. The neighborhood methods (cf. Ebert 2008, 2009; Gilleland et al., 2009, 2010) apply a (kernel) smoothing filter (cf. Hastie and Tibshirani, 1990) to either

the raw forecast (and possibly also the observed) field(s) or to the binary counterpart(s) determined by thresholding.

The specific smoothing filter applied for these methods could be of any type, but those described in Ebert (2008) are generally taken to be "neighborhood" filters. In some circles, this is referred to as a convolution filter with a boxcar kernel. Because the smoothing filter can be represented this way, it is possible to use the convolution theorem with the Fast Fourier Transform (FFT) to perform the neighborhood smoothing operation very quickly. The particular approach used here "zero pads" the field, and replaces all missing values with zero as well, which is also the approach proposed in Roberts and Lean (2008). If any missing values are introduced after the convolution, they are removed.

To simplify the notation for the descriptions of the specific methods employed here, the notation of Ebert (2008) is adopted. That is, if a method uses neighborhood smoothed observations (NO), then the neighborhood smoothed observed field is denoted <X>s, and the associated binary field, by <Ix>s. Otherwise, if the observation field is not smoothed (denoted by SO in Ebert, 2008), then simply X or Ix are used. Similarly, for the forecast field, <Y>s or <Iy>s are used for neighborhood smoothed forecast fields (NF). If it is the binary fields that are smoothed (e.g., the original fields are thresholded before smoothing), then the resulting fields are denoted <Px>s and <Py>s, resp. Below, NO-NF indicates that a neighborhood smoothed observed field (<Yx>s, <Ix>s, or <Px>s) is compared with a neighborhood smoothed forecast field, and SO-NF indicates that the observed field is not smoothed.

Options for which.methods include:

"mincvr": (NO-NF) The minimum coverage method compares <Ix>s and <Iy>s by thresholding the neighborhood smoothed fields <Px>s and <Py>s (i.e., smoothed versions of Ix and Iy) to obtain <Ix>s and <Iy>s. Indicator fields <Ix>s and <Iy>s are created by thresholding <Px>s and <Py>s by frequency threshold `Pe` given by the `obj` argument. Scores calculated between <Ix>s and <Iy>s include: probability of detecting an event (pod, also known as the hit rate), false alarm ratio (far) and ets (cf. Ebert, 2008, 2009).

"multi.event": (SO-NF) The Multi-event Contingency Table method compares the binary observed field Ix against the smoothed forecast indicator field, <Iy>s, which is determined similarly as for "mincvr" (i.e., using Pe as a threshold on <Py>s). The hit rate and false alarm rate (F) are calculated (cf. Atger, 2001).

"fuzzy": (NO-NF) The fuzzy logic approach compares <Px>s to <Py>s by creating a new contingency table where hits = sum_i min(<Px>s_i,<Py>s_i), misses = sum_i min(<Px>s_i,1-<Py>s_i), false alarms = sum_i min(1-<Px>s_i,<Py>s_i), and correct negatives = sum_i min(1-<Px>s_i,1-<Py>s_i) (cf. Ebert 2008).

"joint": (NO-NF) Similar to "fuzzy" above, but hits = sum_i prod(<Px>s_i,<Py>s_i), misses = sum_i prod(<Px>s_i,1-<Py>s_i), false alarms = sum_i prod(1-<Px>s_i,<Py>s_i), and correct negatives = sum_i prod(1-<Px>s_i,1-<Py>s_i) (cf. Ebert, 2008).

"fss": (NO-NF) Compares <Px>s and <Py>s directly using a Fractions Brier and Fractions Skill Score (FBS and FSS, resp.), where FBS is the mean square difference between <Px>s and <Py>s, and the FSS is one minus the FBS divided by a reference MSE given by the sum of the sum of squares of <Px>s and <Py>s individually, divided by the total (cf. Roberts and Lean, 2008).

"pragmatic": (SO-NF) Compares Ix with <Py>s, calculating the Brier and Brier Skill Score (BS and BSS, resp.), where the reference forecast used for the BSS is taken to be the mean square error between the base rate and Ix (cf. Theis et al., 2005).

## Value

A list object of class "hoods2d" with components determined by the `which.methods` argument. Each component is itself a list object containing relevant components to the given method. For example, hit rate is abbreviated pod here, and if this is an output for a method, then there will be a component named pod (all lower case). The Gilbert Skill Score is abbreviated 'ets' (equitable threat score; again all lower case here). The list components will be some or all of the following.

| | |
|---|---|
| `mincvr` | list with components: pod, far and ets |
| `multi.event` | list with components: pod, f and hk |
| `fuzzy` | list with components: pod, far and ets |
| `joint` | list with components: pod, far and ets |
| `fss` | list with components: fss, fss.uniform, fss.random |
| `pragmatic` | list with components: bs and bss |

New attributes are added giving the values for some of the optional arguments: levels, max.n, smooth.fun, smooth.params and Pe.

## Note

Thresholded fields are taken to be >= the threshold.

## Author(s)

Eric Gilleland

## References

Atger, F. (2001) Verification of intense precipitation forecasts from single models and ensemble prediction systems. *Nonlin. Proc. Geophys.*, **8**, 401–417.

Ebert, E. E. (2008) Fuzzy verification of high resolution gridded forecasts: A review and proposed framework. *Meteorol. Appl.*, **15**, 51–64. doi:10.1002/met.25

Ebert, E. E. (2009) Neighborhood verification: A strategy for rewarding close forecasts. *Wea. Forecasting*, **24**, 1498–1510, doi:10.1175/2009WAF2222251.1.

Gilleland, E., Ahijevych, D., Brown, B. G., Casati, B. and Ebert, E. E. (2009) Intercomparison of Spatial Forecast Verification Methods. *Wea. Forecasting*, **24**, 1416–1430, doi:10.1175/2009WAF2222269.1.

Gilleland, E., Ahijevych, D. A., Brown, B. G. and Ebert, E. E. (2010) Verifying Forecasts Spatially. *Bull. Amer. Meteor. Soc.*, October, 1365–1373.

Hastie, T. J. and Tibshirani, R. J. (1990) *Generalized Additive Models*. Chapman \& Hall/CRC Monographs on Statistics and Applied Probability 43, 335pp.

Roberts, N. M. and Lean, H. W. (2008) Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Mon. Wea. Rev.*, **136**, 78–97. doi:10.1175/2007MWR2123.1.

Theis, S. E., Hense, A. Damrath, U. (2005) Probabilistic precipitation forecasts from a deterministic model: A pragmatic approach. *Meteorol. Appl.*, **12**, 257–268.

Yates, E., Anquetin, S. Ducrocq, V., Creutin, J.-D., Ricard, D. and Chancibault, K. (2006) Point and areal validation of forecast precipitation fields. *Meteorol. Appl.*, **13**, 1–20.

Zepeda-Arce, J., Foufoula-Georgiou, E., Droegemeier, K. K. (2000) Space-time rainfall organization and its role in validating quantitative precipitation forecasts. *J. Geophys. Res.*, **105**(D8), 10,129–10,146.

## See Also

fft, kernel2dsmooth, plot.hoods2d, vxstats, thresholder

## Examples

```
x <- y <- matrix( 0, 50, 50)
x[ sample(1:50,10), sample(1:50,10)] <- rexp( 100, 0.25)
y[ sample(1:50,20), sample(1:50,20)] <- rexp( 400)
hold <- make.SpatialVx( x, y, thresholds = c(0.1, 0.5), field.type = "Random Exp. Var." )
look <- hoods2d( hold, which.methods=c("multi.event", "fss"), levels=c(1, 3, 19))
look
plot(look)

## Not run:
data( "UKobs6" )
data( "UKfcst6" )
data( "UKloc" )

hold <- make.SpatialVx( UKobs6, UKfcst6, thresholds = c(0.01, 20.01),
    projection = TRUE, map = TRUE, loc = UKloc, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Nimrod", obs.name = "Observations 6",
    model.name = "Forecast 6" )

hold
plot(hold)
hist(hold, col="darkblue")

look <- hoods2d(hold, which.methods=c("multi.event", "fss"),
    levels=c(1, 3, 5, 9, 17), verbose=TRUE)
plot(look)

data( "geom001" )
data( "geom000" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( geom000, geom001, thresholds = c(0.01, 50.01),
    projection = TRUE, map = TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Geometric Objects Pretending to be Precipitation",
    units = "mm/h", data.name = "ICP Geometric Cases", obs.name = "geom000",
    model.name = "geom001" )

look <- hoods2d(hold, levels=c(1, 3, 9, 17, 33, 65, 129, 257), verbose=TRUE)

plot( look) # Might want to use 'pdf' first.

data( "pert004" )
```

```
data( "pert000" )

hold <- make.SpatialVx( pert000, pert004, thresholds = c(1, 10, 50),
    projection = TRUE, map = TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004" )

plot(hold)

look <- hoods2d(hold, levels=c(1, 3, 17, 33, 65, 129, 257), verbose=TRUE)
plot(look)

## End(Not run)
```

---

`hoods2dPlot` *Quilt Plot and a Matrix Plot.*

---

## Description

Function to make a quilt plot and a matrix plot for a matrix whose rows represent neighborhood lengths, and whose columns represent different threshold choices.

## Usage

```
hoods2dPlot(x, args, matplotcol = 1:6, ...)
```

## Arguments

| | |
|---|---|
| `x` | l by q numeric matrix. |
| `args` | list object with components: threshold (numeric vector giving the threshold values), qs (optional numeric vector giving the quantiles used if the thresholds represent quantiles rather than hard values), levels (numeric giving the neighborhood lengths (in grid squares) used, units (optional character giving the units for the thresholds) |
| `matplotcol` | col argument to `matplot`. |
| `...` | optional arguments to `image` and `image.plot` functions. May not include xaxt, yaxt, lab, lab, col, or legend.only |

## Details

Used by `plot.hoods2d`, but can be useful for other functions. Generally, however, this is an internal function that should not be called by the user. However, it might be called instead of `plot.hoods2d` in order to make a subset of the available plots.

## Value

No value is returned. A plot is created.

## Author(s)

Eric Gilleland

## References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

## See Also

matplot, image, image.plot, plot.hoods2d, hoods2d

## Examples

```
x <- y <- matrix( 0, 50, 50)
x[ sample(1:50,10), sample(1:50,10)] <- rexp( 100, 0.25)
y[ sample(1:50,20), sample(1:50,20)] <- rexp( 400)

hold <- make.SpatialVx(x, y, thresholds=c(0.1, 0.5), field.type="random")

look <- hoods2d(hold, which.methods=c("multi.event", "fss"),
    levels=c(1, 3, 20))

hoods2dPlot( look$multi.event$hk, args=hold,
    main="Hanssen Kuipers Score (Multi-Event Cont. Table)")
## Not run:
data( "geom001" )
data( "geom000" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( geom000, geom001, thresholds = c(0.01, 50.01),
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Geometric ICP Test Cases", obs.name = "geom000",
    model.name = "geom001" )

look <- hoods2d(hold, levels=c(1, 3, 5, 17, 33, 65), verbose=TRUE)
par(mfrow=c(1,2))
hoods2dPlot(look$pragmatic$bss, args=attributes(hold))

## End(Not run)
```

---

hump *Simulated Forecast and Verification Fields*

---

## Description

Simulated forecast and verification fields for optical flow example

## Usage

```
data(hump)
```

## Format

The format is: List of 2 $ initial: num [1:50, 1:50] 202 210 214 215 212 ... $ final : num [1:50, 1:50] 244 252 257 258 257 ...

## Details

Although not identically the same as the data used in Fig. 1 of Marzban and Sandgathe (2010), these are forecast data simulated from the self-same distribution and perturbed in the same manner to get the observation. The component initial is the forecast and final is the observation.

The forecast is on a 50 X 50 grid simulated from a bivariate Gaussian with standard deviation of 11 and centered on the coordinate (10, 10). The observed field is the same as the forecast field, but shifted one grid length in each direction and has 60 added to it everywhere.

## References

Marzban, C. and Sandgathe, S. (2010) Optical flow for verification. *Wea. Forecasting*, **25**, 1479–1494, doi:10.1175/2010WAF2222351.1.

## Examples

```
data(hump)
str(hump)
## Not run:
initial <- hump$initial
final <- hump$final
look <- OF(final, initial, W=9, verbose=TRUE)
plot(look) # Compare with Fig. 1 in Marzban and Sandgathe (2010).
hist(look) # 2-d histogram.
plot(look, full=TRUE) # More plots.

## End(Not run)
```

---

# imomenter *Image Moments*

---

## Description

Calculate some of the raw image moments, as well as some useful image characteristics.

## Usage

```
imomenter(x, loc = NULL, ...)

## S3 method for class 'im'
imomenter(x, loc = NULL, ...)

## S3 method for class 'matrix'
imomenter(x, loc = NULL, ...)

## S3 method for class 'imomented'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | imomenter: matrix or object of class "im" (from package **spatstat**).<br>print: object of class "imomenter". |
| loc | A two-column matrix giving the location coordinates. May be missing in which case they are assumed to be integers giving the row and column numbers. |
| ... | Not used. |

## Details

Calculates Hu's image moments (Hu 1962). Calculates the raw moments: M00 (aka area), M10, M01, M11, M20, and M02, as well as the (normalized) central moments: mu11', mu20', and mu02', which are returned as the image covariance matrix: rbind(c(mu20', mu11'), c(mu11', mu02')). In addition, the image centroid and orientation angle are returned, as calculated using the image moments. It should be noted that while the centroid is technically defined for the null case (all zero-valued grid points), the way it is calculated using image moments means that it will be undefined because of division by zero in the formulation.

The orientation angle calculated here is that which is used by MODE, although not currently used in the MODE analyses in this package (smatr is used instead to find the major axis, etc). The eigenvalues of the image covariance correspond to the major and minor axes of the image.

For more information on image moments, see http://en.wikipedia.org/wiki/Image_moments and references therein.

## Value

A list object of class "imomented" is returned with components:

| | |
|---|---|
| area | Same as M00. |
| centroid | numeric with named components "x" and "y" giving the x- and y- coordinates of the centroid as calculated by the image moment method. |
| orientation.angle | The orientation angle of the image as calculated by image moments. |
| raw.moments | named numeric vector with the raw image moments: M00, M10, M01, M11, M20 and M02 used in calculating the other returned values. |
| cov | 2 by 2 image covariance as calculated by the image moment method. |

## Author(s)

Eric Gilleland

## References

Hu, M. K. (1962) Visual Pattern Recognition by Moment Invariants. *IRE Trans. Info. Theory*, **IT-8**, 179–187.

## See Also

Mij, FeatureAxis

## Examples

```
look <- matrix(0, 10, 10)

look[3:5, 7:8] <- rnorm(6)

imomenter(look)

## Not run:
data( "geom000" )
data( "ICPg240Locs" )

imomenter( geom000 )
imomenter( geom000, loc = ICPg240Locs )

data( "geom004" )

imomenter( geom004 )

imomenter( geom004, loc = ICPg240Locs )

## End(Not run)
```

---

# interester *Feature Interest*

---

## Description

Calculate interest maps for specific feature comparisons and compute the total interest, as well as median of maximum interest.

## Usage

```
interester(x, properties = c("cent.dist", "angle.diff", "area.ratio", "int.area",
    "bdelta", "haus", "ph", "med", "msd", "fom", "minsep"),
    weights = c(0.24, 0.12, 0.17, 0.12, 0, 0, 0, 0, 0, 0, 0.35),
    b1 = c(35, 30, 0, 0, 0.5, 35, 20, 40, 120, 1, 40),
    b2 = c(100, 90, 0.8, 0.25, 85, 400, 200, 200, 400, 0.25, 200),
    verbose = FALSE, ...)

## S3 method for class 'interester'
print(x, ...)

## S3 method for class 'interester'
summary(object, ...,
    min.interest = 0.8, long = TRUE, silent = FALSE)
```

## Arguments

| | |
|---|---|
| x | interester: object of class "features" or "matched".<br>print: object of class "interester". |
| object | object of class "interester". |
| properties | character vector naming which properties from FeatureComps should be considered in the total interest. |
| weights | numeric of length equal to the length of properties. Weights equal to zero will result in the removal of those properties from properties so that the default computes only those values utilized in Davis et al (2009). |
| b1,b2 | All interest maps (except that for "fom") are piecewise linear, and of the form: f(x) = 1,0 (depending on the property) if x < b1, x <= b1, x > b2 or x >= b2 (see details for more information). |
| verbose | logical, should progress information be printed to the screen? |
| min.interest | numeric between zero and one giving the desired minimum value of interest. Only used for display purposes. If long is TRUE, a dashed line is printed where the total interest falls below this value. If long is FALSE, only the total interest values above this value are displayed. All values are returned invisibly regardless of the values of min.interest and long. |
| long | logical, should all interest values be displayed (TRUE) or only those above min.interest (FALSE)? |
| silent | logical, should summary information be displayed to the screen (FALSE)? |
| ... | interester: optional arguments to FeatueComps.<br>Not used by print or summary. |

## Details

This function calculates the feature interest according to the MODE algorithm described in Davis et al (2009). Properties that can be computed are those available in FeatureComps, except for "bearing". Interest maps are computed according to piece-wise linear functions (except for "fom")

depending on the property. For all properties besides "area.ratio", "int.area" and "fom", the interest maps are of the form:

f(x) = 1, if x <= b1

f(x) = a0 + a1 * x, if x > b1 and x <= b2, where a1 = -1/(b2 - b1) and a0 = 1 - a1 * b1

f(x) = 0, if x > b2

For properties "area.ratio" and "int.area", the interest maps are of the form:

f(x) = 0, if x < b1

f(x) = a0 + a1 * x, if x >= b1 and x < b2, where a1 = 1/(b2 - b1) and a0 = 1 - a1 * b2

f(x) = 1, if x >= b2

Finally, for "fom", a function that tries to give as much weight to values near one is applied. It is given by:

f(x) = b1 * exp(-0.5 * ((x - 1) / b2)^4)

The default values for b1 and b2 will not necessarily give the same results as in Davis et al (2009), but also, the distance map for their intersection area ratio differs from that here. The interest function for FOM is further restricted to fall within the interval [0, 1], so care should be taken if b1 and/or b2 are changed for this function.

The `interester` function calculates the individual interest values for each property and each pair of features, and returns both these individual interest values as well as a matrix of total interest. The `print` function will print the entire matrix of individual interest values if there are fewer than twenty pairs of features, and will print their summary otherwise. The `summary` function will order the total interest from highest to lowest and print this information (along with which feature pairs correspond to the total interest value). It will also calculate the median of maximum interest (MMI) as suggested by Davis et al (2009). If there is only one feature in either field, then this value will just be the maximum total interest.

The centroid distance property is less meaningful if the sizes of the two features differ greatly, and therefore, the interest value for this property is further multiplied by the area ratio of the two features. Similarly, angle difference is less meaningful if one or both of the features are circular in shape. Therefore, this property is multiplied by the following factor, following Davis et al (2009) Eq (A1), where r1 and r2 are the aspect ratios (defined as the length of the minor axis divided my that of the major axis) of the two features, respectively.

sqrt( [ (r1 - 1)^2 / (r1^2 + 1) ]^0.3 * [ (r2 - 1)^2 / (r2^2 + 1) ]^0.3 )

The `print` function displays either the individualinterest values for each property and feature pairings, or more often, a summary of these values (if the display would otherwise be too large). It also shows a matrix whose rows are observed features and columns forecast features, with the total interest values therein associated.

`summary` shows the sorted total interest from highest to lowest for each pair. A dashed line separates the values above `min.interest` from those below, and if `long` is TRUE, then values below that line are not displayed. It also reports the median of maximum interest (MMI) defined by Davis et al (2009) as an overall feature-based summary of forecast performance. It is derived by collecting the row maxima and column maxima from the total interest matrix, shown by the `print` command, into a vector, and then finding the median of this vector.

## Value

A list of class "interester" is returned with components:

`interest` matrix whose named rows correspond to the each property that was calculated and whose columns are feature pairings. The values are the interest calculated for the specific property and pair of features.

`total.interest` matrix of total interest for each pair of features where rows are observed features and columns are forecast features.

If no features are available in either field, NULL is returned.

Nothing is returned by print.

summary invisibly returns a list object of class "summary.interester" with components:

`sorted.interest` similar to the interest component from the value returned by interester, but sorted from highest to lowest interest, along with the feature number information for each field.

`mmi` the median of maximum interest value.

## Note

The terminology used for features within the entire **SpatialVx** package attempts to avoid conflict with terminology used by R. So, for example, the term property is used in lieu of "attributes" so as not to be confused with R object attributes. The term "feature" is used in place of "object" to avoid confusion with an R object, etc.

## Author(s)

Eric Gilleland

## References

Davis, C. A., Brown, B. G., Bullock, R. G. and Halley Gotway, J. (2009) The Method for Object-based Diagnostic Evaluation (MODE) applied to numerical forecasts from the 2005 NSSL/SPC Spring Program. *Wea. Forecsting*, **24**, 1252–1267, DOI: 10.1175/2009WAF2222241.1.

## See Also

Identifying features: `FeatureFinder`

Functions for calculating the properties: `FeatureComps`, `FeatureProps`

## Examples

```
x <- y <- matrix(0, 100, 100)
x[ 2:3, c(3:6, 8:10) ] <- 1
y[ c(4:7, 9:10), c(7:9, 11:12) ] <- 1
```

```
x[ 30:50, 45:65 ] <- 1
y[ c(22:24, 99:100), c(50:52, 99:100) ] <- 1

hold <- make.SpatialVx(x, y, field.type = "contrived", units = "none",
    data.name = "Example", obs.name = "x", model.name = "y" )

look <- FeatureFinder(hold, smoothpar = 0.5)

look2 <- interester(look)
look2
summary(look2)

## Not run:
data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004,
    loc = ICPg240Locs, projection = TRUE, map=TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- FeatureFinder( hold, smoothpar=10.5, thresh = 5 )

look2 <- interester(look, verbose = TRUE)
look2
summary(look2)


## End(Not run)
```

---

`locmeasures2d` *Binary Image Measures*

---

## Description

Calculate some binary image measures between two fields.

## Usage

```
locmeasures2d(object, which.stats = c("bdelta", "haus", "qdmapdiff",
    "med", "msd", "ph", "fom"), distfun = "distmapfun", distfun.params = NULL,
    k = NULL, alpha = 0.1, bdconst = NULL, p = 2, ...)

## Default S3 method:
locmeasures2d(object, which.stats = c("bdelta", "haus", "qdmapdiff",
    "med", "msd", "ph", "fom"), distfun = "distmapfun", distfun.params = NULL,
    k = NULL, alpha = 0.1, bdconst = NULL, p = 2, ..., Y, thresholds=NULL)
```

```
## S3 method for class 'SpatialVx'
locmeasures2d(object, which.stats = c("bdelta", "haus", "qdmapdiff",
    "med", "msd", "ph", "fom"), distfun = "distmapfun", distfun.params = NULL,
    k = NULL, alpha = 0.1, bdconst = NULL, p = 2, ..., time.point = 1,
    obs = 1, model = 1)

## S3 method for class 'locmeasures2d'
print(x, ...)

## S3 method for class 'locmeasures2d'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| object | For locmeasures2d, an object of class "SpatialVx" or a valid matrix in which case Y must be explicitly provided. For summary method function, a list object output from locmeasures2d. |
| x | returned object from locmeasures2d. |
| which.stats | character vector telling which measures should be calculated. |
| distfun | character naming a function to calculate the shortest distances between each point x in the grid and the set of events. Default is the Euclidean distance metric. Must take x as an argument, which is the event field for which the distances are to be calculated. Must return a matrix of the same dimension as x. |
| distfun.params | list with named components giving any additional arguments to the distfun function. |
| k | numeric vector for use with the partial Hausdorff distance. For k that are whole numerics or integers >= 1, then the k-th highest value is returned by locmeasures2d. If 0 <= k < 1, then the corresponding quantile is returned. |
| alpha | numeric giving the alpha parameter for Pratt's Figure of Merit (FOM). See the help file for locperf for more details. |
| bdconst | numeric giving the cut-off value for Baddeley's delta metric. |
| p | numeric vector giving one or more values for the parameter p in Baddeley's delta metric. Usually this is just 2. |
| Y | m X n matrix giving the forecast field. |
| thresholds | numeric or two-column matrix giving the threshold to be applied to the verification (column one) and forecast (column two) fields. If a vector, same thresholds are applied to both fields. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| ... | optional arguments to deltametric and distmap from package spatstat. Not used by the summary or print methods here. |

## Details

It is useful to introduce some notation. Let d(x,A) be the shortest distance from a point x, anywhere in the grid, to a set A contained in the grid. Here, Euclidean distance is used (default) for d(x,A), but note that some papers (e.g., Venugopal et al., 2005) use other distances, such as the taxi-cab distance (use `distfun` argument to change the distance method).

The Hausdorff distance between two sets A and B contained in the finite grid is given by max( max( d(x,A), x in B), max( d(x,B), x in A)), and can be re-written as H(A,B) = max( abs( d(x,A) - d(x,B))), where x is taken over all points in the grid. Several of the distances here are modifications of the Hausdorff distance. The Baddeley metric, for example, is the Lp norm of abs( w(d(x,A)) - w(d(x,B))), where again x is taken from over the entire grid, and w is any concave continuous function that is strictly increasing at zero. Here, w(t) = min( t, c), where c is some constant given by the `bdconst` argument.

Calculates one or more of the following binary image measures:

"bdelta" Baddeley delta metric (Baddeley, 1992a,b; Gilleland, 2011; Schwedler and Baldwin, 2011)

"haus" Hausdorff distance (Baddeley, 1992b; Schwedler and Baldwin, 2011)

"qdmapdiff" Quantile (or rank) of the differences in distance maps. See the help file for `locperf`.

"med" Mean Error Distance (Peli and Malah, 1982; Baddeley, 1992a). See the help file for `locperf`.

"msd" Mean Square Error Distance (Peli and Malah, 1982; Baddeley, 1992a). See the help file for `locperf`.

"ph" Partial Hausdorff distance. See the help file for `locperf`.

"fom" Pratt's Figure of Merit (Peli and Malah, 1982; Baddeley, 1992a, Eq (1)). See the help file for `locperf`.

These distances are summaries in and of themselves, so the summary method function simply displays the results in an easy to read manner.

## Value

A list with at least one of the following components depending on the argument which.stats

| | |
|---|---|
| `bdelta` | p by q matrix giving the Baddeley delta metric for each desired value of p (rows) and each threshold (columns) |
| `haus` | numeric vector giving the Hausdorff distance for each threshold |
| `qdmapdiff` | k by q matrix giving the difference in distance maps for each of the k-th largest value(s) or quantile(s) (rows) for each threshold (columns). |
| `medMiss, medFalseAlarm, msdMiss, msdFalseAlarm` | two-row matrix giving the mean error (or square error) distance as (Forecast, Observation) or misses and (Observation, Forecast) or false alarms. |
| `ph` | k by q matrix giving the k-th largest value(s) or quantile(s) (rows) for each threshold (columns) of the maximum between the distances from one field to the other. |
| `fom` | numeric vector giving Pratt's figure of merit. |

## Note

Binary fields are determined by having values >= the thresholds.

## Author(s)

Eric Gilleland

## References

Baddeley, A. (1992a) An error metric for binary images. In *Robust Computer Vision Algorithms*, W. Forstner and S. Ruwiedel, Eds., Wichmann, 59–78.

Baddeley, A. (1992b) Errors in binary images and an Lp version of the Hausdorff metric. *Nieuw Arch. Wiskunde*, **10**, 157–183.

Gilleland, E. (2011) Spatial forecast verification: Baddeley's delta metric applied to the ICP test cases. *Wea. Forecasting*, **26**, 409–415, doi:10.1175/WAF-D-10-05061.1.

Peli, T. and Malah, D. (1982) A study on edge detection algorithms. *Computer Graphics and Image Processing*, **20**, 1–21.

Schwedler, B. R. J. and Baldwin, M. E. (2011) Diagnosing the sensitivity of binary image measures to bias, location, and event frequency within a forecast verification framework. *Wea. Forecasting*, **26**, 1032–1044, doi:10.1175/WAF-D-11-00032.1.

Venugopal, V., Basu, S. and Foufoula-Georgiou, E. (2005) A new metric for comparing precipitation patterns with an application to ensemble forecasts. *J. Geophys. Res.*, **110**, D08111, doi:10.1029/2004JD005395, 11pp.

## See Also

deltametric, distmap, make.SpatialVx

## Examples

```
x <- y <- matrix(0, 10, 12)
x[2,3] <- 1
y[4,7] <- 1

hold <- make.SpatialVx(x, y, thresholds = 0.1,
    field.type = "random", units = "grid squares")
locmeasures2d(hold, k = 1)

# Alternatively ...
locmeasures2d(x, thresholds = 0.1, k = 1, Y = y)

## Not run:
data( "geom000" )
data( "geom001" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( geom000, geom001, thresholds = c(0.1, 50.1),
    projection = TRUE, map=TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Precipitation", units = "in/100",
    data.name= "ICP Geometric Cases", obs.name = "geom000",
    model.name = "geom001" )

hold2 <- locmeasures2d(hold, k=c(4, 0.975), alpha=c(0.1,0.9))
```

```
summary(hold2)

## End(Not run)
```

---

`locperf` *Localization Performance Measures*

---

## Description

Some localization performance (distance) measures for binary images.

## Usage

```
locperf(X, Y, which.stats = c("qdmapdiff", "med", "msd", "ph", "fom", "minsep"),
    alpha = 0.1, k = 4, distfun = "distmapfun", a=NULL, ...)

distob(X, Y, distfun = "distmapfun", ...)

distmapfun(x, ...)
```

## Arguments

| | |
|---|---|
| `X` | list object giving a pixel image as output from `solutionset` from package **spatstat**. This corresponds to the set B in the Details section below. |
| `Y` | list object giving a pixel image as output from `solutionset` from package **spatstat**. This corresponds to the set A in the Details section below. |
| `x` | list object of class "owin" as returned by `solutionset` from package **spatstat**. |
| `which.stats` | character vector stating which localization performance measure to calculate. |
| `alpha` | numeric giving the scaling constant for Pratt's figure of merit (FOM). Only used for `which.stat` method "fom". |
| `k` | single numeric giving the order for the rank/quantile of the difference in distance maps. If 0 <= k < 1, this is assumed to be a quantile for use with the `quantile` function. Otherwise, k should be a whole number such that 1 <= k <= Nxy, where nxy is the total number of grid points in the set. |
| `distfun` | character specifying a distance metric that returns a matrix of same dimension as `X` yielding, at each point `x`, the shortest distances from `x` to the set of events in the field. Default is `distmapfun`, which returns the Euclidean distances. |
| `a` | Not used. For compatibility with `locmeasures2d`. |
| `...` | Optional arguments to the `distfun` function. In the case of `distmapfun`, these are the optional arguments to `distmap` from package **spatstat**. |

## Details

This function computes localization performance (or distance) measures detailed in Peli and Malah (1982) and Baddeley (1992), as well as a modification of one of these distances detailed in Zhu et al. (2011); `distob`.

First, it is helpful to establish some notation. Suppose a distance rho(x,y) is defined between any two pixels x and y in the entire raster of pixels/grid (If `distfun` is `distmapfun` (default), then rho is the Euclidean distance) that satisfies the formal mathematical axioms of a metric. Let d(x,A) denote the shortest distance (smallest value of rho) from the point x in the entire raster to the the set A contained in the raster. That is, d(x,A) = min(rho(x,a): a in A contained in the raster) [formally, the minimum should be the infimum], with d(x, empty set) defined to be infinity. Note that the `distfun` argument is a function that returns d(x,A) for all x in the raster.

The mean error distance ("med") is the mean of d(x,A) over the points in B. That is e.bar = mean( d(x,A)), over all x in B. Because it is not symmetric (i.e., MED(A, B) != MED(B, A)), it is given as medMiss = MED(Forecast, Observation) and medFalseAlarm = MED(Observation, Forecast).

The mean square error distance ("msd") is the mean of the squared d(x,A) over the points in B. That is, e2.bar = mean( d(x,A)^2), over all x in B. Similarly to MED, it is given as msdMiss or msdFalseAlarm.

Pratt's figure of merit ("fom") is given by: FOM(A,B) = sum( 1/(1+alpha*d(x,A)^2))/max(N(A),N(B)), where x in B, and N(A) (N(B)) is the number of points in the set A (B) and alpha is a scaling constant (see, e.g., Pratt, 1977; Abdou and Pratt, 1979). The scaling constant is typically set to 1/9 when rho is normalized so that the smallest nonzero distance between pixel neighbors is 1. The default (0.1) here is approximately 1/9. If both A and B are empty, the value returned for max(N(A), N(B)) is 1e16 and for d(x,A) for x in B is given a value of zero so that the returned value should be close to zero.

Minimum separation distance between boundaries ("minsep") is just the smallest value of the distance map of one field over the subset where events occur in the other. This is mainly for when single features within the fields are being compared.

distob is a modification of the mean error distance where if there are no events in either field, the value is 0, and if there are no events in one field only, the value is something large (in this case the length of the longest side of the grid).

The Hausdorff distance for a finite grid is given by max( max( d(x,B); x in A), max( d(x,A); x in B)), and can be written as max( abs(d(x,A) - d(x,B)), over all x in the raster). The quantile of the difference in distance mapse ("qdmapdiff") is also potentially useful, and replaces the maximum in the latter equation with a k-th order statistic (or quantile). The modified Hausdorff distance is no longer given from this function, but can easily be computed using output from this function as it is given by mhd(A,B) = max( e.bar(A,B), e.bar(B,A)), and in some literature the maximum is replaced by the minimum. See, e.g., Baddeley, (1992) and Schwedler and Baldwin (2011).

For computational efficiency, the distance transform method is used via `distmap` from package **spatstat** for calculating d(x,A) x in the raster.

## Value

`locperf` returns a list object with components depending on `which.stats`: one or more of the following, each of which is a single numeric, except as indicated.

`bdelta` matrix or numeric depending on p and number of thresholds.

haus — numeric giving the Hausdorff distances for each threshold.

qdmapdiff — matrix or numeric, depending on k and number of thresholds, giving the value of the quantile (or k-th highest value) of the difference in distance maps for each threshold.

medMiss, medFalseAlarm, msdMiss, msdFalseAlarm — numeric giving the value of the mean error/square error distance for each threshold.

fom — matrix or numeric, depending on alpha and number of thresholds, giving the value of Pratt his Figure of Merit for each threshold.

minsep — numeric giving the value of the minimum boundary separation distance for each threshold.

distob returns a single numeric.

distmapfun returns a matrix of same dimension as the input argument's field.

## Author(s)

Eric Gilleland

## References

Abdou, I. E. and Pratt, W. K. (1979) Quantitative design and evaluation of enhancement/thresholding edge detectors. *Proc. IEEE*, **67**, 753–763.

Baddeley, A. (1992) An error metric for binary images. In *Robust Computer Vision Algorithms*, W. Forstner and S. Ruwiedel, Eds., Wichmann, 59–78.

Peli, T. and Malah, D. (1982) A study on edge detection algorithms. *Computer Graphics and Image Processing*, **20**, 1–21.

Pratt, W. K. (1977) *Digital Image Processing*. John Wiley and Sons, New York.

Schwedler, B. R. J. and Baldwin, M. E. (2011) Diagnosing the sensitivity of binary image measures to bias, location, and event frequency within a forecast verification framework. *Wea. Forecasting*, **26**, 1032–1044, doi:10.1175/WAF-D-11-00032.1.

Zhu, M., Lakshmanan, V. Zhang, P. Hong, Y. Cheng, K. and Chen, S. (2011) Spatial verification using a true metric. *Atmos. Res.*, **102**, 408–419, doi:10.1016/j.atmosres.2011.09.004.

## See Also

distmap, solutionset, im, boundingbox, as.rectangle, metrV, locmeasures2d

## Examples

```
x <- y <- matrix( 0, 10, 12)
x[2,3] <- 1
y[4,7] <- 1
x <- im( x)
y <- im( y)
x <- solutionset( x > 0)
y <- solutionset( y > 0)
```

```
locperf( x, y)

# Note that ph is NA because there is only 1 event.
# need to have at least k events if k > 1.

par( mfrow=c(1,2))
image.plot( distmapfun(x))
image.plot( distmapfun(y))
```

---

LocSig *Temporal Block Bootstrap Keeping Locations in Space Constant*

---

## Description

Temporal block bootstrap for data at spatial locations (holding locations constant at each iteration). This is a wrapper function to the tsboot or boot functions for use with the field significance approach of Elmore et al. (2006).

## Usage

```
LocSig(Z, numrep = 1000, block.length = NULL, bootfun = "mean",
    alpha = 0.05, bca = FALSE, ...)

## S3 method for class 'LocSig'
plot(x, loc = NULL, nx = NULL, ny = NULL, ...)
```

## Arguments

| | |
|---|---|
| Z | n by m numeric matrix whose rows represent contiguous time points, and whose columns represent spatial locations. |
| numrep | numeric/integer giving the number of bootstrap replications to use. |
| block.length | positive numeric/integer giving the desired block lengths. If NULL, floor(sqrt(n)) is used. If 1, then the IID bootstrap is performed, and the BCa method may be used to find CI's, if bca is TRUE. |
| bootfun | character naming an R function to be applied to each replicate sample. Must return a single number, but is otherwise the statistic argument for function tsboot (or boot if block.length = 1). |
| alpha | numeric giving the value of alpha to obtain (1-alpha)*100 percent CI's for bootfun. |
| bca | logical, should bias-corrected and adjusted (BCa) CI's be calculated? Only used if block.length = 1. Will give a warning if this argument is TRUE, and block.length > 1, and will use the percentile method. |
| x | data frame of class "LocSig" as returned by LocSig. |
| loc | m by 2 matrix of location coordinates. |

| | |
|---|---|
| nx,ny | If loc is NULL, then nx and ny must be supplied. These give the number of rows and columns of a grid to make an image (using as.image) for plotting. If these are used, the data Z must be from a regular grid of points. |
| ... | LocSig: optional additional arguments to the tsboot (or boot if block.length=1) function. plot.LocSig: optional additional arguments to image.plot. |

## Details

This function performs the circular block bootstrap algorithm over time at each of m locations (columns of x). So, at each bootstrap iteration, entire blocks of rows of x are resampled with replacement. If Z represents forecast errors at grid points, and bootfun="mean", then this finds the grid-point CI's in steps 1 (a) to 1 (c) of Elmore et al. (2006).

## Value

LocSig: A data frame with class attribute "LocSig" with components:

| | |
|---|---|
| Estimate | numeric giving the estimated values of bootfun (the statistic for which CI's are computed). |
| Lower, Upper | numeric giving the estimated lower (upper) (1-alpha)*100 percent CI's. |

plot.LocSig: invisibly returns a list containing the estimate as returned by LocSig, and the confidence range.

## Author(s)

Eric Gilleland

## References

Elmore, K. L., Baldwin, M. E. and Schultz, D. M. (2006) Field significance revisited: Spatial bias errors in forecasts as applied to the Eta model. *Mon. Wea. Rev.*, **134**, 519–531.

## See Also

spatbiasFS, tsboot, boot, boot.ci, MCdof, sig.cor.t, sig.cor.Z, cor.test, image.plot, as.image

## Examples

```
## Not run:
data( "GFSNAMfcstEx" )
data( "GFSNAMobsEx" )
data( "GFSNAMlocEx" )

id <- GFSNAMlocEx[,"Lon"] >=-90 & GFSNAMlocEx[,"Lon"] <= -75 & GFSNAMlocEx[,"Lat"] <= 40

look <- LocSig(GFSNAMfcstEx[,id] - GFSNAMobsEx[,id], numrep=500)

stats(look)
```

```
plot(look, loc = GFSNAMlocEx[ id, ] )

## End(Not run)
```

---

lossdiff *Test for Equal Predictive Ability on Average Over a Regularly Gridded Space*

---

## Description

Test for equal predictive ability (for two forecast models) on average over a regularly gridded space using the method of Hering and Genton (2011).

## Usage

```
lossdiff(x, ...)

## Default S3 method:
lossdiff(x, ..., xhat1, xhat2, threshold = NULL,
    lossfun = "corrskill", loc = NULL, zero.out = FALSE)

## S3 method for class 'SpatialVx'
lossdiff(x, ..., time.point = 1, obs = 1, model = c(1, 2),
    threshold = NULL, lossfun = "corrskill", zero.out = FALSE)

empiricalVG.lossdiff( x, trend = 0, maxrad, dx = 1, dy = 1 )

flossdiff(object, vgmodel = "expvg", ...)

## S3 method for class 'lossdiff'
summary(object, ...)

## S3 method for class 'lossdiff'
plot(x, ..., icol = c("gray", tim.colors(64)))

## S3 method for class 'lossdiff'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x,xhat1, xhat2 | lossdiff: m by n matrices defining the (gridded) verification set where xhat1 and xhat2 are the two forecast models being compared. plot.lossdiff: x is a list returned by lossdiff. |
| object | flossdiff this is the output returned by lossdiff. summary.lossdiff: list object returned by lossdiff or flossdiff. |

| | |
|---|---|
| threshold | numeric vector of length one, two or three giving a threshold under which (non-inclusive) all values will be set to zero. If length is one, the same threshold is used for all fields (observed, and both models). If length is two, the same threshold will be used for both models (the second value of threshold). Otherwise, the first entry is used for the observed field, the second for the first model and the third for the second model. |
| lossfun | character anming a loss function to use in finding the loss differential for the fields. Default is to use correlation as the loss function. Must have arguments x and y, and may have any additional arguments. |
| trend | a matrix (of appropriate dimension) or single numeric (if constant trend) giving the value of the spatial trend. the value is simply subtracted from the loss differential field before finding the empirical variogram. If zero.out is TRUE, then wherever the original three fields all had zero-valued grid points are returned back to zero before continuing (hence ignored in the computation of the variogram). |
| loc | (optional) mn by 2 matrix giving location coordinates for each grid point. If NULL, they are taken to be the grid expansion of the dimension of x (i.e., cbind(rep(1:dim(x)[1],dim(x)[2]), rep(1:dim(x)[2],each=dim(x)[1]))). This argument is not used by lossdiff, but may be used subsequently by the plot method function. |
| maxrad | numeric giving the maximum radius for finding variogram differences per the R argument of vgram.matrix. |
| dx, dy | dx and dy of vgram.matrix. |
| zero.out | logical, should the variogram be computed only over non-zero values of the process? If TRUE, a modified version of vgram.matrix is used (variogram.matrix). |
| vgmodel | character string naming a variogram model function to use. Default is the exponential variogram, expvg. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| icol | (optional) color scheme. |
| ... | lossdiff: optional additional arguments to lossfun. Not used by the summary or plot functions. |

## Details

Hering and Genton (2011) introduce a test procedure for comparing spatial fields, which is based on a time series test introduced by Diebold and Mariano (1995). First, a loss function, g(x,y), is calculated, which can be any appropriate loss function. This is calculated for each of two forecast fields. The loss differential field is then given by:

D(s) = g(x(s),y1(s)) - g(x(s),y2(s)), where s are the spatial locations, x is the verification field, and y1 and y2 are the two forecast fields.

It is assumed that D(s) = phi(s) + psi(s), where phi(s) is the mean trend and psi(s) is a mean zero stationary process with unknown covariance function C(h) = cov(psi(s),psi(s+h)). In particular, the argument trend represents phi(s), and the default is that the mean is equal (and zero) over the entire

domain. If it is believed that this is not the case, then it should be removed before finding the covariance.

To estimate the trend, see e.g. Hering and Genton (2011) and references therein.

A test is constructed to test the null hypothesis of equal predictive ability on average. That is,

H_0: 1/|D| int_D E[D(s)]ds = 0, where |D| is the area of the domain,

The test statistic is given by

S_V = mean(D(s))/sqrt(mean(C(h))),

where C(h) = gamma(infinity|p) - gamma(h|p) is a fitted covariance function for the loss differential field. The test statistic is assumed to be N(0,1) so that if the p-value is smaller than the desired level of significance, the null hypothesis is not accepted.

For 'flossdiff', an exponential variogram is used. Specifically,

gamma(h | theta=(s,r)) = s^2*(1 - exp(-h/r)),

where s is sqrt(sill) and r is the range (nugget effects are not accounted for here). If `flossdiff` should fail, and the empirical variogram appears to be reasonable (e.g., use the `plot` method function on `lossdiff` output to check that the empirical variogram is concave), then try giving alternative starting values for the `nls` function by using the `start.list` argument. The default is to use the variogram value for the shortest separation distance as an initial estimate for s, and `maxrad` as the initial estimate for r.

Currently, it is not possible to fit other variogram models with this function. Such flexibility may possibly be added in a future release. In the meantime, use `flossdiff` as a template to make your own similar function; just be sure to return an object of class "nls", and it should work seamlessly with the `plot` and `summary` method functions for a "lossdiff" object. For example, if it is desired to include the nugget or an extra factor (e.g., 3 as used in Hering and Genton, 2011), then a new similar function would need to be created.

Also, although the testing procedure can be applied to irregularly spaced locations (non-gridded), this function is set up only for gridded fields in order to take advantage of computational efficiencies (i.e., use of vgram.matrix), as these are the types of verification sets in mind for this package. For irregularly spaced grids, the function `spct` can be used.

The above test assumes constant spatial trend. It is possible to remove any spatial trend in D(s) before applying the test.

The procedure requires four steps (hence four functions). The first is to calculate the loss differential field using `lossdiff`. Next, calculate the empirical variogram of the loss differential field using `empiricalVG.lossdiff`. This second step was originally included within the first step in `lossdiff`, but that setup presented a problem for determining if a spatial trend exists or not. It is important to determine if a trend exists, and if so, to (with care) estimate the trend, and remove it. If a trend is detected (and estimated), it can be removed before calling `empiricalVG.lossdiff` (then use the default `trend = 0`), or it can be passed in via the `trend` argument; the advantage (or disadvantage) of which is that the trend term will be included in the output object. The third step is to fit a parametric variogram model to the empirical one using `flossdiff`. The final, fourth step, is to conduct the test, which is performed by the `summary` function.

In each step, different aspects of the model assumptions can be checked. For example, isotropy can be checked by the plot in the lower right panel of the result of the `plot` method function after having called `empiricalVG.lossdiff`. The function `nlminb` is used to fit the variogram model.

For application to precipitation fields, and introduction to the image warp (coming soon) and distance map loss functions, see Gilleland (2013).

## Value

A list object is returned with possible components:

| | |
|---|---|
| `data.name` | character vector naming the fields under comparison |
| `lossfun,lossfun.args,vgram.args` | same as the arguments input to the lossdiff function. |
| `d` | m by n matrix giving the loss differential field, D(s). |
| `trend.fit` | An OLS trend fitting the locations to the field via lm. |
| `loc` | the self-same value as the argument passed in, or if NULL, it is the expanded grid coordinates. |

empiricalVG.lossdiff returns all of the above (carried over) along with

| | |
|---|---|
| `lossdiff.vgram` | list object as returned by vgram.matrix |
| `trend` | it is the self-same as the value passed in. |

flossdiff returns all of the above plus:

| | |
|---|---|
| `vgmodel` | list object as returned by nls containing the fitted exponential variogram model where s is the estimate of sqrt(sill), and r of the range parameter (assuming 'flossdiff' was used to fit the variogram model). |

summary.lossdiff invisibly returns the same list object as above with additional components:

| | |
|---|---|
| `Dbar` | the estimated mean loss differential (over the entire field). |
| `test.statistic` | the test statistic. |
| `p.value` | list object with components: two.sided–the two-sided alternative hypothesis–, less–the one-sided alternative hypothesis that the true value mu(D) < 0–and greater–the one-sided alternative hypothesis that mu(D) > 0–, p-values under the assumption of standard normality of the test statistic. |

## Author(s)

Eric Gilleland

## References

Diebold, F. X. and Mariano, R. S. (1995) Comparing predictive accuracy. *Journal of Business and Economic Statistics*, **13**, 253–263.

Gilleland, E. (2013) Testing competing precipitation forecasts accurately and efficiently: The spatial prediction comparison test. *Mon. Wea. Rev.*, **141**, (1), 340–355.

Hering, A. S. and Genton, M. G. (2011) Comparing spatial predictions. *Technometrics* **53**, (4), 414–425.

## See Also

vgram.matrix, nls, corrskill, abserrloss, sqerrloss, distmaploss

## Examples

```
grid<- list( x = seq( 0, 5,, 25), y = seq(0,5,,25) )
obj<-Exp.image.cov( grid = grid, theta = .5, setup = TRUE)

look<- sim.rf( obj )
look[ look < 0 ] <- 0
look <- zapsmall( look )

look2 <- sim.rf( obj ) * .25
look2[ look2 < 0 ] <- 0
look2 <- zapsmall( look2 )

look3 <- sim.rf( obj) * 2 + 5
look3[ look3 < 0 ] <- 0
look3 <- zapsmall( look3 )

res <- lossdiff( x = look, xhat1 = look2, xhat2 = look3, lossfun = "abserrloss" )
res <- empiricalVG.lossdiff( res, maxrad = 8 )
res <- flossdiff( res )
res <- summary( res )

plot( res )

## Not run:
data( "pert000" )
data( "pert004" )
data( "pert006" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, list( pert004, pert006 ), loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = c( "pert004", "pert006" ) )

look <- lossdiff( hold, lossfun = "abserrloss" )
look <- empiricalVG.lossdiff( look, maxrad = 8 )
look <- flossdiff( look )

plot( look )
summary( look )

## End(Not run)
```

---

make.SpatialVx *Spatial Verification Sets – SpatialVx Object*

---

## Description

A list object containing the verification sets of spatial verification and forecast fields with pertinent information.

## Usage

```
make.SpatialVx(X, Xhat, thresholds = NULL, loc = NULL, projection =
                 FALSE, subset = NULL, time.vals = NULL, reg.grid =
                 TRUE, map = FALSE, loc.byrow = FALSE, field.type = "",
                 units = "", data.name = "", obs.name = "X", model.name
                 = "Xhat", q = c(0, 0.1, 0.25, 0.33, 0.5, 0.66, 0.75,
                 0.9, 0.95), qs = NULL)

## S3 method for class 'SpatialVx'
hist(x, ..., time.point = 1, obs = 1, model = 1,
                 threshold.num = NULL)

## S3 method for class 'SpatialVx'
plot( x, ..., time.point = 1, obs = 1, model = 1,
    col = c( "gray", tim.colors( 64 ) ), zlim, mfrow = c(1, 2) )

## S3 method for class 'SpatialVx'
print(x, ...)

## S3 method for class 'SpatialVx'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| X | An n X m matrix or n X m X T array giving the verification field of interest. If an array, T is the number of time points. |
| Xhat | An n X m matrix or n X m X T array giving the forecast field of interest, or a list of such matrices/arrays with each component of the list an n X m matrix or n X m X T array defining a separate forecast model. |
| thresholds | single numeric, numeric vector, or Nu X Nf matrix, where Nu are the number of thresholds and Nf the number of forecast models plus one (for the verification) giving the threshold values of interest for the verification set or components of the set. If NULL (default), then thresholds will be calculated as the quantiles (defined through argument q) of each field. If a single numeric or a numeric vector, then an n X 2 matrix will be created (with column names "X" and "Xhat" where each column is identical. Otherwise, different thresholds may be applied to each of the verification and forecast fields. For example, if quantiles are used for thresholds, then each field will have their own unique thresholds. |
| loc | If lon/lat coordinates are available, then this is an n * m X 2 matrix giving the lon/lat coordinates of each grid point or location. Should follow the convention used by the **maps** package. |

`projection` logical, are the grids projections onto the globe? If so, when plotting, it will be attempted to account for this by using the `poly.image` function from package `fields`. In this case, each column of `loc` will be converted to a matrix using `byrow` equal to the value of `loc.byrow`.

`subset` vector identifying which specific grid points should be included (if not all of them). This argument may be ignored by most functions and is included for possible future functionality.

`time.vals` If more than one time point is available in the set (i.e., the set is of n X m X T arrays, with T > 1), then this argument can be used to define the time points. If missing, the default will yield the vector `1:T`. But, it is possible to include actual time information. This is also a forward looking feature that may or may not have any subsequent functionality.

`reg.grid` logical, is the verification set on a regular grid? This is another feature intended for possible future functionality. Most functions in this package assume the set is on a regular grid.

`map` logical, should the plot function attempt to place a map onto the plot? Only possible if the `loc` argument is given.

`field.type, units`
character used for plot labelling and printing information to the screen. Describes what variable and in what units the field represents.

`data.name, obs.name, model.name`
character vector describing the verification set overall, the observation(s) and the model(s), resp.

`q` numeric vector giving the values of quantiles to be used for thresholds. Only used if `thresholds` is NULL.

`qs` character vector describing the quantiles used. Again, only used if `thresholds` is NULL. This is for subsequent plot/print labelling purposes.

`time.point` numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. May also be a function name, in which case the function is applied at each grid point individually across time.

`obs, model` numeric indicating which observation/forecast model to select for the analysis.

`col, zlim` optional arguments to `image`, and/or **fields** functions `poly.image` and `image.plot`

`mfrow` optional argument to change the mfrow argument for the graphic device. Default is one row with two plots (obs and model). If null, then the mfrow argument will not be changed.

`x, object` list object of class "SpatialVx".

`loc.byrow` logical determining whether to set up the location matrices using `byrow` = TRUE or FALSE (for use with `poly.image`.

`threshold.num` If not null, then the threshold index to apply a threshold to the fields before creating the histogram.

`...` `hist` method: optional arguments to `hist`.
`plot` method: if `time.point` is a function, then these allow for optional arguments to this function to be passed.
`print` and `summary` methods: Not used.

## Details

This function merely describes a spatial verification set that includes the actual data as well as numerous attributes that are used by several of the subsequent functions that might be employed. In many cases, the attribute information may be passed on to output from other functions for plot labelling and printing purposes (e.g., in order to identify the verification set, time point(s), etc.).

All (or perhaps most) subsequent functions in this package utilize objects of this class and the information contained in the attributes. This function simply gathers information and data sets into a particular form.

The plot method function attempts to create an image plot of each field in the set (at each time point). If projection is TRUE, then it will attempt to preserve the projection (via `poly.image` of package **fields**). It will also add white contour lines showing the thresholds. If map is TRUE and `loc` was supplied, then a map will also be added, if possible.

## Value

A list object with two (unnamed) components:

| | |
|---|---|
| `1` | matrix or array (same as input argument) giving the observation |
| `2` | Either a matrix or array (same as input argument) or a list of such objects if more than one forecast model. |

Several attributes are also included among the following:

| | |
|---|---|
| `xdim` | numeric of length 2 or 3 giving the dimensions of the verification set (i.e., m, n and T, if relevant). |
| `time` | vector giving the time values |
| `thresholds` | matrix giving the thresholds for each field. If there is more than one forecast, and they use the same threshold, this matrix may have only two columns. |
| `udim` | the dimensions of the thresholds matrix. |
| `loc` | nm X 2 matrix giving the locations. If loc was not given, this will be c(rep(1:n, m), rep(1:m, each=n)). |
| `subset` | If given, this is a numeric vector describing a subset of loc to be used. |
| `data.name, obs.name, model.name` | character vector giving the names of the data sets (same as input arguments). |
| `nforecast` | single numeric giving the number of different forecast models contained in the object. |
| `field.type, units` | character strings, same as input arguments. |
| `projection` | logical, is the grid a projection? |
| `reg.grid` | logical, is the grid a regular grid? |
| `map` | logical, should a map be added to image plots of the data? |
| `qs` | character vector giving the names of the threshold quantiles. |
| `msg` | A message involving the data name, field type and units for adding info to plots, etc. |

## Author(s)

Eric Gilleland

## See Also

hoods2d, poly.image

## Examples

```
data( "UKobs6" )
data( "UKfcst6" )
data( "UKloc" )

hold <- make.SpatialVx( UKobs6, UKfcst6, thresholds = c(0.01, 20.01),
    loc = UKloc, field.type = "Precipitation", units = "mm/h",
    data.name = "Nimrod", obs.name = "Observations 6", model.name = "Forecast 6",
    map = TRUE)

hold

plot( hold )

hist( hold )

hist( hold, threshold.num = 2 )

## Not run:
# Stage 2 Observation files from
# NSSL/NCEP Spring 2005 Forecast Experiment.
data( "obs0426" )
data( "obs0513" )
data( "obs0514" )
data( "obs0518" )
data( "obs0519" )
data( "obs0525" )
data( "obs0601" )
data( "obs0603" )
data( "obs0604" )

st2 <- array(c(c(obs0426), c(obs0513), c(obs0514), c(obs0518),
    c(obs0519), c(obs0525), c(obs0601), c(obs0603), c(obs0604)),
    dim=c(601, 501, 9))

rm(obs0426, obs0513, obs0514, obs0518, obs0519, obs0525, obs0601,
    obs0603, obs0604)

# wrf2caps
data( "wrf2caps0425" )
data( "wrf2caps0512" )
data( "wrf2caps0513" )
data( "wrf2caps0517" )
data( "wrf2caps0518" )
```

```
data( "wrf2caps0524" )
data( "wrf2caps0531" )
data( "wrf2caps0602" )
data( "wrf2caps0603" )

# wrf4ncar
data( "wrf4ncar0425" )
data( "wrf4ncar0512" )
data( "wrf4ncar0513" )
data( "wrf4ncar0517" )
data( "wrf4ncar0518" )
data( "wrf4ncar0524" )
data( "wrf4ncar0531" )
data( "wrf4ncar0602" )
data( "wrf4ncar0603" )

# wrf4ncep
data( "wrf4ncep0425" )
data( "wrf4ncep0512" )
data( "wrf4ncep0513" )
data( "wrf4ncep0517" )
data( "wrf4ncep0518" )
data( "wrf4ncep0524" )
data( "wrf4ncep0531" )
data( "wrf4ncep0602" )
data( "wrf4ncep0603" )

wrf2caps <- array(c(c(wrf2caps0425), c(wrf2caps0512),
    c(wrf2caps0513), c(wrf2caps0517), c(wrf2caps0518),
    c(wrf2caps0524), c(wrf2caps0531), c(wrf2caps0602),
    c(wrf2caps0603)),
    dim=c(601, 501, 9))

wrf4ncar <- array(c(c(wrf4ncar0425), c(wrf4ncar0512),
    c(wrf4ncar0513), c(wrf4ncar0517), c(wrf4ncar0518),
    c(wrf4ncar0524), c(wrf4ncar0531), c(wrf4ncar0602),
    c(wrf4ncar0603)),
    dim=c(601, 501, 9))

wrf4ncep <- array(c(c(wrf4ncep0425), c(wrf4ncep0512),
    c(wrf4ncep0513), c(wrf4ncep0517), c(wrf4ncep0518),
    c(wrf4ncep0524), c(wrf4ncep0531), c(wrf4ncep0602),
    c(wrf4ncep0603)),
    dim=c(601, 501, 9))

rm(wrf2caps0425, wrf2caps0512, wrf2caps0513, wrf2caps0517,
    wrf2caps0518, wrf2caps0524, wrf2caps0531, wrf2caps0602,
    wrf2caps0603, wrf4ncar0425, wrf4ncar0512, wrf4ncar0513,
    wrf4ncar0517, wrf4ncar0518, wrf4ncar0524, wrf4ncar0531,
    wrf4ncar0602, wrf4ncar0603, wrf4ncep0425, wrf4ncep0512,
    wrf4ncep0513, wrf4ncep0517, wrf4ncep0518, wrf4ncep0524,
    wrf4ncep0531, wrf4ncep0602, wrf4ncep0603)
```

```
fcst <- list(wrf2caps, wrf4ncar, wrf4ncep)

rm(wrf2caps, wrf4ncar, wrf4ncep)

# Now, create the object.
data( "ICPg240Locs" )

ICPreal <- make.SpatialVx( st2, fcst, thresholds = c(0.1, 20.1),
    loc = ICPg240Locs, projection = TRUE, loc.byrow = TRUE,
    time.vals = c(2005042600, 2005051300, 2005051400, 2005051800,
        2005051900, 2005052500, 2005060100, 2005060300, 2005060400),
    map = TRUE, field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Real Test Cases",
    obs.name = "Stage II Analysis",
    model.name = c( "WRF 2 CAPS", "WRF 4 NCAR", "WRF 4 NCEP"))

plot( ICPreal )

plot( ICPreal, time.point = mean )

plot( ICPreal, model = 2 )

plot( ICPreal, model = 3 )

plot( ICPreal, time.point = 2, model = 2 )


## End(Not run)
```

---

MCdof *Monte Carlo Degrees of Freedom*

---

## Description

Estimate the distribution of the proportion of spatial locations that contain significant correlations with randomly generated data along the lines of Livezey and Chen (1983).

## Usage

```
MCdof(x, ntrials = 5000, field.sig = 0.05, zfun = "rnorm", zfun.args = NULL,
      which.test = c("t", "Z", "cor.test"), verbose = FALSE, ...)

sig.cor.t(r, len = 40, ...)

sig.cor.Z(r, len = 40, H0 = 0)

fisherz(r)
```

## Arguments

| | |
|---|---|
| x | n by m numeric matrix whose rows represent temporal points, and whose columns are spatial locations. |
| ntrials | numeric/integer giving the number of times to generate random samples of size n, and correlate them with the columns of x. |
| field.sig | numeric between 0 and 1 giving the desired fields significance level. |
| zfun | character naming a random number generator that takes n (the size of the sample to be drawn) as an argument, and any other arguments necessary. |
| zfun.args | list object giving the values for additional arguments to the function named by zfun. |
| which.test | character naming which type of test to do (default, "t", is a t-test, calls sig.cor.t). "Z" does Fisher's Z transform (calls sig.cor.Z). "cor.test" calls cor.test giving more options, but is also considerably slower than "t" or "Z". |
| r | numeric giving the correlation value(s). |
| len | numeric giving the size of the data for the test. |
| H0 | numeric giving the null hypothesis value (not used by MCdof). |
| verbose | logical, should progress information (including total run time) be printed to the screen? |
| ... | optional arguments to sig.cor.t (not used), sig.cor.Z, or cor.test depending on argument which.test. |

## Details

This function does the Livezey and Chen (1983) Monte Carlo step 2 (a) from Elmore et al. (2006). It generates a random sample of size n, and finds the p-values of a correlation test with this random sample and each column of x. From this, it estimates the proportion of spatial locations that could contain significant bias purely by chance.

## Value

MCdof returns a list object with components:

| | |
|---|---|
| MCprops | numeric vector of length ntrials giving the proportion of locations with significant bias found by chance for each repition of the experiment. |
| minsigcov | single numeric giving the 1 - field.sig quantile of the resulting proportions given by MCprops. |

sig.cor.t and sig.cor.Z return umeric vectors of p-values, and fisherz returns a numeric vector of test statistics.

## Author(s)

Kimberly L. Elmore, Kim.Elmore "at" noaa.gov, and Eric Gilleland

## References

Elmore, K. L., Baldwin, M. E. and Schultz, D. M. (2006) Field significance revisited: Spatial bias errors in forecasts as applied to the Eta model. *Mon. Wea. Rev.*, **134**, 519–531.

Livezey, R. E. and Chen, W. Y. (1983) Statistical field significance and its determination by Monte Carlo techniques. *Mon. Wea. Rev.*, **111**, 46–59.

## See Also

spatbiasFS, LocSig, cor.test, rnorm, runif, rexp, rgamma

## Examples

```
data( "GFSNAMfcstEx" )
data( "GFSNAMobsEx" )
data( "GFSNAMlocEx" )

id <- GFSNAMlocEx[,"Lon"] >=-90 & GFSNAMlocEx[,"Lon"] <= -75 & GFSNAMlocEx[,"Lat"] <= 40

look <- MCdof(GFSNAMfcstEx[,id] - GFSNAMobsEx[,id], ntrials=500)

stats(look$MCprops)
look$minsigcov

fisherz( abs(cor(rnorm(10),rexp(10), use="pairwise.complete.obs")))
```

---

# MergeForce *Force Merges in Matched Feature Objects*

---

## Description

Force merges in matched feature objects so that, among other things, subsequent analyses are quicker and cleaner.

## Usage

```
MergeForce(x, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| x | list object of class "matched". |
| verbose | logical, should progress information be printed to the screen. |

## Details

Objects returned by functions such as `deltamm` and `centmatch` provide information necessary to merge and match features from "features" objects. In the case of `centmatch`, only implicit merges are given, and this function creates objects where the implicit merges are forced to be merged. In the case of `deltamm`, a second pass through might yield better merges/matches in that without a second pass, only features in one field or the other can be merged and matched (not both simultaneously). Using this function, and apssing the result back through `deltamm` can result in subsequent matches of merged features from both fields simultaneously. Moreover, in some cases, it may be more computationally efficient to run this function once for subsequent analyses/plotting.

## Value

A list object of class "matched" is returned containing several components and the same attributes as x.

| | |
|---|---|
| `match.message` | A character string stating how features were matched with (merged) apended. |
| `match.type` | character of length 2 naming the original matching function used and this function to note that the features have been forced to be merged/clustered together. |
| `matches` | two-column matrix with forecast object numbers in the first column and corresponding matched observed features in the second column. If no matches, this will have value integer(0) for each column giving a matrix with dimension 0 by 2. |
| `unmatched` | list with components X and Xhat giving the unmatched object numbers, if any, from the observed and forecast fields, resp. If none, the value will be integer(0). |

Note that all of the same list components of x are passed back, except for special information (which is usually no longer relevant) such as Q (deltamm), criteria, criteria.values, centroid.distances (centmatch)

Additionally, merges and/or implicit.merges (centmatch) are not included as they have been merged.

## Author(s)

Eric Gilleland

## See Also

For identifying features in a field: `FeatureFinder`

For merging and/or matching features: `deltamm`, `centmatch`, `plot.matched`

## Examples

```
x <- y <- matrix(0, 100, 100)
x[2:3,c(3:6, 8:10)] <- 1
y[c(4:7, 9:10),c(7:9, 11:12)] <- 1

x[30:50,45:65] <- 1
y[c(22:24, 99:100),c(50:52, 99:100)] <- 1
```

```
hold <- make.SpatialVx( x, y, field.type="contrived", units="none",
    data.name = "Example", obs.name = "x", model.name = "y" )

look <- FeatureFinder(hold, smoothpar=0.5)

look2 <- centmatch( look )

look2

look2 <- MergeForce( look2 )

look2

# plot( look2 )

## Not run:
look3 <- deltamm( look, N = 201 )
look3 <- MergeForce( look3 )

data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004,
    loc=ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units="mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- FeatureFinder(hold, smoothpar=10.5, thresh = 5)

look2 <- centmatch( look, verbose = TRUE )

look2 <- MergeForce( look2 )

plot( look2 )


## End(Not run)
```

---

metrV *Binary Location Metric Proposed in Zhu et al. (2011)*

---

## Description

Calculate the metric metrV proposed in Zhu et al (2011), which is a linear combination of the square root of the sum of squared error between two binary fields, and the mean error distance (Peli and Malah, 1982); or the difference in mean error distances between two forecast fields and the verification field, if the comparison is performed between two forecast models against the same verification field.

## Usage

```
metrV(x, ...)

## Default S3 method:
metrV(x, xhat, xhat2 = NULL, thresholds, lam1 = 0.5, lam2 = 0.5,
    distfun = "distmapfun", a = NULL, verbose = FALSE, ...)

## S3 method for class 'SpatialVx'
metrV(x, time.point = 1, obs = 1, model = 1, lam1 = 0.5, lam2 = 0.5,
    distfun = "distmapfun", verbose = FALSE, ...)

## S3 method for class 'metrV'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | Either a list object as returned by make.SpatialVx or a matrix representing a verificaiton grid. For the print method, this is an object returned by metrV. |
| xhat, xhat2 | (xhat2 is optional) matrix representing a forecast grid. |
| thresholds | q X 2 or q X 3 (if xhat is not NULL) matrix giving the thresholds to apply to the verification field (first column) and each forecast field. |
| lam1 | numeric giving the weight to be applied to the square root of the sum of squared errors of binary fields term in metrV. |
| lam2 | numeric giving the weight to be applied to the mean error distance term in metrV. |
| distfun | character naming a function with which to calculate the shortest distances between each point x in the grid and the set of events. Default is the Euclidean distance metric (see the help file for locperf for more information). |
| a | list object giving certain information about the verification set. These are the attributes of the "SpatailVx" object. May be used here to include information (as attributes of the returned object) that would otherwise not be available to the print method function. In particular, the components, msg, data.name and qs are printed if available. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. May have length one or two. If it has length two, the second value is taken to be the second forecast model (i.e., xhat2 in the call to metrV.default). |
| verbose | logical, should progress information be printed ot the screen. |
| ... | Optional arguments to the distfun function. |

## Details

The binary location metric proposed in Zhu et al. (2011) is a linear combination of two measures: the amount of overlap between events in two fields, given by distOV (simply the square root of sum of squared errors between two binary fields), and (if there are events in both fields) the mean

error distance described in Peli and Malah (1982); see also Baddeley (1992). The metric can be computed between a forecast field, M1, and the verificaiton field, V, or it can be compared between two foreast models M1 and M2 with reference to V. That is,

metrV(M1,M2) = lam1*distOV(I.M1,I.M2) + lam2*distDV(I.M1,I.M2),

where I.M1 (I.M2) is the binary field determined by M1 >= threshold (M2 >= threshold), distOV(I.M1,I.M2) = sqrt( sum( (I.M1 - I.M2)^2)), distDV(I.M1,I.M2) = abs(distob(I.V,I.M1) - distob(I.V,I.M2)), where distob(A,B) is the mean error distance between A and B, given by:

e(A,B) = 1/(N(A))*sqrt( sum( d(x,B)), where the summation is over all the points x corresponding to events in A, and d(x,B) is the minimum of the shortest distance from the point x to each point in B. e(A,B) is calculated by using the distance transform as calculated by the `distmap` function from package `spatstat` for computational efficiency.

Note that if there are no events in both fields, then by definition, the term distob(A,B) = 0, and if there are no events in one and only one of the two fields, then a large constant (here, the maximum dimension of the field), is returned. In this way, distob differs from the mean error distance described in Peli and Malah (1982).

If comparing between the verification field and one forecast model, then the distDV term simplifies to just distob(I.V,I.M1).

One final note is that Eq (6) that defines `distOV` in Zhu et al. (2011) is correct (or rather, what is used in the paper). It is not, as is stated below Eq (6) in Zhu et al. (2011) the root *mean* square error, but rather the root square error. This function computes Eq (6) as written.

## Value

list object of class "metrV" with components:

| | |
|---|---|
| `OvsM1` | k by 3 matrix whose rows represent thresholds and columns give the component distOV, distob and metrV between the verification field and the forecast model 1. |
| `OvsM2` | If object2 supplied, k by 3 matrix whose rows represent thresholds and columns give the component distOV, distob and metrV between the verification field and the forecast model 2. |
| `M1vsM2` | If object2 supplied, k by 3 matrix whose rows represent thresholds and columns give the component distOV, distob and metrV between model 1 and model 2. |

May also contain attributes as passed by either the a argument or the "SpatialVx" object.

## Author(s)

Eric Gilleland

## References

Baddeley, A. J. (1992) An error metric for binary images. In *Robust Computer Vision Algorithms*, W. Forstner and S. Ruwiedel, Eds., Wichmann, 59–78.

Peli, T. and Malah, D. (1982) A study on edge detection algorithms. *Computer Graphics and Image Processing*, **20**, 1–21.

Zhu, M., Lakshmanan, V. Zhang, P. Hong, Y. Cheng, K. and Chen, S. (2011) Spatial verification using a true metric. *Atmos. Res.*, **102**, 408–419, doi:10.1016/j.atmosres.2011.09.004.

## See Also

distob, distmap, im, solutionset, deltametric, locmeasures2d, make.SpatialVx

## Examples

```
A <- B <- B2 <- matrix( 0, 10, 12)
A[2,3] <- 3
B[4,7] <- 400
B2[10,12] <- 17
hold <- make.SpatialVx( A, list(B, B2), thresholds = c(0.1, 3.1, 500),
    field.type = "contrived", units = "none",
    data.name = "Example", obs.name = "A",
    model.name = c("B", "B2") )

metrV(hold)

metrV(hold, model = c(1,2) )

## Not run:
data( "pert000" )
data( "pert001" )
data( "ICPg240Locs" )

testobj <- make.SpatialVx( pert000, pert001, thresholds = 1e-8,
    projection = TRUE, map = TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert001" )

metrV(testobj)

# compare above to results in Fig. 3 (top right panel) of Zhu et al. (2011).

data( "geom000" )
data( "geom001" )

testobj <- make.SpatialVx( geom000, geom001, thresholds = 0,
    projection = TRUE, map = TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Geometric Cases", obs.name = "geom000",
    model.name = "geom001" )

metrV(testobj)

# compare above to results in Fig. 2 (top right panel)
# of Zhu et al. (2011).  Note that they differ wildly.
# Perhaps because an actual elliptical area is taken in
# the paper instead of finding the values from the fields
# themselves?

## End(Not run)
```

`Mij` *Raw Image Moments.*

## Description

Calculate the raw Hu image moment Mij.

## Usage

```
Mij(x, s, i = 0, j = 0)
```

## Arguments

| | |
|---|---|
| `x` | A matrix. |
| `s` | A two-column matrix giving the location coordinates. May be missing in which case they are assumed to be integers giving the row and column numbers. |
| `i,j` | Integer giving the moment order for each coordinate x and y, resp. |

## Details

The raw moment M_ij (Hu 1962) is calculated by

M_ij = sum(x^i * y^j * Im[i, j])

where x and y are the pixel coordinates and Im is the (image) matrix. Various useful properties of an image may be gleaned from certain moments. For example, the image area is given by M_00, and the image centroid is (M_10 / M_00, M_01 / M_00). The image orientation angle can also be derived.

## Value

A single numeric giving the desired moment is returned.

## Author(s)

Eric Gilleland

## References

Hu, M. K. (1962) Visual Pattern Recognition by Moment Invariants. *IRE Trans. Info. Theory*, **IT-8**, 179–187.

## See Also

`imomenter`

## Examples

```
data( "geom000" )

Mij( geom000 ) # area
```

---

`minboundmatch` *Minimum Boundary Separation Feature Matching*

---

## Description

Match identified features within a spatial verification set via their minimum boundary separation.

## Usage

```
minboundmatch(x, type = c("single", "multiple"), mindist = Inf, verbose = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class "features". |
| `type` | character string stating either "single" or "multiple". In the former case, each feature in one field will be matched to only one feature in the other, which will be taken to be the features who have the smallest minimum boundary separation. In the case of "multiple", the `mindist` argument should be set to something small enough so that not every feature will be matched to every other feature. Also, the `MergeForce` function may be useful in this case. |
| `mindist` | single numeric giving the minimum boundary separation distance (measured by grid squares) beyond which features should not be matched. |
| `verbose` | logical, should progress information be printed to the screen? |
| `...` | Optional arguments to the `distmap` function from package **spatstat**. |

## Details

the minimum boundary separation is calculated by first finding the distance map for every feature in the observed field, masking it by each feature in the forecast field, and then finding the minimum of the resulting masked distance map. If `type` is "single", then the features are matched by the smallest minimum boundary separation per feature in each field. If `type` is "multiple", then every feature is matched so long as their minimum boundary separation (measured in grid squares) is less than or equal to `mindist`.

## Value

A list object of class "matched" is returned. If the type argument is "multiple", then an implicite.merges component is included, which will work with the MergeForce function.

## Author(s)

Eric Gilleland

## See Also

deltamm, centmatch, MergeForce

## Examples

```
x <- y <- matrix(0, 100, 100)
x[2:3,c(3:6, 8:10)] <- 1
y[c(4:7, 9:10),c(7:9, 11:12)] <- 1

x[30:50,45:65] <- 1
y[c(22:24, 99:100),c(50:52, 99:100)] <- 1

hold <- make.SpatialVx( x, y, field.type = "contrived", units = "none",
    data.name = "Example", obs.name = "x", model.name = "y" )

look <- FeatureFinder(hold, smoothpar=0.5)

look2 <- minboundmatch( look )

look2 <- MergeForce( look2 )

par( mfrow = c(1,2) )
plot( look2 )

look3 <- minboundmatch( look, type = "multiple", mindist = 50 )
look3 <- MergeForce( look2 )
plot( look3 )

look4 <- minboundmatch( look, type = "multiple", mindist = 20 )
look4 <- MergeForce( look4 )
plot( look4 )
```

---

# obs0426 — *Spatial Forecast Verification Methods Inter-Comparison Project (ICP) Test Cases and other example verification sets*

---

## Description

Test cases used for the ICP. In particular, those actually analyzed in the special collection of the journal, Weather and Forecasting. Includes the nine "real" cases, five simple geometric cases, and the seven perturbed "real" cases.

## Usage

```
data(obs0426)
data(obs0513)
```

```
data(obs0514)
data(obs0518)
data(obs0519)
data(obs0525)
data(obs0601)
data(obs0603)
data(obs0604)

data(wrf2caps0425)
data(wrf2caps0512)
data(wrf2caps0513)
data(wrf2caps0517)
data(wrf2caps0518)
data(wrf2caps0524)
data(wrf2caps0531)
data(wrf2caps0602)
data(wrf2caps0603)

data(wrf4ncar0425)
data(wrf4ncar0512)
data(wrf4ncar0513)
data(wrf4ncar0517)
data(wrf4ncar0518)
data(wrf4ncar0524)
data(wrf4ncar0531)
data(wrf4ncar0602)
data(wrf4ncar0603)

data(wrf4ncep0425)
data(wrf4ncep0512)
data(wrf4ncep0513)
data(wrf4ncep0517)
data(wrf4ncep0518)
data(wrf4ncep0524)
data(wrf4ncep0531)
data(wrf4ncep0602)
data(wrf4ncep0603)

data(geom000)
data(geom001)
data(geom002)
data(geom003)
data(geom004)
data(geom005)

data(pert000)
data(pert001)
data(pert002)
```

```
data(pert003)
data(pert004)
data(pert005)
data(pert006)
data(pert007)

data(ICPg240Locs)
```

## Format

The format is: num [1:601, 1:501] 0 0 0 0 0 0 0 0 0 0 ...

The format is: num [1:301101, 1:2] -110 -110 -110 -110 -110 ... - attr(*, "dimnames")=List of 2 ..$ : NULL ..$ : chr [1:2] "lon" "lat"

## Details

The nine "real" cases are forecast model output from three different versions of the Weather Research Forecast (WRF) model denoted `wrf2caps`, `wrf4ncar` and `wrf4ncep` (see Kain et al. 2008; Ahijevych et al., 2009 for complete details), and the corresponding "observed" fields are stage II reanalyses denoted here by "obs". The models are 24-h forecasts so that the valid time is for the next day (e.g., `obs0426` corresponds with `wrf2caps0425`). The final four digits for the "real" cases give the month and day of the forecast/observation. These data were from the 2005 Spring Program of the Storm Prediction Center/National Severe Storms Laboratory (SPC/NSSL, cf. Weiss et al., 2005; Kain et al., 2008). Units for the real cases are in mm/h, and are on the NCEP g240 grid (~4-km resolution) with 601 X 501 grid points. Both SPC and NSSL should be cited as sources for these cases, as well as Weiss et al. (2005) and possibly also Kain et al. (2008). The data were made available to the ICP by M. E. Baldwin.

The five geometric cases are simple ellipses (each with two intensities) that are compared against the verification case (`geom000`) on the same NCEP g240 grid as the nine real cases. See Ahijevych et al. (2009) for complete details. Case `geom001` is exactly the same as `geom000`, but is displaced 50 grid points to the right (i.e., ~200 km too far east). Case `geom002` is also identical to `geom000`, but displaced 200 grid points to the right. case `geom003` is displaced 125 grid points to the right, and is also too big 9i.e., has a spatial extent, or coverage, bias). Case `geom004` is also displaced 125 grid points to the right, but also has a different orientation (note, however, that it is not a true rotation of `geom000`). Case `geom005` is displaced 125 grid points to the right, and has a huge spatial extent bias. This last case is also the only one that actually overlaps with `geom000`, and therefore may be regarded by some as the best case. It is certainly the case that comes out on top by the traditional verification statistics that are calculated on a grid point by grid point basis. Ahijevych et al. (2009) should be cited if these geometric cases are used for publications, etc.

The seven perturbed cases (dented here by `pert00x`) are perturbations of `pert000`, which is adapted from `wrf2caps0531` (again, see Ahijevych et al, 2009 for more details). Case `pert001` shifts `pert000` three grid points to the right, and five grid points down (i.e., ~12 km to the east and ~20 km to the south). Case `pert002` is a shift six points to the right, and ten points down. Case `pert003` is a shift 12 points to the right and 20 points down. Case 4 is a shift 24 points to the right, and 40 points down. Case `pert005` is a shift 48 points to the right and 80 points down. Case `pert006` is a shift 12 points to the right, and 20 points down, and the entire field is multiplied by 1.5. Case `pert007` has the same spatial displacements as `pert003` and `pert006`, but also subtracts

1.27 from the entire field. Ahijevych et al. (2009) should be cited if these perturbed cases are used for publications, etc.

The longitude and latitude information for each grid (the NCEP g240 grid) is contained in the `ICPg240Locs` dataset.

These constitute all of the official data test cases used for the first round of the ICP, and in the special collection of papers for *Wea. Forecasting*. Other data sets for the ICP can be obtained from the ICP web site (http://www.ral.ucar.edu/projects/icp). Future data sets for the next round of the ICP will also be available there, and may potentially also be made available here.

## Source

http://www.ral.ucar.edu/projects/icp/

## References

Ahijevych, D., Gilleland, E., Brown, B. G. and Ebert, E. E. (2009) Application of spatial verification methods to idealized and NWP gridded precipitation forecasts. *Wea. Forecasting*, **24** (6), 1485–1497.

Kain, J. S., Weiss, S. J., Bright, D. R., Baldwin, M. E. Levit, J. J. Carbin, G. W. Schwartz, C. S. Weisman, M. L. Droegemeier, K. K. Weber, and D. B. Thomas, K. W. (2008) Some Practical Considerations Regarding Horizontal Resolution in the First Generation of Operational Convection-Allowing NWP. *Wea. Forecasting*, **23**, 931–952.

Weiss, S., Kain, J. Levit, J. Baldwin, M. E., Bright, D. Carbin, G. and Hart, J. (2005) NOAA Hazardous Weather Testbed. SPC/NSSL Spring Program 2005 Program Overview and Operations Plan. 61pp.

## Examples

```
## Not run:
data(obs0426)
data(wrf2caps0425)
data(wrf4ncar0425)
data(wrf4ncep0425)
data(ICPg240Locs)
## Plot verification sets with a map.
## Two different methods.

# First way does not preserve projections.
locr <- c( range( ICPg240Locs[,1]), range( ICPg240Locs[,2]))
zl <- range( c( c(obs0426), c( wrf2caps0425), c( wrf4ncar0425),
c( wrf4ncep0425)))
par( mfrow=c(2,2), mar=rep(0.1,4))
image( obs0426, axes=FALSE, col=c("grey", tim.colors(256)), zlim=zl)
par( usr=locr)
if( map.available) map( add=TRUE, database="state")
image( wrf2caps0425, axes=FALSE, col=c("grey", tim.colors(256)), zlim=zl)
par( usr=locr)
if( map.available) map( add=TRUE, database="state")
image( wrf4ncar0425, axes=FALSE, col=c("grey", tim.colors(256)), zlim=zl)
par( usr=locr)
```

```
if( map.available) map( add=TRUE, database="state")
image( wrf4ncep0425, axes=FALSE, col=c("grey", tim.colors(256)), zlim=zl)
par( usr=locr)
if( map.available) map( add=TRUE, database="state")
image.plot( obs0426, legend.only=TRUE, horizontal=TRUE,
col=c("grey", tim.colors(256)), zlim=zl)

# Second way preserves projections, but values are slighlty interpolated.
zl <- range( c( c(obs0426), c( wrf2caps0425), c( wrf4ncar0425),
               c( wrf4ncep0425)))
par( mfrow=c(2,2), mar=rep(2.1,4))
image(as.image(c(t(obs0426)), x=ICPg240Locs, nx=601, ny=501, na.rm=TRUE), zlim=zl,
      col=c("grey", tim.colors(64)), axes=FALSE, main="Stage II Reanalysis 4/26/05 0000 UTC")
map(add=TRUE, lwd=1.5)
map(add=TRUE, database="state", lty=2)
image(as.image(c(t(wrf2caps0425)), x=ICPg240Locs, nx=601, ny=501, na.rm=TRUE), zlim=zl,
col=c("grey", tim.colors(64)), axes=FALSE, main="WRF CAPS valid 4/26/05 0000 UTC")
map(add=TRUE, lwd=1.5)
map(add=TRUE, database="state", lty=2)
image(as.image(c(t(wrf4ncar0425)), x=ICPg240Locs, nx=601, ny=501, na.rm=TRUE), zlim=zl,
      col=c("grey", tim.colors(64)), axes=FALSE, main="WRF NCAR valid 4/26/05 0000 UTC")
map(add=TRUE, lwd=1.5)
map(add=TRUE, database="state", lty=2)
image(as.image(c(t(wrf4ncep0425)), x=ICPg240Locs, nx=601, ny=501, na.rm=TRUE), zlim=zl,
      col=c("grey", tim.colors(64)), axes=FALSE, main="WRF NCEP valid 4/26/05 0000 UTC")
map(add=TRUE, lwd=1.5)
map(add=TRUE, database="state", lty=2)
image.plot(obs0426, col=c("grey", tim.colors(64)), zlim=zl, legend.only=TRUE, horizontal=TRUE)

## End(Not run)
```

---

OF *Optical Flow Verification*

---

## Description

Perform verification using optical flow as described in Marzban and Sandgathe (2010).

## Usage

```
OF(x, ...)

## Default S3 method:
OF(x, ..., xhat, W = 5, grads.diff = 1, center = TRUE,
    cutoffpar = 4, verbose = FALSE)

## S3 method for class 'SpatialVx'
OF(x, ..., time.point = 1, obs = 1, model = 1, W = 5, grads.diff = 1,
    center = TRUE, cutoffpar = 4, verbose = FALSE)
```

```
## S3 method for class 'OF'
plot(x, ...)

## S3 method for class 'OF'
print(x, ...)

## S3 method for class 'OF'
hist(x, ...)

## S3 method for class 'OF'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x, xhat | Default: m by n matrices describing the verification and forecast fields, resp. The forecast field is considered the initial field that is morphed into the final (verification) field.<br>OF.SpatialVx: list object of class "SpatialVx".<br>plot, hist and print methods: list object as returned by OF. |
| object | list object as returned by OF. |
| W | numeric/integer giving the window size (should be no smaller than 5). |
| grads.diff | 1 or 2 describing whether to use first or second differences in finding the first derivative. |
| center | logical, should the fields be centered before performing the optical flow? |
| cutoffpar | numeric, set to NaN everything exceeding median +/- cutoffpar*sd. |
| verbose | logical, should progress information be printed to the screen? |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| ... | For OF: optional arguments to the optim function (cannot be par, fn, gr or method). See details section for plot and hist method functions. Not used by the summary method function. |

## Details

Estimates the optical flow of the forecast field into the verification field. Letting I_o(x,y) and I_f(x,y) represent the intensities of each field at coordinate (x,y), the collection of pairs (dx, dy) is the optical flow field, where:

I_o(x,y) ~ I_f(x,y) + [partial(I_f) wrt x]*dx + [partial(I_f) wrt y]*dy.

The procedure follows that proposed by Lucas and Kanade (1981) whereby for some window, W, it is assumed that all dx (dy) are assumed constant, and least squares estimation is used to estimate dx and dy (see Marzban and Sandgathe, 2010 for more on this implementation). This function iteratively calls optflow for each window in the field.

The above formulation is linear in the parameters. Marzban and Sandgathe (2010) also introduce an additive error component, which leads to a nonlinear version of the above. Namely,

I_o(x,y) ~ I_f(x,y) + [partial(I_f) wrt x]*dx + [partial(I_f) wrt y]*dy + A(x,y).

See Marzban and Sandgathe for more details.

The plot method function can produce a figure like that of Fig. 1, 5, and 6 in Marzban and Sandgathe (2010) or with option `full=TRUE`, even more plots. Optional arguments that may be passed in via the ellipses include: `full` (logical, produce a figure analogous to Fig. 1, 5 and 6 from Marzban and Sandgathe (2010) (FALSE/default) or make more plots (TRUE)), `scale` (default is 1 or no scaling, any numeric value by which the fields are divided/scaled before plotting), `of.scale` (default is 1, factor by which display vectors can be magnified), `of.step` (plot OF vectors every of.step, default is 4), `prop` (default is 2, value for `prop` argument in the call to `rose.diag` from package **CircStats**), `nbins` (default is 40, number of bins to use in the call to `rose.diag`).

The `hist` method function produces a two-dimensional histogram like that of Fig. 3 and 7 in Marzban and Sandgathe (2010). It can also take various arguments passed via the ellipses. They include: `xmin`, `xmax`, `ymin`, `ymax` (lower and upper bounds for the histogram breaks in the x- (angle) and y- (magnitude/displacement error) directions, resp. Defaults to (0,360) and (0,4)), nbreaks (default is 100, the number of breaks to use).

The `summary` method mostly uses the `stats` function from package **fields** to summarize results of the errors, but also uses `circ.summary` from package **CircStats** for the angular errors.

## Value

OF returns a list object of class "OF" with components:

| | |
|---|---|
| `data` | list with components x and xhat containing the data. |
| `data.name` | character vector giving the names of the verification and forecast fields. |
| `call` | object of class "call" giving the original function call. |
| `rows,cols` | numeric vector giving the rows and columns used for finding the centers of windows. Needed by the plot and hist method functions. |
| `err.add.lin` | m by n matrix giving the linear additive errors (intensities). |
| `err.mag.lin` | m by n matrix giving the linear magnitude (displacement) errors. |
| `err.ang.lin` | m by n matrix giving the linear angular errors. |
| `err.add.nlin,err.mag.nlin,err.ang.nlin` | same as above but for nonlinear errors. |
| `err.vc.lin,err.vr.lin,err.vc.nlin,err.vr.nlin` | m by n matrices giving the x- and y- direction movements for the linear and nonlinear cases, resp. |

The hist method function invisibly returns a list object of class "OF" that contains the same object that was passed in along with new components:

| | |
|---|---|
| `breaks` | a list with components x and y giving the breaks in each direction |
| `hist.vals` | itself a list with components xb, yb (the number of breaks -1 used for each direction), and nb (the histogram values for each break) |

The plot and summary mehtod functions do not return anything.

## Author(s)

Caren Marzban, marzban "at" u.washington.edu, with modifications by Eric Gilleland

## References

Lucas, B D. and Kanade, T. (1981) An iterative image registration technique with an application to stereo vision. *Proc. Imaging Understanding Workshop*, DARPA, 121–130.

Marzban, C. and Sandgathe, S. (2010) Optical flow for verification. *Wea. Forecasting*, **25**, 1479–1494, doi:10.1175/2010WAF2222351.1.

## See Also

optflow, optim, circ.summary

## Examples

```
## Not run:
data(hump)
initial <- hump$initial
final <- hump$final
look <- OF(final, xhat=initial, W=9, verbose=TRUE)
plot(look) # Compare with Fig. 1 in Marzban and Sandgathe (2010).
par(mfrow=c(1,1))
hist(look) # 2-d histogram.
plot(look, full=TRUE) # More plots.
summary(look)

# Another way to skin the cat.
hold <- make.SpatialVx( final, initial, field.type = "Bi-variate Gaussian",
    obs.name = "final", model.name = "initial" )

look2 <- OF(hold, W=9, verbose=TRUE)
plot(look2)
par(mfrow=c(1,1))
hist(look2)
plot(look2, full=TRUE)
summary(look2)

## End(Not run)
```

---

# optflow *Optical Flow*

---

## Description

Estimate the optical flow from one gridded field (image) to another.

## Usage

```
optflow(initial, final, grads.diff = 1, mean.field = NULL, ...)
```

## Arguments

| | |
|---|---|
| initial,final | m by n matrices where the optical flow is determined from initial (forecast) to final (observation). |
| grads.diff | either 1 or 2, where 1 calculates first derivatives with first differences and 2 first derivatives with second differences. |
| mean.field | Should they first be centered? If so, give the value for the centering here (usually the mean of initial). |
| ... | optional arguments to the optim function (cannot be par, fn, gr or method). |

## Details

This function estimates the optical flow from the initial field (image) to the final one as described in Marzban and Sandgathe (2010). Letting I_o(x,y) and I_f(x,y) represent the intensities of each field at coordinate (x,y), the collection of pairs (dx, dy) is the optical flow field, where:

I_o(x,y) ~ I_f(x,y) + [partial(I_f) wrt x]*dx + [partial(I_f) wrt y]*dy.

The procedure follows that proposed by Lucas and Kanade (1981) whereby for some window, W, it is assumed that all dx (dy) are assumed constant, and least squares estimation is used to estimate dx and dy (see Marzban and Sandgathe, 2010 for more on this implementation). It is assumed that the fields (initial and final) include only the window around the point of interest (i.e., this function finds the optical flow estimate for a single window). See the function OF, which iteratively calls this function, for performing optical flow over the entire field.

The above formulation is linear in the parameters. Marzban and Sandgathe (2010) also introduce an additive error component, which leads to a nonlinear version of the above. Namely,

I_o(x,y) ~ I_f(x,y) + [partial(I_f) wrt x]*dx + [partial(I_f) wrt y]*dy + A(x,y).

See Marzban and Sandgathe for more details.

## Value

numeric vector whose first three components are the optimized estimates (returned by the par component of optim) for the regression I_o(x,y) - I_f(x,y) = a0 + a1*[partial(I_f) wrt x] + a2*[partial(I_f) wrt y] (i.e., a1 and a2 are the estimates for dx and dy, resp.) and the latter three values are the initial estimates to optim as determined by linear regression (i.e., returned from the lm function).

## Author(s)

Caren Marzban, marzban "at" u.washington.edu, and modified by Eric Gilleland

## References

Lucas, B D. and Kanade, T. (1981) An iterative image registration technique with an application to stereo vision. *Proc. Imaging Understanding Workshop*, DARPA, 121–130.

Marzban, C. and Sandgathe, S. (2010) Optical flow for verification. *Wea. Forecasting*, **25**, 1479–1494, doi:10.1175/2010WAF2222351.1.

## See Also

OF, optim, lm

## Examples

```
x <- y <- matrix(0, 10, 10)
x[1:2,3:4] <- 1
y[3:4,5:6] <- 2

optflow(x,y)

## Not run:
initial <- hump$initial
final <- hump$final
look <- OF(final, initial, W=9, verbose=TRUE)
plot(look) # Compare with Fig. 1 in Marzban and Sandgathe (2010).
hist(look) # 2-d histogram.
plot(look, full=TRUE) # More plots.

## End(Not run)
```

---

# pphindcast2d *Practically Perfect Hindcast Neighborhood Verification Method*

---

## Description

Function to perform the practically perfect hindcast neighborhood verification method. Finds the optimal threhsold, Pthresh, and calculates the desired statistic for that threshold.

## Usage

```
pphindcast2d(object, which.score = "ets", time.point = 1, obs = 1,
                 model = 1, levels = NULL, max.n = NULL, smooth.fun =
                 "hoods2dsmooth", smooth.params = NULL, rule = ">=",
                 verbose = FALSE, ...)

## S3 method for class 'pphindcast2d'
plot(x, ..., mfrow = NULL,
    type = c("quilt", "line"),
    col = heat.colors(12), horizontal = FALSE)

## S3 method for class 'pphindcast2d'
print(x, ...)
```

## Arguments

| | |
|---|---|
| object | A list object returned by the make.SpatialVx function. |
| which.score | character stating which verification score is to be used. Must be one that is accepted by vxstats. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| levels | numeric vector giving the successive values of the smoothing parameter. For example, for the default method, these are the neighborhood lengths over which the levels^2 nearest neighbors are averaged for each point. Values should make sense for the specific smoothing function. For example, for the default method, these should be odd integers. |
| max.n | (optional) single numeric giving the maximum neighborhood length to use. Only used if levels are NULL. |
| smooth.fun | character giving the name of a smoothing function to be applied. Default is an average over the n^2 nearest neighbors, where n is taken to be each value of the levels argument. |
| smooth.params | list object containing any optional arguments to smooth.fun. Use NULL if none. |
| rule | character string giving the threshold rule to be applied. See help file for thresholder function for more information. |
| verbose | logical, should progress information be printed to the screen? |
| x | An object of class "pphindcast2d" as returned by the self-same function. |
| mfrow | mfrow parameter (see help file for par). If NULL, then the parameter is not re-set. |
| type | character specifying whether two quilt plots (one for the score and one for Pthresh) should be made, or one line plot incorporating both the score and the Pthresh values; the latter's values being displayed on the right axis. |
| col, horizontal | arguments used in the calls by image and image.plot. |
| ... | pphindcast2d: optional arguments to the optim function. May not include lower, upper or method as these are hard coded into the function.<br>plot method function: optional arguments to the image function.<br>print method function: not used. |

## Details

The practically perfect hindcast method is described in Ebert (2008). Using a similar notation as that described therein (and in the help page for hoods2d), the method is a SO-NF approach that first compares the observed binary field (obtained from the trheshold(s) provided by object), Ix, with the smoothed binary field, <Px>s. This smoothed binary field is thresholded by Pthresh to obtain a new binary field. The value of Pthresh that maximizes the verification score (provided by the which.score argument) is then used to compare Ix with <Iy>s, the binary forecast field obtained by thresholding the smoothed binary forecast field Iy using the value of Pthresh found above. The verification statistic determined by which.score is calculated between Ix and <Iy>s.

## Value

A list object is returned with components:

| | |
|---|---|
| which.score | value of which.score, same as the argument passed in. |
| Pthresh | l by q matrix giving the value of Pthresh applied at each level (rows) and threshold (columns). |
| values | l by q matrix giving the value of which.score found for each level (rows) and threshold (columns). |

## Warning

The value Pthresh is optimized under the assumption that larger values of which.score are better.

## Author(s)

Eric Gilleland

## References

Ebert, E. E. (2008) Fuzzy verification of high resolution gridded forecasts: A review and proposed framework. *Meteorol. Appl.*, **15**, 51–64. doi:10.1002/met.25

## See Also

hoods2d, kernel2dsmooth, vxstats, hoods2dPlot, optim

## Examples

```
x <- y <- matrix( 0, 50, 50)
x[ sample(1:50,10), sample(1:50,10)] <- rexp( 100, 0.25)
y[ sample(1:50,20), sample(1:50,20)] <- rexp( 400)

hold <- make.SpatialVx( x, y, thresholds=c(0.1, 0.5), field.type = "random")
look <- pphindcast2d(hold, levels=c(1, 3))
look
## Not run:
data( "geom001" )
data( "geom000" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( geom000, geom001, thresholds = c(0.01, 50.01),
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    data.name = "Geometric", obs.name = "geom000", model.name = "geom001",
    field.type = "Precipitation", units = "mm/h")

look <- pphindcast2d( hold, levels=c(1, 3, 65), verbose=TRUE)

plot(look, mfrow = c(1, 2) )
plot(look, mfrow = c(1, 2), type = "line")

# Alternatively:
```

```
par( mfrow = c(1, 2) )
hoods2dPlot( look$values, args = attributes( look ),
    main="Gilbert Skill Score")

## End(Not run)
```

---

`rigider` *Rigid Transformation*

---

## Description

Find the optimal rigid transformation for a spatial field (e.g. an image).

## Usage

```
rigider(x1, x0, p0, init = c(0, 0, 0), type = c("regular", "fast"),
    translate = TRUE, rotate = FALSE, loss, loss.args = NULL,
    interp = "bicubic", stages = TRUE,
    verbose = FALSE, ...)

## S3 method for class 'rigided'
plot(x, ...)

## S3 method for class 'rigided'
print(x, ...)

## S3 method for class 'rigided'
summary(object, ...)

rigidTransform(theta, p0, N, cen)
```

## Arguments

| | |
|---|---|
| `x1, x0` | matrices of same dimensions giving the forecast (or 1-energy) and observation (or 0-energy) fields, resp. |
| `x, object` | list object of class "rigided" as output by `rigider`. |
| `N` | (optional) the dimension of the fields (i.e., if `x1` and `x0` are n by m, then `N` is the product m * n). |
| `cen` | N by 2 matrix whoes rows are all the same giving the center of the field (used to subtract before determining rotations, etc.). |
| `p0` | N by 2 matrix giving the coordinates for the 0-energy (observed) field. |

| | |
|---|---|
| `init` | (optional) numeric vector of length equal to the number of parameters (e.g., 2 for translation only, 3 for both, and 1 for rotation only). If missing, then these will be estimated by taking the difference in centroids (translation) and the difference in orientation angles (rotation) as determined using image moments by way of `imomenter`. |
| `theta` | numeric vector of length 1, 2 or 3 (depending on whether you want to translate only (2), rotate only (1) or both (3)) giving the rigid transformation parameters. |
| `type` | character stating whether to optimize a loss function or just find the centroid (and possibly orientation angle) difference(s). |
| `translate, rotate` | logical, should the optimal translation/rotation be found? |
| `loss` | character naming a loss function (see details) to use in optimizing the rigid transformation (defaults to square error loss. |
| `loss.args` | named list giving any optional arguments to `loss`. |
| `interp` | character naming the 2-d interpolation method to use in calls to `Fint2d`. Must be one of "round" (default), "bilinear" or "bicubic". |
| `stages` | logical. Should the optimal translation be found before finding both the optimal tranlsation and rotation? |
| `verbose` | logical. Should progress information be printed to the screen? |
| `...` | optional arguments to `nlminb`. |

## Details

A rigid transformation translates coordinates of values in a matrix and/or rotates them. That is, if (r, s) are coordinates in a field with center (c1, c2), then the rigid transformation with parameters (x, y) and theta is given by:

(r, s) + (x, y) + Phi ((r, s) - (c1, c2)),

where Phi is the matrix with first column given by (cos( theta ), - sin( theta)) and second column given by (sin( theta), cos( theta )).

The optimal transformation is found by way of numerical optimization using the `nlminb` function on the loss function given by `loss`. If no value is given for `loss`, then square error loss is assumed. In this case, the loss function is based on an assumption of Gaussian errors, but this assumption is only important if you try to make inferences based on this model, in which case you should probably think much harder about what you are doing. In particular, the default objective function, Q, is given by:

Q = - sum( ( F(W(s)) - O(s) )^2 / (2 * sigma^2) - (N / 2) * log( sigma^2 ),

where s are the coordinates, W(s) are the rigidly transformed coordinates, F(W(s)) is the value of the 1-enegy field (forecast) evaluated at W(s) (which is interpolated as the translations typically do not give integer translations), O(s) is the 0-energy (observed) field evaluated at coordinate s, and sigma^2 is the estimated variance of the error field. A good alternative is to use "QcorrRigid", which calculates the correlation between F and O instead, and has been found by some to give better performance.

The function `rigidTransform` performs a rigid transform for given parameter values. It is intended as an internal function, but may be of use to some users.

## Value

A list object of class "rigided" is returned with components:

| | |
|---|---|
| `call` | the function call. |
| `translation.only` | If stages argument is true, this part is the optimal translation before rotation. |
| `rotate` | optimal translation and rotation together, if stages argument is true. |
| `initial` | initial values used. |
| `interp.method` | same as input argument interp. |
| `optim.args` | optional arguments passed to nlminb. |
| `loss, loss.args` | same as input arguments. |
| `par` | optimal parameter values found. |
| `value` | value of loss function at optimal parameters. |
| `x0, x1, p0` | same as input arguments. |
| `p1` | transformed p0 coordinates. |
| `x1.transformed` | The field F(W(s)). |

## Note

Finding the optimal rigid transformation can be very tricky when applying both rotatons and translations. This function helps, but for some fields may require more user input than is ideal, and should be considered experimental for the time being; as the examples will demonstrate. It does seem to work well for translations only, which has been the recommended course of action for the CRA method.

## Author(s)

Eric Gilleland

## See Also

`nlminb`, `Fint2d`

## Examples

```
# Simple uninteresting example for the R robots.
x <- y <- matrix(0, 20, 40)

x[ 12:18, 2:3 ] <- 1

y[ 13:19, 5:6 ] <- 1

xycoords <- cbind(rep(1:20, 40), rep(1:40, each = 20))

tmp <- rigider(x1 = x, x0 = y, p0 = xycoords)
```

```
tmp
plot(tmp)

# Rotate a coordinate system.
data( "geom000" )

loc <- cbind(rep(1:601, 501), rep(1:501, each = 601))

# Rotate the coordinates by pi / 4.
th <- c(0, 0, pi / 4)
names(th) <- c("x", "y", "rotation")
cen <- colMeans(loc[ geom000 > 0, ])
loc2 <- rigidTransform(theta = th, p0 = loc, cen = cen)

geom101 <- Fint2d(X = geom000, Ws = loc2, s = loc, method = "round")

## Not run:

image.plot(geom101)

# Try to find the optimal rigid transformation.
# First, allow a translation as well as rotation.

tmp <- rigider(x1 = geom101, x0 = geom000, p0 = loc,
    rotate = TRUE, verbose = TRUE)
tmp
plot(tmp)

# Now, only allow rotation, which does not work as
# well as one would hope.
tmp <- rigider(x1 = geom101, x0 = geom000, p0 = loc,
    translate = FALSE, rotate = TRUE, verbose = TRUE)
tmp
plot(tmp)

# Using correlation.
tmp <- rigider(x1 = geom101, x0 = geom000, p0 = loc,
    rotate = TRUE, loss = "QcorrRigid", verbose = TRUE)
tmp
summary(tmp)
plot(tmp)

##
## Examples from ICP phase 1.
##
## Geometric cases.
##

data( "geom001" )
data( "geom002" )
data( "geom003" )
data( "geom004" )
data( "geom005" )
```

```
tmp <- rigider(x1 = geom001, x0 = geom000, p0 = loc, verbose = TRUE)
tmp
plot(tmp)

tmp <- rigider(x1 = geom002, x0 = geom000, p0 = loc, verbose = TRUE)
tmp
plot(tmp)

tmp <- rigider(x1 = geom003, x0 = geom000, p0 = loc, verbose = TRUE)
tmp
plot(tmp)

tmp <- rigider(x1 = geom004, x0 = geom000, p0 = loc, verbose = TRUE)
tmp
plot(tmp)

# Note: Above is a scale error rather than a rotation, but can we
# approximate it with a rotation?
tmp <- rigider(x1 = geom004, x0 = geom000, p0 = loc, rotate = TRUE,
    verbose = TRUE)
tmp
plot(tmp)
# Ok, maybe need to give it better starting values?  Or, run it again
# with just the translation.

tmp <- rigider(x1 = geom005, x0 = geom000, p0 = loc, verbose = TRUE)
tmp
plot(tmp)



## End(Not run)
```

---

S1 *S1 Score, Anomaly Correlation*

---

## Description

Calculate the S1 score and anomaly correlation for a verification set.

## Usage

```
S1(x, ...)

## Default S3 method:
S1(x, ..., xhat, gradFUN = "KernelGradFUN")

## S3 method for class 'SpatialVx'
```

```
S1(x, ..., xhat, gradFUN = "KernelGradFUN",
    time.point = 1, obs = 1, model = 1)

ACC(x, ...)

## Default S3 method:
ACC(x, ..., xhat, xclim = NULL, xhatclim = NULL)

## S3 method for class 'SpatialVx'
ACC(x, ..., xclim = NULL, xhatclim = NULL,
    time.point = 1, obs = 1, model = 1)
```

## Arguments

| | |
|---|---|
| x,xhat | m by n matrices giving the verification and forecast fields, resp.<br>For S1.SpatialVx and ACC.SpatialVx, x is an object of class "SpatialVx". |
| xclim, xhatclim | m by n matrices giving the climatologies for X and Y, resp. If NULL, the result is simply a usual correlation. |
| gradFUN | character identifying a function used to calculate the gradient fields for X and Y. The default KernelGradFUN is to use a Laplacian of Gaussian kernel. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| ... | optional arguments to the gradFUN function. In the case of the default, the kernel can be changed (e.g., if only "laplacian" is desired), and optional arguments to the kernel2dmeitsjer function (in this case, nx, ny and sigma). Not used by ACC. |

## Details

The S1 score is given by

S1 = 100*sum(abs(DY_i - DX_i))/sum(max(abs(DY_i),abs(DX_i))),

where DY_i (DX_i)is the gradient at grid point i for the forecast (verification). See Brown et al. (2012) and Thompson and Carter (1972) for more on this score.

The ACC is just the correlation between X - Xclim and Y - Yclim.

## Value

single numeric

## Author(s)

Eric Gilleland

## References

Brown, B.G., Gilleland, E. and Ebert, E.E. (2012) Chapter 6: Forecasts of spatial fields. pp. 95–117, In *Forecast Verification: A Practitioner's Guide in Atmospheric Science*, 2nd edition. Edts. Jolliffee, I. T. and Stephenson, D. B., Chichester, West Sussex, U.K.: Wiley, 274 pp.

Thompson, J. C. and Carter, G. M. (1972) On some characteristics of the S1 score. *J. Appl. Meteorol.*, **11**, 1384–1385.

## See Also

kernel2dmeitsjer

## Examples

```
data( "UKobs6" )
data( "UKfcst6" )

S1( UKobs6, xhat = UKfcst6 )
ACC( UKobs6, xhat = UKfcst6 )

## Not run:
data( "obs0426" )
data( "wrf4ncar0425" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( obs0426, wrf4ncar0425, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP NSSL/SPC Spring 2005 Cases",
    obs.name = "obs0426", model.name = "wrf4ncar0425" )

plot( hold )

S1( hold )
ACC( hold )

## End(Not run)
```

---

# saller *Feature-based Analysis of a Field (Image)*

---

## Description

Feature-based analysis of a field (image)

## Usage

```
saller(x, d = NULL, distfun = "rdist", ...)

## S3 method for class 'saller'
print(x, ...)

## S3 method for class 'saller'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x | saller: x is a list object returned by FeatureFinder or other feature identification function that returns a list with components X.feats, Y.feats (themselves lists with owin class objects defining separate features in the verification and forecast fields, resp.), and X.labeled, Y.labeled (fields with the numbers from 0 to the number of features also defining the separate feature locations (e.g., as returned by the connected function of package **spatstat**.<br>print: list object returned by saller. |
| object | summary: object the returned by saller. |
| d | (optional) the SAL (saller) method requires division by the longest distance between two border points. If NULL, this is taken to be simply the length of the longest side. |
| distfun | Function with which to calculate centroid distances. Default uses straight Euclidean. To do great-circle distance, use rdist.earth and be sure that object has a loc attribute with lon/lat coordinates. |
| ... | Optional arguments to distfun. Not used by print or summary. |

## Details

saller: Computes S, A, and L of the SAL method introduced by Wernli et al. (2008).

## Value

saller returns a list with components:

| | |
|---|---|
| A | numeric giving the amplitude component. |
| L | numeric giving the lcoation component. |
| S | numeric giving the structure component. |
| L1,L2 | numeric giving the values that sum together to give L. |
| L1.alt, L.alt | numeric giving an alternative L1 component, and subsequently alternative L where it is calculated using the centroid of the field containing only defined features rather than the original raw field. |

print invisibly returns a named vector with S, A and L.

summary does not return anything.

## Note

There are several ways to identify features, and some are provided by this package, but only a few. For example, the method for identifying features in the SAL method as introduced by Wernli et al. (2008) utilizes information from a contour field of a particular variable, and is therefore not currently included in this package. Users are encouraged to write their own such functions, and should feel free to contribute them to this package by contacting the maintainer.

The SAL method typically looks at a small domain, and it is up to the user to set this up before calling these functions, as they are not designed to handle such a situation.

## Author(s)

Eric Gilleland

## References

Wernli, H., Paulat, M., Hagen, M. and Frei, C. (2008) SAL–A novel quality measure for the verification of quantitative precipitation forecasts. *Mon. Wea. Rev.*, **136**, 4470–4487, doi:10.1175/2008MWR2415.1.

## See Also

centroid.owin, connected, tiles, tess, deltamm, make.SpatialVx

## Examples

```
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx
xhat <- ExampleSpatialVxSet$fcst

q <- mean( c(c(x[x>0]),c(xhat[xhat>0])), na.rm=TRUE)

hold <- make.SpatialVx( x, xhat, field.type="contrived", units="none",
    data.name = "Example", obs.name = "x", model.name = "xhat" )

hold2 <- FeatureFinder(hold, smoothpar=5, thresh=q)
## Not run: plot(hold2)

look <- saller(hold2)
summary(look)

## Not run:
data( "pert000" )
data( "pert004" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Perturbed ICP Cases", obs.name = "pert000",
    model.name = "pert004" )
```

```
look <- FeatureFinder(hold, smoothpar=10.5)
summary(look)
plot(look)

saller(look)


## End(Not run)
```

---

`Sindex` *Shape Index*

---

## Description

Calculate the shape index (Sindex) as described in AghaKouchak et al. (2011)

## Usage

```
Sindex(x, thresh = NULL, ...)

## Default S3 method:
Sindex(x, thresh = NULL, ...,
    loc = NULL)

## S3 method for class 'SpatialVx'
Sindex(x, thresh = NULL, ...,
    time.point = 1, obs = 1, model = 1)
```

## Arguments

| | |
|---|---|
| `x` | Default: m by n numeric matrix giving the field for which the shape index is to be calculated.<br>`Sindex.SpatialVx`: list object of class "SpatialVx". |
| `thresh` | numeric giving a threshold under which (and including, i.e., <=) all values are set to zero, and the shape index is calculated for the non-zero (positive-valued) grid-points. |
| `loc` | (optional) mn by 2 numeric matrix giving the grid point locations. If NULL, the expanded grid with x=1:m and y=1:n is used. |
| `time.point` | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| `obs, model` | numeric indicating which observation/forecast model to select for the analysis. |
| `...` | Not used. |

## Details

The shape index introduced in AghaKouchak et al. (2011) is defined as

Sindex = Pmin/P,

where for n = the number of positive-valued grid points, Pmin = 4*sqrt(n) if floor(sqrt(n)) = sqrt(n), and Pmin = 2 * floor(2*sqrt(n)+1) otherwise. P is the permieter of the non-zero grid points. Range is 0 to 1. Values closer to 1 indicate shapes that are closer to circular.

## Value

numeric with named components:

| | |
|---|---|
| `Sindex` | the shape index |
| `Pmin,P` | the numerator and denominator (perimeter) that make the Sindex. |

For "SpatialVx" objects, the routine is applied to both the verification and forecast objects so that a two-row matrix is returned containing the above vectors for each field.

## Author(s)

Eric Gilleland

## References

AghaKouchak, A., Nasrohllahi, N., Li, J., Imam, B. and Sorooshian, S. (2011) Geometrical characterization of precipitation patterns. *J. Hyrdometeorology*, **12**, 274–285, doi:10.1175/2010JHM1298.1.

## See Also

Cindex, Aindex

## Examples

```
# Re-create Fig. 7a from AghaKouchak et al. (2011).
tmp <- matrix(0, 8, 8)
tmp[3,2:4] <- 1
tmp[5,4:6] <- 1
tmp[7,6:7] <- 1
Sindex(tmp)

## Not run:
# Two separate areas with highly structured shapes, but far away from each other.
data( "pert000" )
data( "pert006" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert006, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert006" )
```

```
plot( hold )

Sindex( hold )

## End(Not run)
```

---

spatbiasFS *Field Significance Method of Elmore et al. (2006)*

---

## Description

Apply field significance method of Elmore et al. (2006).

## Usage

```
spatbiasFS(X, Y, loc = NULL, block.length = NULL, alpha.boot = 0.05, field.sig = 0.05,
    bootR = 1000, ntrials = 1000, verbose = FALSE)

## S3 method for class 'spatbiasFS'
summary(object, ...)

## S3 method for class 'spatbiasFS'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| X,Y | m by n matrices giving the verification and forecast fields, resp., for each of m time points (rows) and n locations (columns). |
| x,object | list object as returned by spatbiasFS. |
| loc | optional (for subsequent plotting) n by 2 matrix giving the lon/lat coordinates for the locations. |
| block.length | numeric giving the block length to be used n the block bootstrap algorithm. If NULL, floor(sqrt(n)) is used. |
| alpha.boot | numeric between 0 and 1 giving the confidence level desired for the bootstrap algorithm. |
| field.sig | numeric between 0 and 1 giving the desired field significance level. |
| bootR | numeric integer giving the number of bootstrap replications to use. |
| ntrials | numeric integer giving the number of Monte Carol iterations to use. |
| verbose | logical, should progress information be printed to the screen? |
| ... | not used. |

## Details

See Elmore et al. (2006) for details.

## Value

A list object with components:

| | |
|---|---|
| `data.name` | character vector giving the name of the verification and forecast spatio-temporal fields used, and the associated location object (if not NULL). |
| `block.boot.results` | object of class LocSig |
| `sig.results` | list object containing information about the significance of the results. |
| `field.significance,alpha.boot` | field significance level and bootstrap CI level as input by field.sig alpha.boot arguments. |
| `bootR,ntrials` | same as arguments above. |

## Author(s)

Eric Gilleland and Kimberly L. Elmore

## References

Elmore, K. L., Baldwin, M. E. and Schultz, D. M. (2006) Field significance revisited: Spatial bias errors in forecasts as applied to the Eta model. *Mon. Wea. Rev.*, **134**, 519–531.

## See Also

MCdof, LocSig, tsboot

## Examples

```
data(GFSNAMfcstEx)
data(GFSNAMobsEx)
data(GFSNAMlocEx)
id <- GFSNAMlocEx[,"Lon"] >=-95 & GFSNAMlocEx[,"Lon"] <= -75 & GFSNAMlocEx[,"Lat"] <= 32
loc <- GFSNAMlocEx[id,]
GFSobsSub <- GFSNAMobsEx[,id]
GFSfcstSub <- GFSNAMfcstEx[,id]
look <- spatbiasFS(GFSobsSub, GFSfcstSub, loc=loc, bootR=500, ntrials=500)
plot(look)
summary(look)
```

---

# spct *Spatial Prediction Comparison Test*

---

## Description

Spatial Prediction Comparison Test (SPCT) for spatial locations that are on a regular or irregular coordinate system.

## Usage

```
spct(d, loc, trend = 0, lon.lat = TRUE,
    dmax = NULL, vgmodel = "expvgram", vgmodel.args = NULL,
    init, alpha = 0.05, alternative = c("two.sided", "less", "greater"), mu = 0,
    verbose = FALSE, ...)
```

## Arguments

| | |
|---|---|
| d | numeric vector of length n giving the (spatial) loss differential field (at a single point in time). |
| loc | n by 2 numeric matrix giving the spatial coordinates for each data point in d. |
| trend | a numeric vector of length one or n to be subtracted from d before finding the variogram and perfomring the test. |
| lon.lat | logical stating whether or not the values in loc are longitude/latitude coordinates or not. If TRUE, then the **fields** function rdist.earth is used to calculate distances. If FALSE, then the **fields** function rdist is used. |
| dmax | single numeric giving the maximum lag distance over which to fit the parametric variogram model. The default uses half of the maximum lag. |
| vgmodel | character string naming a function defining the parametric variogram model to be used. The default uses expvgram, the exponential variogram model. Must have arguments p (vector of parameters), h (vector of distances) and .... |
| vgmodel.args | Optional list of other arguments to be passed to vgmodel. Not used by the default method. |
| init | Initial parameter values to be used in the call to nlminb for estimating the parameters of the variogram model. The default for the default exponential variogram is to use the square root of the first-lag of the empirical variogram for the nugget and the difference between the second and first lag variogram values (if the second lag term is positive), and the first lag term otherwise for the range parameter. |
| alpha | single numeric giving the desired level of significance. |
| alternative | character string naming which type of hypothesis test to conduct. Default is to do a two-sided test. Note that the SPCT is paired test. |
| mu | The mean loss differential value under the null hypothesis. Usually, this will be zero (the default value). |
| verbose | logical, should progress information be printed to the screen? It may also provide other useful information in the event that a problem occurs somewhere. |
| ... | Optional arguments to vgram from the **fields** package. |

## Details

If using a large spatial data set that occurs on a regular grid, you should probably use lossdiff, empiricalVG.lossdiff, flossdiff and summary to perform this self-same test (the SPCT), as those functions make use of special tricks for regular grids to speed things up. Otherwise, this function should work on either type of grid.

The SPCT is a paired test introduced by Hering and Genton (2011)–and based on the time series test introduced by Diebold and Mariano (1995) of whether one of two competing forecasts is better than the other (alternative) or not (null). Apart from being a test for spatial fields, the SPCT test fits a parametric model to the empirical variogram (instead of using the empirical one), which turns out to be more accurate.

The loss differential field is a field giving the straight difference between the two loss functions calculated for each of two forecasts. For example, suppose Z(x,y) is an observed spatial field with (possibly irregularly spaced) locations (x, y), and Y1(x, y) and Y2(x, y) are two competing forecasts. One might be interested in whether or not, on average, the difference in the absolute error for Y1 and Y2 is significantly different from zero. First, g1 = abs( Y1(x, y) - Z(x, y) ) and g2 = abs( Y2(x, y) - Z(x, y) ). Second, the loss differential field is D(x, y) = g1 - g2. It is the average of D(x, y) that is of interest. Because D(x, y) is likely to have a strong spatial correlation, the standard error for Dbar = mean( D(x, y) ) is calculated from the variogram. Hering and Genton (2011) found the test to have proper size and good power, and found it to be relatively robust to contemporaneous correlation–i.e., if Y1 and Y2 are correlated (even if they are not, which is unlikely, g1 and g2 will necessarily be correlated because both involve the same field Z).

If the sample size is less than 30, a t-test is used, and a normal approximation otherwise.

See also, Gilleland (2013) for a modification of this test that accounts for location errors (coming soon).

## Value

A list object of class "htest" with components:

`data.name` a character string giving the name of the loss differential field.

`loss.differential` The original loss differential field as passed by argument d.

`nloc` the number of spatial locations.

`trend` Same as the argument passed in.

`optional.arguments` list with any arguments passed into vgram.

`empirical.variogram` the object returned by vgram giving the empirical variogram.

`parametric.vgram.fit` the value returned by nlminb or an object of class "try-error".

`estimate` the estimated mean loss differential value.

`se` the estimated standard error estimated from the fitted variogram model.

`statistic` the value of the statistic ( mean( d ) - mu ) / se.

`null.value` the argument mu.

`parameter` numeric vector giving the parameter values estimated for the variogram model.

`fitted.values` the predicted variogram values from the fitted parametric model.

`loss.differential.detrended` this is the loss differential field after having been de-trended.

`alternative` a character string describing the alternative hypothesis.

| | |
|---|---|
| p.value | the p-value for the test. |
| conf.int | The (1 - alpha) * 100 percent confidence interval found using the standard error based on the variogram model per hering and Genton (2011). |
| method | a character string indicating the type of test performed. |

## Author(s)

Eric Gilleland

## References

Diebold, F.X. and Mariano, R.S. (1995) Comparing predictive accuracy. *Journal of Business and Economic Statistics*, **13**, 253–263.

Gilleland, E. (2013) Testing competing precipitation forecasts accurately and efficiently: The spatial prediction comparison test. *Mon. Wea. Rev.*, **141**, (1), 340–355.

Hering, A. S. and Genton, M. G. (2011) Comparing spatial predictions. *Technometrics* **53**, (4), 414–425.

## See Also

vgram, lossdiff, flossdiff, summary.lossdiff, expvgram

## Examples

```
## Not run:
y1 <- predict( Tps( fields::ozone$x, fields::ozone$y ) )
y2 <- predict( Krig( fields::ozone$x, fields::ozone$y, theta = 20 ) )

y <- fields::ozone$y

spct( abs( y1 - y ) - abs( y2 - y ), loc = fields::ozone$x )

spct( abs( y1 - y ) - abs( runif( 20, 1, 5 ) - y ), loc = fields::ozone$x )

## End(Not run)
```

---

# structurogram *Structure Function for Non-Gridded Spatial Fields.*

---

## Description

Computes pairwise differences (raised to the q-th power) as a function of distance. Returns either raw values or statistics from binning.

## Usage

```
structurogram(loc, y, q = 2, id = NULL, d = NULL, lon.lat = FALSE, dmax = NULL,
    N = NULL, breaks = NULL)

## S3 method for class 'structurogram'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| loc | numeric matrix where each row is the coordinate of a point in the field. |
| x | list object returned by structurogram function. |
| y | numeric vector giving the value of the field at each location. |
| q | numeric giving the value to which the paired differences should be raised. Default (q=2) gives the usual semivariogram. |
| id | A 2 column matrix that specifies which variogram differnces to find. If omitted all possible pairings are found. This can used if the data has an additional covariate that determines proximity, for example a time window. |
| d | numeric matrix giving the distances among pairs (indexed by id). If not included, these are determined directly from loc. |
| lon.lat | logical, are the coordinates longitude/latitude coordinates? If so, distances are found using great-circle distance. |
| dmax | numeric giving the maximum distance for which to compute the structure function. |
| N | numeric giving the number of bins to use. |
| breaks | numeric vector giving bin boundaries for binning structure function values. Need not be equally spaced, but must be ordered. |
| ... | optional arguments to plot function. |

## Details

This function is basically an exact copy of vgram from package **fields** whereby the differences are raised to a power of q instead of 2. That is, it calculates the structure function given by Eq (4) in harris et al. (2001). Namely,

S_q(l_x,l_y) = <|R(x+l_x,y+l_y) - R(x,y)|^q>

where R is the field of interest, <> denotes the average over pixels in the image (note, in Harris et al. (2001), this is only over non-zero pixels, so is only equivalent to this equation if zero-valued points are first removed from y and loc), l_x and l_y are lags in the x and y directions, resp. If q=2, then this is the semivariogram.

The plot method function plots the structure by separation distance (circles) along with a dark blue line giving the bin centers.

## Value

A list object of class "structurogram" is returned with components:

| | |
|---|---|
| d | numeric vector giving the pair-wise distances. |
| val | numeric vector giving the structure function values for each distance. |
| q | numeric giving the value of q passed into the function. |
| call | Calling string |
| stats | Matrix of statistics for values in each bin. Rows are the summaries returned by the stats function or describe (see package fields). If either breaks or N arguments are not supplied then this component is not computed. |
| centers | numeric vector giving the bin centers. |

The plot method function does not return anything.

## Author(s)

Eric Gilleland

## References

Harris, D., Foufoula-Georgiou, E., Droegemeier, K. K. and Levit, J. J. (2001) Multiscale statistical properties of a high-resolution precipitation forecast. *J. Hydrometeorol.*, **2**, 406–418.

## See Also

vgram, vgram.matrix, structurogram.matrix

## Examples

```
data( ozone2)
good<- !is.na(ozone2$y[16,])
x<- ozone2$lon.lat[good,]
y<- ozone2$y[16,good]

look <- structurogram( x,y, N=15, lon.lat=TRUE)
plot(look)
# Compare above with results from example for function vgram from package fields.

look <- structurogram( x,y, N=15, lon.lat=TRUE, q=1)
plot(look)
```

`structurogram.matrix` *Structure Function for Gridded Fields*

## Description

Calculates the structure function to the q-th order for gridded fields.

## Usage

```
structurogram.matrix(dat, q = 2, R = 5, dx = 1, dy = 1, zero.out = FALSE)

## S3 method for class 'structurogram.matrix'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| `dat` | n by m matrix of numeric values defining a gridded spatial field (or image) such that distances can be determined from their positions in the matrix. |
| `x` | list object output from `structurogram.matrix` |
| `q` | numeric giving the order for the structure function (q = 2 yields the more common semi-variogram). |
| `R` | numeric giving the maximum radius for finding the structure differences assuming that the grid points are spaced one unit apart. Default is to go to a radius of 5. |
| `dx,dy` | numeric giving the spacing of the grid points on the x- (y-) axis. This is used to calculate the correct distance between grid points. |
| `zero.out` | logical, should zero-valued pixels be ignored? |
| `...` | optional arguments to the `plot` function. |

## Details

This function is basically an exact copy of `variogram.matrix`, which itself is a copy of `vgram.matrix` from package **fields** (but allows and ignores missing values, in order to ignore zero-valued pixels and does not include Cressie's robust version of the variogram), whereby the differences are raised to a power of q instead of 2. That is, it calculates the structure function given by Eq (4) in harris et al. (2001). Namely,

S_q(l_x,l_y) = <|R(x+l_x,y+l_y) - R(x,y)|^q>

where R is the field of interest, <> denotes the average over pixels in the image (note, in Harris et al. (2001), this is only over non-zero pixels, so is only equivalent to this equation if zero.out=TRUE), l_x and l_y are lags in the x and y directions, resp. If q=2, then this is the semivariogram.

The `plot` method function makes two plots. The first shows the structure by separation distance ignoring direction (circles) and all values (i.e., for each direction, dots). The second shows the structure function values for separation distance and direction (see, e.g., `plot.vgram.matrix`).

## Value

A list with the following components:

| | |
|---|---|
| d | numeric vector of distances for the differences (ignoring direction). |
| vgram | numeric vector giving the structure function values. Note that the term 'vgram' is used here for compatibility with the plot.vgram.matrix function, which is employed by the plot method function used here. This set of values ignores direction. |
| d.full | numeric vector of distances for all possible shifts up distance R. |
| ind | two column matrix giving the x- and y- increment used to compute shifts. |
| vgram.full | numeric vector giving the structure function for each direction in addition to separation distance. Again, the word 'vgram' is used for compatibility with plot.vgram.matrix. |

Note that the plot method function does not return anything.

## Author(s)

Eric Gilleland

## References

Harris, D., Foufoula-Georgiou, E., Droegemeier, K. K. and Levit, J. J. (2001) Multiscale statistical properties of a high-resolution precipitation forecast. *J. Hydrometeorol.*, **2**, 406–418.

## See Also

vgram.matrix, vgram, structurogram, plot.vgram.matrix

## Examples

```
data( "lennon" )
look <- structurogram.matrix(lennon, q=2)
plot(look)
# Compare the above with
## Not run:
look2 <- vgram.matrix(lennon)
dev.new()
par(mfrow=c(1,2),bg="beige")
plot(look2$d, look2$vgram, xlab="separation distance", ylab="variogram")
points(look2$d.full, look2$vgram.full, pch=".")
plot.vgram.matrix(look2)

look <- structurogram.matrix(lennon, q=1)
plot(look)

look <- structurogram.matrix(lennon, q=1, zero.out=TRUE)
plot(look)

## End(Not run)
```

surrogater2d *Create Surrogate Fields*

## Description

Create surrogate fields that have the same power spectrum and pdf as the original field.

## Usage

```
surrogater2d(Im, frac = 0.95, n = 10, lossfun = "mae", maxiter = 100, zero.down = TRUE,
    verbose = FALSE, ...)

aaft2d(Im, bigdim = NULL)

fft2d(x, bigdim = NULL, ...)

mae(x1, x2, ...)
```

## Arguments

| | |
|---|---|
| Im | matrix from which surrogates are to be made. |
| x | matrix to be Fourier transformed. |
| x1,x2 | numeric or array of same dimensions giving the two fields over which to calculate the mean aboslute error. |
| frac | single numeric giving the fraction of original amplitudes to maintain. |
| n | single numeric giving the number of surrogate fields to create (should be a whole number). |
| lossfun | character naming the loss function to use in computing the error between simulated surrogate fields in the iterative process. Default is the mean absolute error given by the mae function detailed here. |
| maxiter | Maximum number of iterations allowed per surrogate. |
| zero.down | logical, does Im contain many zeros, and is otherwise positive? If so, this sets negative numbers and unusually small numbers to zero. |
| bigdim | numeric vector of length two giving the dimensions (larger than dimensions of Im) to compute the FFT's more efficiently (at least potentially). |
| verbose | logical, should progress information be printed to the screen? |
| ... | additional arguments: in the case of fft2d, they are additional arguments to fft (i.e., to use inverse=TRUE), in the case of surrogater2d, they are additional arguments to the loss function given by lossfun, and in the case of mae (default), these are not used. |

## Details

The `fft2d` function was written to simplify some of the code in `surrogater2d` and `aaft2d`. It is simply a call to the R function `fft`, but it first resets the dimensions to ones that should maximize the efficiency. It will also return the dimensions if they are not passed in.

Surrogates are used in non-linear time series analysis to simulate similar time series for hypothesis testing purposes (e.g., Kantz and Schreiber, 1997). Venugopal et al. (2005) use surrogates of two-dimensional fields as part of their Forecast Quality Index (FQI); which is the intention here. Theiler et al. (1992) proposed a method known as the amplitude adjusted Fourier transform (AAFT) algorithm, and Schreiber and Schmitz (1996) proposed a modification to this approach in order to obtain surrogates with both the same power spectrum and pdf as the original series.

The AAFT method first renders the original data, denoted here as s_n, Gaussian via a rank ordering based on randomly generated Gaussian simulated data. The resulting series, s_n'=g(s_n), is Gaussian and follows the same measured time evolution as s_n. Next, phase randomized surrogates are made for s_n', call them s_n". The rescaling g is then inverted by rank ordering s_n" according to the distribution of the original data, s_n. This algorithm yields surrogates with the same pdf of amplitudes as s_n by construction, but typically not the same power spectra. The algorithm proposed by Schreiber and Schmitz (1996) begins with the AAFT, and then iterates through a further algorithm as follows.

1. Hold a sorted list of s_n and the squared amplitudes of the Fourier transform of s_n, denote them by S2_k.

2. Take a random shuffle without replacement of the data, denote as s_n(0).

3. Take the Fourier transform of s_n(i).

4. Replace the S2_k(i) with S2_k.

5. Inverse the Fourier transform with the replaced amplitudes.

6. Rank order the series from 5 in order to assume exactly the values taken by s_n.

7. Check the accuracy of 6 using a loss function of some sort, and repeat steps 3 through 6 until a desired level of accuracy is achieved.

## Value

In the case of surrogater2d: A three dimesnional array of matrices with same dimension as Im, and third dimension giving the n surrogate fields.

In the case of aaft2d: A matrix of the same dimension as Im.

In the case of fft2d: If bigdim is NULL, a list object is returned with components fft and bigdim giving the FFT of x and the larger dimesnions used. Otherwise, a matrix of dimension x is returned giving the FFT (or inverse FFT) of x.

In the case of mae: a single numeric giving the mean absolute error between x1 and x2.

## Author(s)

Eric Gilleland, this code was adapted from matlab code written by Sukanta Basu (2007) available at: http://www.ral.ucar.edu/projects/icp/Software/FeaturesBased/FQI/Perturbed.m

## References

Kantz, H. and Schreiber, T. (1997) *Nonlinear time series analysis*. Cambridge University Press, Cambridge, U.K., 304pp.

Schreiber, T. and Schmitz, A. (1996) Improved surrogate data for nonlinearity tests. *Physical Review Letters*, **77**(4), 635–638.

Theiler, J., Eubank, S. Longtin, A. Galdrikian, B. and Farmer, J. D. (1992) *Physica* (Amsterdam) **58D**, 77.

Venugopal, V., Basu, S. and Foufoula-Georgiou, E. (2005) A new metric for comparing precipitation patterns with an application to ensemble forecasts. *J. Geophys. Res.*, **110**, D08111, doi:10.1029/2004JD005395, 11pp.

## See Also

fft, locmeasures2d, UIQI, ampstats

## Examples

```
data( "ExampleSpatialVxSet" )

x <- ExampleSpatialVxSet$vx

z <- surrogater2d( x, zero.down=FALSE, n=3)

## Not run:
par( mfrow=c(2,2))
image.plot( look)
image.plot( look2[,,1])
image.plot( look2[,,2])
image.plot( look2[,,3])

data( pert000)
tmp <- surrogater( pert000, n=10, verbose=TRUE)
boxplot( cbind( c(pert000), apply( tmp, 3, c)))

## End(Not run)
```

---

# thresholder *Apply a Threshold to a Field*

---

## Description

App;y a threshold to a field and return either a binary field or a field with replace.width everywhere the rule is not true.

## Usage

```
thresholder(x, type = c("binary", "replace.below"), th, rule = ">=",
    replace.with = 0, ...)

## Default S3 method:
thresholder(x, type = c("binary", "replace.below"), th, rule = ">=",
    replace.with = 0, ... )

## S3 method for class 'SpatialVx'
thresholder(x, type = c("binary", "replace.below"), th, rule = ">=",
    replace.with = 0, ..., time.point = 1, obs = 1, model = 1 )
```

## Arguments

| | |
|---|---|
| x | A field or "SpatialVx" object to which to apply the thresholds. |
| type | character describing which type of field(s) to return: binary or replace. |
| rule | If type is "binary", return 0 when the rule applied to a grid point's value is not true in relation to the threshold value, and 1 elsewhere. If type is "replace.below", then return replace.with wherever the rule is not true and return the original value otherwise. By default, it replaces values below the threshold with zero (hence its name), but if rule is, e.g., "<=", then it will replace values above with zero; or whatever value is chosen for replace.with. |
| replace.with | Only used if type is "replace.below". The value with which to replace values that are below (default) the threshold. |
| th | Value of the threshold (default) or index to which row of threshold matrices in thresholds attribute of "SpatialVx" object. Must be a single number. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| ... | Not used. |

## Details

At each point, p, in the field, the expression: p rule threshold is applied. If type is "binary", then if the expression is false, zero is returned for that grid point, and if it is true, then one is returned. If type is "replace.below", then if the expression is false, replace.with is returned for that grid point, and if true, then the original value is returned. By default, the original field is returned, but with values below the threshold set to zero. If rule is "<=", then replace.below will actually replace values above the threshold with "replace.with" instead.

If applied to a "SpatialVx" class object, then observation obs and model model at time point time.point will each be thresholded using the respective th threshold value for the observed and modeled fields as taken from the thresholds attribute of the object (see the help file for make.SpatialVx).

## Value

A field of the same dimension as x if a matrix. If x is a "SpatialVx" class object, then a list is returned with components:

`X, Xhat` The matrices giving the respective thresholded fields for the observation and forecast.

## Author(s)

Eric Gilleland

## See Also

`make.SpatialVx`

## Examples

```
x <- matrix( 12 + rnorm( 100, 10, 10 ), 10, 10 )

par( mfrow = c(2, 2) )
image.plot( thresholder( x, th = 12 ), main = "binary" )

image.plot( thresholder( x, type = "replace.below", th = 12 ),
    main = "replace.below" )

image.plot( thresholder( x, th = 12, rule = "<=" ),
    main = "binary with rule <=" )

image.plot( thresholder( x, type = "replace.below", th = 12, rule = "<=" ),
    main = "replace.below with rule <=" )

par( mfrow = c(1,1) )
## Not run:
data("geom000")
data("geom004")
data("ICPg240Locs")

hold <- make.SpatialVx( geom000, geom004, thresholds = c(0.01, 50.01),
    projection = TRUE, map = TRUE, loc = ICPg240Locs, loc.byrow = TRUE,
    field.type = "Geometric Objects Pretending to be Precipitation",
    units = "mm/h", data.name = "ICP Geometric Cases", obs.name = "geom000",
    model.name = "geom004" )

# Note: th = 1 means threshold = 0.01.
look <- thresholder( hold, th = 1 )

image.plot( look$X )
contour( look$Xhat, add = TRUE, col = "white" )

# Note: th = 2, means threshold = 50.01
look <- thresholder( hold, th = 2 )

image.plot( look$X )
contour( look$Xhat, add = TRUE, col = "white" )
```

```
look <- thresholder( hold, th = 1, rule = "<" )

image.plot( look$X )
contour( look$Xhat, add = TRUE, col = "white" )

## End(Not run)
```

---

UKobs6 *Example Precipitation Rate Verification Set (NIMROD)*

---

## Description

Example precipitation rate verification set from the very short-range mesoscale Numerical Weather Prediction (NWP) system used operationally at the UK Met Office.

## Usage

```
data(UKobs6)
data(UKloc)
```

## Format

The format is: chr "UKobs6"

The format is: num [1:65536, 1:2] -11 -10.9 -10.9 -10.8 -10.7 ...

## Details

Precipitation rate (mm/h) verification set from the very short-range NWP system called NIMROD used operationally at the UK Met Office, and described in detail in Casati et al. (2004). In particular, this is case 6 from Casati et al. (2004), showing a front timing error. These data are made available for scientific purposes only. Please cite the source in any papers or presentations. The proper reference is the U.K. Met Office.

Refer to Casati et al. (2004) for more information on these data.

The original lon/lat information is not available. 'UKloc' was created to match reasonably well with the figures in Casati et al. (2004), but should not be considered definite.

## Source

UK Met Office

## References

Casati, B., Ross, G. and Stephenson, D. B. (2004) A new intensity-scale approach for the verification of spatial precipitation forecasts. *Meteorol. Appl.*, **11**, 141–154, doi:10.1017/S1350482704001239.

## Examples

```
data( "UKobs6" )
data( "UKfcst6" )
data( "UKloc" )

zl <- range(c(c(UKobs6), c(UKfcst6)))
par(mfrow=c(1,2))
image(UKobs6, col=c("grey", tim.colors(64)), zlim=zl, main="analysis", axes=FALSE)
par(usr=apply(UKloc, 2, range))
map(add=TRUE)
image.plot(UKfcst6, col=c("grey", tim.colors(64)), zlim=zl, main="forecast", axes=FALSE)
par(usr=apply(UKloc, 2, range))
map(add=TRUE)
```

---

# upscale2d *Upscaling Neighborhood Verification on a 2-d Verification Set*

---

## Description

Perform upscaling neighborhood verification on a 2-d verification set.

## Usage

```
upscale2d(object, time.point = 1, obs = 1,
                 model = 1, levels = NULL, max.n = NULL, smooth.fun =
                 "hoods2dsmooth", smooth.params = NULL, rule = ">=",
                 verbose = FALSE)

## S3 method for class 'upscale2d'
plot(x, ... )

## S3 method for class 'upscale2d'
print(x, ...)
```

## Arguments

| | |
|---|---|
| object | list object of class "SpatialVx". |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| levels | numeric vector giving the successive values of the smoothing parameter. For example, for the default method, these are the neighborhood lengths over which the levels^2 nearest neighbors are averaged for each point. Values should make sense for the specific smoothing function. For example, for the default method, these should be odd integers. |
| max.n | (optional) single numeric giving the maximum neighborhood length to use. Only used if levels are NULL. |

| | |
|---|---|
| smooth.fun | character giving the name of a smoothing function to be applied. Default is an average over the n^2 nearest neighbors, where n is taken to be each value of the levels argument. |
| smooth.params | list object containing any optional arguments to smooth.fun. Use NULL if none. |
| rule | character string giving the threshold rule to be applied. See help file for thresholder function for more information. |
| verbose | logical, should progress information be printed to the screen? |
| x | list object of class "upscale2d" as returned by upscale2d. |
| ... | optional arguments to the image.plot function from package **fields**. Can also include the argument type, which must be one of "all", "gss", "ts", "bias" or "rmse". |

## Details

Upscaling is performed via neighborhood smoothing. Here, a boxcar kernel is convolved (using the convolution theorem with FFT's) to obtain an average over the nearest n^2 grid squares at each grid point. This is performed on the raw forecast and verification fields. The root mean square error (RMSE) is taken for each threshold (Yates et al., 2006; Ebert, 2008). Further, binary fields are obtained for each smoothed field via thresholding, and frequency bias, threat score ts) and equitable threat score (ets) are calculated (Zepeda-Arce et al., 2000; Ebert, 2008).

## Value

upscale2d returns a list of class "upscale2d" with components:

| | |
|---|---|
| rmse | numeric vector giving the root mean square error for each neighborhood size provided by object. |
| bias,ts,ets | numeric matrices giving the frequency bias, ts and ets for each neighborhood size (rows) and threshold (columns). |

## Author(s)

Eric Gilleland

## References

Ebert, E. E. (2008) Fuzzy verification of high resolution gridded forecasts: A review and proposed framework. *Meteorol. Appl.*, **15**, 51–64. doi:10.1002/met.25

Yates, E., Anquetin, S., Ducrocq, V., Creutin, J.-D., Ricard, D. and Chancibault, K. (2006) Point and areal validation of forecast precipitation fields. *Meteorol. Appl.*, **13**, 1–20.

Zepeda-Arce, J., Foufoula-Georgiou, E., Droegemeier, K. K. (2000) Space-time rainfall organization and its role in validating quantitative precipitation forecasts. *J. Geophys. Res.*, **105**(D8), 10,129–10,146.

## See Also

hoods2d, kernel2dsmooth, kernel2dmeitsjer, fft

## Examples

```
x <- matrix( 0, 50, 50)
x[ sample(1:50,10), sample(1:50,10)] <- rexp( 100, 0.25)
y <- kernel2dsmooth( x, kernel.type="disk", r=6.5)
x <- kernel2dsmooth( x, kernel.type="gauss", nx=50, ny=50, sigma=3.5)

hold <- make.SpatialVx( x, y, thresholds = seq(0.01,1,,5), field.type = "random")

look <- upscale2d( hold, levels=c(1, 3, 20) )
look

par( mfrow = c(4, 2 ) )
plot( look )

## Not run:
data( "geom001" )
data( "geom000" )
data( "ICPg240Locs" )

hold <- make.SpatialVx( geom000, geom001, thresholds = c(0.01, 50.01),
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Geometric", obs.name = "geom000", model.name = "geom001" )

look <- upscale2d(hold, levels=c(1, 3, 9, 17, 33, 65, 129, 257),
    verbose=TRUE)

par( mfrow = c(4, 2 ) )

plot(look )
look <- upscale2d(hold, q.gt.zero=TRUE, verbose=TRUE)
plot(look)
look <- upscale2d(hold, verbose=TRUE)
plot(look)


## End(Not run)
```

---

`variographier` *Variography Score*

---

## Description

Calculate the variography score between two spatial fields based on the fitted exponential variogram.

## Usage

```
variographier(x, init, zero.out = FALSE, ...)
```

```
## Default S3 method:
variographier( x, init, zero.out = FALSE, ..., y )

## S3 method for class 'SpatialVx'
variographier( x, init, zero.out = FALSE, ...,
    obs = 1, model = 1, time.point = 1 )
```

## Arguments

| | |
|---|---|
| x, y | matrices giving the fields on which to calculate the variography or a "SpatialVx" class object (x only). |
| init | list with components px and py that give initial values for parameter estimates (sill + nugget and range). If missing, default will attempt to find reasonable starting values. |
| zero.out | logical should the variogram be calculated over all grid points or just ones where one or both fields are non-zero? See variogram.matrix. |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| ... | optional arguments to vgram.matrix or variogram.matrix (if zero.out is TRUE). Can also have optional arguments to nlminb (but not lower or upper). |

## Details

The variography score calculated here is that from Ekstr\"om (2016). So far, only the exponential variogram is allowed.

Note that in the fitting, the model g(h) = c * ( 1 - exp( -a * h ) ) is used, but the variography is calculated for theta = 3 / a. Therefore, the values in the par component of the returned fitted variograms correspond to a, while the variography score corresponds to theta. The score is given by:

v = 1 / sqrt( c_0^2 + c_m^2 + ( theta_0 - theta_m )^2 )

where c_0 and c_m are the sill + nugget terms for the observation and model, resp., and similarly for theta_0 and theta_m.

The parameters are *not* currently normalized, here, to give equal weight between sill + nugget and range. If several fields are analyzed (e.g., an ensemble), then the fitted parameters could be gathered, and one could use that information to calculate the score based on a normalized version.

## Value

A list object of class "variographied" is returned with components:

| | |
|---|---|
| obs.vg, mod.vg | Empirical variogram objects as returned by either vgram.matrix or variogram.matrix |
| obs.parvg, mod.parvg | objects returned by nlminb containing the fitted exponential variogram model parameters and some information about the optimization. |
| variography | single numeric giving the variography measure. |

### Author(s)

Eric Gilleland

### References

Ekstr\"om, M. (2016) Metrics to identify meaningful downscaling skill in WRF simulations of intense rainfall events. *Environmental Modelling and Software*, **79**, 267–284, DOI: 10.1016/j.envsoft.2016.01.012.

### See Also

vgram.matrix, variogram.matrix

### Examples

```
data( "UKobs6" )
data( "UKfcst6" )
data( "UKloc" )

hold <- make.SpatialVx( UKobs6, UKfcst6, thresholds = c(0.01, 20.01),
    loc = UKloc, field.type = "Precipitation", units = "mm/h",
    data.name = "Nimrod", obs.name = "Observations 6", model.name = "Forecast 6",
    map = TRUE)

look <- variographier( hold )
look
plot( look )
```

---

## vxstats *Some Common Traditional Forecast Verification Statistics.*

---

### Description

Calculates some common traditional forecast verification statistics.

### Usage

```
vxstats(X, Xhat, which.stats = c("bias", "ts", "ets", "pod",
                 "far", "f", "hk", "bcts", "bcets", "mse"), subset =
                 NULL)
```

### Arguments

| | |
|---|---|
| X, Xhat | k by m matrix of verification and forecast values, resp. |
| which.stats | character vector giving the names of the desired statistics. See Details below. |
| subset | numeric vector indicating a subset of the verification set over which to calculate the verification statistics. |

## Details

Computes several traditional verification statistics (see Wilks, 2006, Ch. 7; Jolliffe and Stephenson, 2012 for more on these forecast verification statistics; as well as the Issues, Methods and FAQ web page of the Joint Working Group on Forecast Verification of the World Meteorological Organization at: http://www.cawcr.gov.au/projects/verification/) The possible statistics that can be computed, as determined by `which.stats` are:

"bias" the number of forecast events divided by the number of observed events (sometimes called frequency bias).

"ts" threat score, given by hits/(hits + misses + false alarms)

"ets" equitable threat score, given by (hits - hits.random)/(hits + misses + false alarms - hits.random), where hits.random is the number of observed events times the number of forecast events divided by the total number of forecasts.

"pod" probability of detecting an observed event (aka, hit rate). It is given by hits/(hits + misses).

"far" false alarm ratio, given by (false alarms)/(hits + false alarms).

"f" false alarm rate (aka probability of false detection) is given by (false alarms)/(correct rejections + false alarms).

"hk" Hanssen-Kuipers Score is given by the difference between the hit rate ("pod") and the false alarm rate ("f").

"bcts", "bcets", Bias Corrected Threat Score (Equitable Threat Score) as introduced in Mesinger (2008); see also Brill and Mesinger (2009). Also referred to as the dHdA versions of these scores.

"mse" mean square error (not a contingency table statistic, but can be used with binary fields). This is the only statistic that can be calculated here that does not require binary fields for `Fcst` and `Obs`.

## Value

A list with components determined by which.stats, which may include any or all of the following.

| | |
|---|---|
| `bias` | numeric giving the frequency bias. |
| `ts` | numeric giving the threat score. |
| `ets` | numeric giving the equitable threat score, also known as the Gilbert Skill Score. |
| `pod` | numeric giving the probability of decking an event, also known as the hit rate. |
| `far` | numeric giving the false alarm ratio. |
| `f` | numeric giving the false alarm rate. |
| `hk` | numeric giving the Hanssen and Kuipers statistic. |
| `bcts, bcets` | numeric giving the bias corrected version of the threat- and/or equitable threat score. |
| `mse` | numeric giving the mean square error. |

## Warning

It is up to the user to provide the appropriate type of fields for the given statistics to be computed. For example, they must be binary for all types of which.stats except mse.

## Note

See the web page: http://www.cawcr.gov.au/projects/verification/ for more details about these statistics, and references.

## Author(s)

Eric Gilleland

## References

Brill, K. F. and Mesinger, F. (2009) Applying a general analytic method for assessing bias sensitivity to bias-adjusted threat and equitable threat scores. *Wea. Forecasting*, **24**, 1748–1754.

Jolliffe, I. T. and Stephenson, D. B., Edts. (2012) *Forecast Verification: A Practitioner's Guide in Atmospheric Science*, 2nd edition. Chichester, West Sussex, U.K.: Wiley, 274 pp.

Mesinger, F. (2008) Bias adjusted precipitation threat scores. *Adv. Geosci.*, **16**, 137–142.

Wilks, D. S. (2006) *Statistical Methods in the Atmospheric Sciences*. 2nd Edition, Academic Press, Burlington, Massachusetts, 627pp.

## See Also

hoods2d

## Examples

```
# Calculate the traditional verification scores for the first geometric case
# of the ICP.
data( "geom001" )
data( "geom000" )

rmse <- sqrt(vxstats( geom001, geom000, which.stats="mse")$mse)
rmse
vxstats( geom001 > 0, geom000 > 0, which.stats=c("bias", "ts", "ets", "pod", "far", "f", "hk"))

data( "geom005" )
vxstats( geom005 > 0, geom000 >0, which.stats=c("ts","ets","bcts","bcets"))
```

---

# waveIS *Intensity Scale (IS) Verification*

---

## Description

Intensity Scale (IS) verification based on Casat et al (2004) and Casati (2010).

## Usage

```
waveIS(x, th = NULL, J = NULL, wavelet.type = "haar", levels
                 = NULL, max.n = NULL, smooth.fun = "hoods2dsmooth",
                 smooth.params = NULL, rule = ">=", verbose = FALSE,
                 ...)

## S3 method for class 'SpatialVx'
waveIS(x, th = NULL, J = NULL, wavelet.type = "haar", levels
                 = NULL, max.n = NULL, smooth.fun = "hoods2dsmooth",
                 smooth.params = NULL, rule = ">=", verbose = FALSE,
                 ..., time.point = 1, obs = 1, model = 1 )

## Default S3 method:
waveIS(x, th = NULL, J = NULL, wavelet.type = "haar", levels
                 = NULL, max.n = NULL, smooth.fun = "hoods2dsmooth",
                 smooth.params = NULL, rule = ">=", verbose = FALSE,
                 ...)

## S3 method for class 'waveIS'
plot(x, main1 = "X", main2 = "Y",
    which.plots = c("all", "mse", "ss", "energy"),
    level.label = NULL, ...)

## S3 method for class 'waveIS'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x | For waveIS either a list object of class "SpatialVx", a two-component list giving the two fields to be compared (the verification field is assumed to be the first one) or a named list with components "X" and "Xhat" giving the two fields to be compared. list object returned by waveIS. |
| object | list object returned by waveIS. |
| main1,main2 | character giving labels for the plots where main1 refers to the verification field and main2 to the forecast field. |
| which.plots | character vector naming one or more specific plots to do. |
| level.label | optional character vector to use for level names on the plot(s). |
| J | numeric integer giving the number of levels to use. If NULL and the field is dyadic, this will be log2(min(dim(X))), where X is a field from the verification set. If NULL and the field is not dyadic, then J is set equal to 4. Note that if the fields are not dyadic, the function will be much slower. |
| wavelet.type | character giving the name of the wavelet type to use as accepted by dwt.2d and modwt.2d. |
| th | list object with named components "X" and "Xhat" giving the thresholds to use for each field. If null, taken from teh thresholds attribute for "SpatialVx" objects. |

| | |
|---|---|
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| levels | numeric vector giving the successive values of the smoothing parameter. For example, for the default method, these are the neighborhood lengths over which the levels^2 nearest neighbors are averaged for each point. Values should make sense for the specific smoothing function. For example, for the default method, these should be odd integers. |
| max.n | (optional) single numeric giving the maximum neighborhood length to use. Only used if levels are NULL. |
| smooth.fun | character giving the name of a smoothing function to be applied. Default is an average over the n^2 nearest neighbors, where n is taken to be each value of the levels argument. |
| smooth.params | list object containing any optional arguments to smooth.fun. Use NULL if none. |
| rule | If type is "binary", return 0 when the rule applied to a grid point's value is not true in relation to the threshold value, and 1 elsewhere. If type is "replace.below", then return replace.with wherever the rule is not true and return the original value otherwise. By default, it replaces values below the threshold with zero (hence its name), but if rule is, e.g., "<=", then it will replace values above with zero; or whatever value is chosen for replace.with. |
| verbose | logical, should progress information be printed to the screen? |
| ... | Not used by waveIS (or its method functions) or plot.waveIS. Only sort of used by summary.waveIS. One can put the argument silent=TRUE so that nothing is printed to the screen (useful if you just want the values calculated and stored without writing to the screen). |

## Details

This function applies various statistics to the detail fields (in wavelet space) of a discrete wavelet decomposition (DWT) of the binary error fields for a verification set. In particular, the statistics described in Casati et al (2004) and Casati (2010) are calculated. This function depends on the waverify2d or mowaverify2d function (depending on whether the fields are dyadic or not, resp.), which themselves depend on the dwt.2d and idwt.2d or modwt.2d and imodwt.2d functions.

See the references herein and the help files and references therein for dwt.2d and modwt.2d for more information on this approach, as well as Percival and Guttorp (1994) and Lindsay et al. (1996).

## Value

A list object of class "waveIS" that contains the entire prep object passed in by obj, as well as additional components:

| | |
|---|---|
| EnVx,EnFcst | J by q matrices giving the energy for the verification and forecast fields, resp., for each threshold (columns) and scale (rows). |
| MSE,SS | J by q matrices giving the mean square error and IS skill score for each threshold (column) and scale (rows). |

`Bias` numeric vector of length q giving the frequency bias of the original fields for each threshold.

plot.waveIS does not return any value. A plot is created on the current graphic device. summary.waveIS returns a list invisibly with the same components as returned by waveIS along with extra components:

`MSEu,SSu,EnVx.u,EnFcst.u`
length q numeric vectors giving the MSE, SS, and Vx and Fcst energy for each threshold (i.e., ignoring the wavelet decomposition).

`MSEperc,EnVx.perc,EnFcst.perc`
J by q numeric matrices giving percentage form of MSE, Vx Energy and Fcst Energy values, resp.

`EnRelDiff` J by q numeric matrix giving the energy relative difference.

## Author(s)

Eric Gilleland

## References

Casati, B., Ross, G. and Stephenson, D. B. (2004) A new intensity-scale approach for the verification of spatial precipitation forecasts. *Meteorol. Appl.* **11**, 141–154.

Casati, B. (2010) New Developments of the Intensity-Scale Technique within the Spatial Verification Methods Inter-Comparison Project. *Wea. Forecasting* **25**, (1), 113–143, doi:10.1175/2009WAF2222257.1.

Lindsay, R. W., Percival, D. B. and Rothrock, D. A. (1996) The discrete wavelet transform and the scale analysis of the surface properties of sea ice. *IEEE Transactions on Geoscience and Remote Sensing*, **34** (3), 771–787.

Percival, D. B. and Guttorp, P. (1994) Long-memory processes, the Allan variance and wavelets. In *Wavelets in Geophysics*, Foufoula-Georgiou, E. and Kumar, P., Eds., New York: Academic, 325–343.

## See Also

`IS`, `int.scale.verify` from package **verification**,

`dwt.2d`, `modwt.2d`, `idwt.2d`, `imodwt.2d`, `hoods2d`

`thresholder`

## Examples

```
data( "UKobs6" )
data( "UKfcst6" )
data( "UKloc" )

hold <- make.SpatialVx( UKobs6, UKfcst6,
    thresholds = c(0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50),
    loc = UKloc, map = TRUE, field.type = "Rainfall", units = "mm/h",
    data.name = "Nimrod", obs.name = "UKobs6", model.name = "UKfcst6" )
```

```
look <- waveIS(hold, J=8, levels=2^(8-1:8), verbose=TRUE)
plot(look, which.plots="mse")
plot(look, which.plots="ss")
plot(look, which.plots="energy")
summary(look)

## Not run:
data( "pert004" )
data( "pert000" )

hold <- make.SpatialVx( pert000, pert004, thresholds = c(1, 10, 50),
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Perturbed ICP", obs.name = "pert000", model.name = "pert004" )

look <- waveIS(hold, levels=1:4, verbose=TRUE)
plot(look, which.plots="mse")
plot(look, which.plots="ss")
plot(look, which.plots="energy")
summary(look)


## End(Not run)
```

---

wavePurifyVx *Apply Traditional Forecast Verification After Wavelet Denoising*

---

## Description

Apply traditional forecast verification after wavelet denoising ala Briggs and Levine (1997).

## Usage

```
wavePurifyVx( x, climate = NULL, which.stats = c("bias",
    "ts", "ets", "pod", "far", "f", "hk", "mse"), thresholds = NULL,
    rule = ">=", return.fields = FALSE, verbose = FALSE, ...)

## S3 method for class 'SpatialVx'
wavePurifyVx( x, climate = NULL, which.stats = c("bias",
    "ts", "ets", "pod", "far", "f", "hk", "mse"), thresholds = NULL,
    rule = ">=", return.fields = FALSE, verbose = FALSE, ...,
    time.point = 1, obs = 1, model = 1 )

## Default S3 method:
wavePurifyVx( x, climate = NULL, which.stats = c("bias",
    "ts", "ets", "pod", "far", "f", "hk", "mse"), thresholds = NULL,
    rule = ">=", return.fields = FALSE, verbose = FALSE, ...)
```

```
## S3 method for class 'wavePurifyVx'
plot(x, ..., col = c("gray", tim.colors(64)), zlim, mfrow,
                horizontal = TRUE, type = c("stats", "fields") )

## S3 method for class 'wavePurifyVx'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| x | For wavePurifyVx, either a list object of class "SpatialVx", or a list with only two components consisting of m by n matrices giving the verification and forecast fields, resp., or a list with named components "X" and "Xhat".<br>For plot.wavePurifyVx, list object as output from wavePurifyVx. |
| object | list object as returned by wavePurifyVx. |
| climate | m by n matrix defining a climatology field. If not NULL, then the anamoly correlation coefficient will be applied to the wavelet denoised fields. |
| which.stats | character describing which traditional verification statistics to calculate on the wavelet denoised fields. This is the argument passed to the argument of the same name in vxstats. |
| thresholds | Either a numeric vector or a list with components named "X" and "Xhat" giving thresholds used to define events for all of the verification statistics except MSE. However, if supplied or other statistics are to be computed, then MSE will be calculated for the fields at values >= thresholds. If only MSE is to be computed, and thresholds is NULL, then no thresholding is applied. If NULL, and any of the statistics besides MSE are to be calculated, then default values of the 0, 0.1, 0.25, 0.33, 0.5, 0.66, 0.75, 0.9 and 0.95 quantiles (for each field, so that the thresholds may differ between fields) are used. The same holds for anamoly correlation coefficient. The exception is that if the argument is null and x is a "SpatialVx" class object, then the thresholds are taken to be those associated with the "thresholds" attribute of this object. |
| rule | character string giving the threshold rule to be applied. See help file for thresholder function for more information. |
| return.fields | logical, should the denoised fields be returned (e.g., for subsequent plotting)? |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| verbose | logical, should progress information (including total run time) be printed to the screen? |
| col, zlim, horizontal | optional arguments to image, and/or **fields** functions poly.image and image.plot |
| mfrow | optionally set the plotting panel via mfrow (see help file for par). Default sets a rgion that will show all plots in one set of panels. |
| type | character string stating whether to plot the resulting statistics or the original fields along with their de-noised counter parts. |

`...` For `wavePurifyVx`, optional additional arguments to `denoise.dwt.2d` (or `denoise.modwt.2d`) from package **waveslim**. Note that if the argument `J` is not passed, then it will be determined as J=log2(min(m,n)). If the fields are dyadic, then the usual DWT is used, otherwise the maximal overlap DWT is used instead. For the plot and summary method functions, these are not used. Also passed to `poly.image` and `image.plot` for plotting routine for "fields" type plots when the "maps" attribute from the "SpatialVx" object is TRUE.

## Details

If the fields are dyadic, then the `denoise.dwt.2d` function from package **waveslim** is applied to each field before calculating the chosen verification statistics. Otherwise `denoise.modwt.2d` from the same package is used. The result is that high-frequency fluctuations in the two fields are removed before calculating verification statistics so that the resulting statistics are less susceptible to small-scale errors (see Briggs and Levine, 1997). See Percival and Guttorp (1994) and Lindsay et al. (1996) for more on this type of wavelet analysis including maximal overlap DWT.

## Value

A list object of class "wavePurifyVx" is returned with possible components (depending on what is supplied in the arguments, etc.):

`X2,Y2` m by n matrices of the denoised verification and forecast fields, resp. (only if return.fields is TRUE).

`thresholds` q by 2 matrix of thresholds applied to the forecast (first column) and verification (second column) fields, resp. If climate is not NULL, then the same thresholds for the forecast field are applied to the climatology.

`qs` If object and thresholds are NULL, and statistics other than MSE or ACC are desired, then this will be created along with the thresholds, and is just a character version of the trhesholds.

`args` list object containing all the optional arguments passed into ..., and the value of J used (e.g., even if not passed into ...).

`bias,ts,ets,pod,far,f,hk,mse,acc`
numeric vectors of length q (i.e., the number of thresholds) giving the associated verification statistics.

## Author(s)

Eric Gilleland

## References

Briggs, W. M. and Levine, R. A. (1997) Wavelets and field forecast verification. *Mon. Wea. Rev.*, **125**, 1329–1341.

Lindsay, R. W., Percival, D. B. and Rothrock, D. A. (1996) The discrete wavelet transform and the scale analysis of the surface properties of sea ice. *IEEE Transactions on Geoscience and Remote Sensing*, **34** (3), 771–787.

Percival, D. B. and Guttorp, P. (1994) Long-memory processes, the Allan variance and wavelets. In *Wavelets in Geophysics*, Foufoula-Georgiou, E. and Kumar, P., Eds., New York: Academic, pp. 325–343.

## See Also

denoise.dwt.2d, denoise.modwt.2d, waverify2d, mowaverify2d, waveIS, vxstats

## Examples

```
grid <- list( x= seq( 0,5,,100), y= seq(0,5,,100))
obj <- Exp.image.cov( grid=grid, theta=.5, setup=TRUE)
look <- sim.rf( obj)
look[ look < 0] <- 0
look <- zapsmall( look)

look2 <- sim.rf( obj)
look2[ look2 < 0] <- 0
look2 <- zapsmall( look2)

look3 <- sim.rf( obj)
look3[ look3 < 0] <- 0
look3 <- zapsmall( look3)

hold <- make.SpatialVx( look, look2, thresholds = c(0.1, 1),
            field.type = "random", units = "units")

plot( hold )

res <- wavePurifyVx( hold, climate = look3, return.fields = TRUE, verbose = TRUE )

plot(res, type="fields")
plot(res, type="stats")

summary(res)

## Not run:
data( "UKobs6" )
data( "UKfcst6" )
data( "UKloc" )

hold <- surrogater2d( UKobs6, n=1, maxiter=50, verbose=TRUE)
hold <- matrix(hold, 256, 256)

UKobj <- make.SpatialVx( UKobs6, UKfcst6, thresholds = c(0.1, 2, 5, 10),
    loc = UKloc, map = TRUE, field.type = "Rainfall", units = "mm/h",
    data.name = "Nimrod", obs.name = "obs 6", model.name = "fcst 6" )

plot(UKobj )

look <- wavePurifyVx( object = UKobj, climate = hold,
    return.fields = TRUE, verbose = TRUE)
```

```
plot(look, type = "fields" )
plot(look, type = "stats" )

summary( look )

data( "pert004" )
data( "pert000" )

hold <- make.SpatialVx( pert000, pert004, thresholds = c(1, 10, 50),
    loc = ICPg240Locs, projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "Perturbed ICP Cases", obs.name = "pert000",
    model.name = "pert004" )

plot( hold )

look <- wavePurifyVx( hold, return.fields = TRUE, verbose = TRUE )

plot(look, type = "fields" )
plot(look, type = "stats" )

summary( look )

## End(Not run)
```

---

waverify2d *High-Resolution Gridded Forecast Verification Using Discrete Wavelet Decomposition*

---

## Description

High-resolution gridded forecast verification using discrete wavelet decomposition.

## Usage

```
waverify2d(X, ..., Clim = NULL, wavelet.type = "haar", J = NULL)

## Default S3 method:
waverify2d(X, ..., Y, Clim = NULL, wavelet.type = "haar", J =
                  NULL, useLL = FALSE, compute.shannon = FALSE,
                  which.space = "field", verbose = FALSE)

## S3 method for class 'SpatialVx'
waverify2d(X, ..., Clim = NULL, wavelet.type = "haar", J = NULL,
    useLL = FALSE, compute.shannon = FALSE, which.space = "field",
    time.point = 1, obs = 1, model = 1, verbose = FALSE)
```

```
mowaverify2d(X, ..., Clim = NULL, wavelet.type = "haar", J = 4)

## Default S3 method:
mowaverify2d(X, ..., Clim = NULL, Y, wavelet.type = "haar", J = 4,
   useLL = FALSE, compute.shannon = FALSE, which.space = "field", verbose = FALSE)

## S3 method for class 'SpatialVx'
mowaverify2d(X, ..., Clim = NULL, wavelet.type = "haar", J = 4,
    useLL = FALSE, compute.shannon = FALSE, which.space = "field",
    time.point = 1, obs = 1, model = 1, verbose = FALSE)

## S3 method for class 'waverify2d'
plot(x, ..., main1 = "X", main2 = "Y", main3 = "Climate",
    which.plots = c("all", "dwt2d", "details", "energy", "mse",
        "rmse", "acc"), separate = FALSE, col, horizontal = TRUE)

## S3 method for class 'waverify2d'
print(x, ...)
```

## Arguments

| | |
|---|---|
| X,Y,Clim | m by n dyadic matrices (i.e., m = 2^M and n = 2^N, for M, N some integers) giving the verification and forecast fields (and optionally a climatology field), resp. Alternatively, X may be a "SpatialVx" object, in which case, Y is not given and in either case Clim must be provided if it is to be used. |
| x | list object of class "waverify2d" as returned by waverify2d. |
| wavelet.type | character naming the type of wavelet to be used. This is given as the wf argument to the dwt.2d function of package **waveslim**. |
| J | (optional) numeric integer giving the pre-determined number of levels to use. If NULL, J is set to be log2(m) = M in waverify2d only. |
| useLL | logical, should the LL submatrix (i.e., the father wavelet or grand mean) be used to find the inverse DWT's for calculating the detail fields? |
| compute.shannon | logical, should the Shannon entropy be calculated for the wavelet decomposition? |
| which.space | character (one of "field" or "wavelet") naming from which space the detail fields should be used. If "field", then it is in the original field (or image) space (i.e., the detail reconstruction), and if "wavelet", it will be done in the wavelet space (i.e., the detail wavelet coefficients). |
| time.point | numeric or character indicating which time point from the "SpatialVx" verification set to select for analysis. |
| obs, model | numeric indicating which observation/forecast model to select for the analysis. |
| main1,main2,main3 | optional characters naming each field to be used for the detail field plots and legend labelling on the energy plot. |

| | |
|---|---|
| `which.plots` | character vector describing which plots to make. The default is to make all of them. "dwt2d" option uses `plot.dwt2d` from **waveslim**. "details" option makes image plots of the detail fields on which the various statistics are calculated. The rest of the options give line plots showing the statistics. |
| `separate` | logical, should the plots be on their own devices (TRUE) or should some of them be put onto a single multi-panel device (FALSE, default)? |
| `col` | optional argument specifying the `col` argument in calls to functions like `image`. Default is a concatenation of gray with `time.colors(64)`. |
| `horizontal` | logical, should the legend on image plots be horizontal (TRUE, placed at the bottom of the plot) or vertical (FALSE, placed at the right side of the plot)? |
| `verbose` | logical, should progress information be printed to the screen, including total run time? |
| `...` | optional additonal plot or image.plot parameters. If detail and energy, mse, rmse or acc plots are desired, must be applicable to both types of plots. Not used by `print` method function. |

## Details

This is a function to use discrete wavelet decomposition to analyze verification sets along the lines of Briggs and Levine (1997), as well as Casati et al. (2004) and Casati (2009). In the originally proposed formulation of Briggs and Levine (1997), continuous verification statistics (namely, the anomaly correlation coefficient (ACC) and root mean square error (RMSE)) are calculated for detail fields obtained from wavelet decompositions of each of a forecast and verification field (and for ACC a climatology field as well). Casati et al. (2004) introduced an intensity scale approach that applies 2-d DWT to binary (obtained from thresholding) difference fields (Forecast - Verification), and applying a skill score at each level based on the mean square error (MSE). Casati (2009) extended this idea to look at the energy at each level as well.

This function makes use of the `dwt.2d` and `idwt.2d` functions from package **waveslim**, and `plot.waverify2d` uses the `plot.dwt.2d` function if `dwt2d` is selected through the `which.plots` argument. See the help file for these functions, the references therein and the references herein for more on these approaches.

Generally, it is not necessary to use the father wavelet for the detail fields, but for some purposes, it may be desired.

`mowaverify2d` is very similar to `waverify2d`, but it allows fields to be non-dyadic (and may subsequently be slower). It uses the `modwt.2d` and `imodwt.2d` functions from the package **waveslim**. In particular, it performs a maximal overlap discrete wavelet transform on a matrix of arbitrary dimension. See the help file and references therein for `modwt.2d` for more information, as well as Percival and Guttorp (1994) and Lindsay et al. (1996).

In Briggs and Levine (1997), they state that the calculations can be done in either the data (called field here) space or the wavelet space, and they do their examples in the field space. If the wavelets are orthogonal, then the detail coefficeints (wavelet space), can be analyzed with the assumption that they are independent; whereas in the data space, they typically cannot be assumed to be independent. Therefore, most statistical tests should be performed in the wavelet space to avoid issues arising from spatial dependence.

## Value

A list object of class "waverify2d" with components:

| | |
|---|---|
| `J` | single numeric giving the number of levels. |
| `X.wave, Y.wave, Clim.wave` | objects of class "dwt.2d" describing the wavelet decompositions for the verification and forecast fields (and climatology, if applicable), resp. (see the help file for dwt.2d from package waveslim for more about these objects). |
| `Shannon.entropy` | numeric matrix giving the Shannon entropy for each field. |
| `energy` | numeric matrix giving the energy at each level and field. |
| `mse,rmse` | numeric vectors of length J giving the MSE/RMSE for each level between the verification and forecast fields. |
| `acc` | If a climatology field is supplied, this is a numeric vector giving the ACC for each level. |

## Author(s)

Eric Gilleland

## References

Briggs, W. M. and Levine, R. A. (1997) Wavelets and field forecast verification. *Mon. Wea. Rev.*, **125**, 1329–1341.

Casati, B., Ross, G. and Stephenson, D. B. (2004) A new intensity-scale approach for the verification of spatial precipitation forecasts. *Meteorol. Appl.* **11**, 141–154.

Casati, B. (2010) New Developments of the Intensity-Scale Technique within the Spatial Verification Methods Inter-Comparison Project. *Wea. Forecasting* **25**, (1), 113–143, doi:10.1175/2009WAF2222257.1.

Lindsay, R. W., Percival, D. B. and Rothrock, D. A. (1996) The discrete wavelet transform and the scale analysis of the surface properties of sea ice. *IEEE Transactions on Geoscience and Remote Sensing*, **34** (3), 771–787.

Percival, D. B. and Guttorp, P. (1994) Long-memory processes, the Allan variance and wavelets. In *Wavelets in Geophysics*, Foufoula-Georgiou, E. and Kumar, P., Eds., New York: Academic, pp. 325–343.

## See Also

`dwt.2d`, `idwt.2d`, `hoods2d`

## Examples

```
grid<- list( x= seq( 0,5,,64), y= seq(0,5,,64))
obj<-Exp.image.cov( grid=grid, theta=.5, setup=TRUE)
look<- sim.rf( obj)
look[ look < 0] <- 0
look <- zapsmall( look)
```

```
look2 <- sim.rf( obj)
look2[ look2 < 0] <- 0
look2 <- zapsmall( look2)

res <- waverify2d(look, Y=look2)
plot(res)
summary(res)

## Not run:
data( "UKobs6" )
data( "UKfcst6" )

look <- waverify2d(UKobs6, Y=UKfcst6)

plot(look, which.plots="energy")
look2 <- mowaverify2d(UKobs6, UKfcst6, J=8)
plot(look2, which.plots="energy")

plot(look, main1="NIMROD Analysis", main2="NIMROD Forecast")

plot(look2, main1="NIMROD Analysis", main2="NIMROD Forecast")

# Alternative using "SpatialVx" object.
data( "UKloc" )

hold <- make.SpatialVx( UKobs6, UKfcst6, loc = UKloc,
    map = TRUE, field.type = "Rainfall", units = "mm/h",
    data.name = "Nimrod", obs.name = "Obs 6",
    model.name = "Fcst 6" )

look <- waverify2d(hold)

plot(look, which.plots="details")

data( "pert000" )
data( "pert004" )

# The following is slow, but does not require fields to be dyadic.
look <- mowaverify2d(pert000, Y=pert004, J=8, verbose=TRUE)

# Also can just do plot(look), but should print to a pdf file (e.g., using pdf()).
plot(look, which.plots="energy")

# Using a "SpatialVx" object.
data( "ICPg240Locs" )

hold <- make.SpatialVx( pert000, pert004, loc = ICPg240Locs,
    projection = TRUE, map = TRUE, loc.byrow = TRUE,
    field.type = "Precipitation", units = "mm/h",
    data.name = "ICP Perturbed Cases", obs.name = "pert000",
    model.name = "pert004" )

look <- mowaverify2d( hold, verbose = TRUE )
```

```
plot(look, which.plots = "details")

# Try one with some kind of climatology field.  Here using surrogater2d function.
hold <- surrogater2d(UKobs6, n=1, maxiter=50, verbose=TRUE)
hold <- matrix(hold, 256, 256)
image(hold, col=c("grey",tim.colors(64)), axes=FALSE)
image.plot(UKloc, col=c("grey",tim.colors(64)), legend.only=TRUE, horizontal=TRUE)

look <- waverify2d(UKobs6, Y=UKfcst6, Clim=hold)

plot(look)


## End(Not run)
```

# Index