# Package 'smoothie'

February 20, 2015

**Version** 1.0-1

**Date** 2013-06-01

**Title** Two-dimensional Field Smoothing

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

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

**Depends** R (>= 3.0.0)

**Suggests** SpatialVx, fields

**Description** Functions to smooth two-dimensional fields using FFT and the convolution theorem

**License** GPL (>= 2)

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

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

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2013-07-06 07:13:34

## R topics documented:

smoothie-package *Two-dimensional Field Smoothing*

## Description

**smoothie** contains code originally contained as part of the package, **SpatialVx**; a package for performing weather forecast verification spatially. However, the code is potentially useful for much wider purposes than spatial weather forecast verification. It contains functions to perform convolution smoothing using several different types of kernels.

## Details

The manual for this package is given by Gilleland (2013).

Primary functions include:

`Fourier2d`, `kernel2dsmooth` and `kernel2dmeitsjer`

As well as the following wrapper functions, which can be useful in the context of having functions that take the same set of arguments (e.g., with the smoothing parameter as the second argument with the name `lambda`) for ease of allowing a user to supply their own desired kernel to a function as is utilized, for example, in the **SpatialVx** package.

`hoods2dsmooth` (neighborhood or boxcar kernel)

`gauss2dsmooth` (Gaussian kernel)

`disk2dsmooth` (Disk kernel)

`identity2dsmooth` (No smoothing, just returns the field)

See their help files for more information.

The functions utilize the convolution theorem along with the Fast Fourier Transform (FFT) to smooth the field (Hastie and Tibshirani, 1990; Souza, 2010)

## Author(s)

Eric Gilleland

## References

Gilleland, E. (2013) Two-dimensional kernel smoothing: Using the R package smoothie. *NCAR Technical Note*, TN-502+STR, 17pp. Available at: http://opensky.library.ucar.edu/collections/TECH-NOTE-000-000-000-869

Hastie, T. J. and Tibshirani, R. J. (1990) *Generalized Additive Models*. Chapman \& Hall/CRC Monographs on Statistics and Applied Probability 43, 335pp.

Souza, C. R. (2010) *Kernel Functions for Machine Learning Applications*. 17 Mar 2010. Web. http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html.

## Examples

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

`Fourier2d` *Fast Fourier Transform (FFT) of a 2-d Gridded Field*

## Description

Function to compute the Fast Fourier Transform (FFT) of a gridded field. The field is first expanded, and zero padded.

## Usage

```
Fourier2d(x, bigdim = NULL, kdim = NULL)
```

## Arguments

| | |
|---|---|
| `x` | n by m matrix giving the gridded field to transform. |
| `bigdim` | (optional) numeric vector of length 2. If NULL, it will be found from kdim. One of this argument or kdim must be given. |
| `kdim` | (optional) numeric vector of length 2 giving the dimension of the kernel to be applied. If bigdim is NULL, then the optimal dimension for the FFT is calculated from the dimension of x and kdim. |

## Details

The gridded field is expanded to bigdim, which is the nearest power of 2 above N=dim( x) + kdim - 1 if N <= 1024, or to the nearest multiple of 512 otherwise, if bigdim is not supplied. `x` is placed in the upper left corner of a matrix of zeros that has dimension bigdim. Missing values are replaced by zero. The FFT is conducted on the expanded/zero-padded gridded field. This is intended to be used more internally for the `kernel2dsmooth` function in order to reduce the number of FFT's that need be calculated (e.g., if performed for the same threshold over several neighborhood lengths). It is currently not used by SpatialVx function anymore, but may still be usefull.

## Value

A possibly complex matrix of size bigdim of the Fourier transformed gridded field is returned.

## Author(s)

Eric Gilleland

## See Also

`fft`, `kernel2dsmooth`, `kernel2dmeitsjer`

## Examples

```
look <- matrix( 0, 10, 12)
look[4,7] <- 1
lookFFT <- Fourier2d( look, kdim=c(3,3))
# Now, call 'kernel2dsmooth' with a neighborhood boxcar kernel that averages the
# nearest grid squares (i.e., neighborhood length of 3).  That is, a square
# of 1/(3^2) = 1/9 ~ 0.111111 with length 3 surrounding the 1 in 'look'.
kernel2dsmooth( look, kernel.type="boxcar", n=3, X=lookFFT)

# Note that the above could have been done with just:
kernel2dsmooth( look, kernel.type="boxcar", n=3)
# But, in the previous one, one less FFT was calculated.
```

---

# hoods2dsmooth *Wrapper Functions for kernel2dsmooth*

---

## Description

Wrapper functions for kernel2dsmooth to perform specific convolution smooths of 2-d fields.

## Usage

```
hoods2dsmooth(x, lambda, W = NULL, setup = FALSE, ...)
gauss2dsmooth(x, lambda, W = NULL, setup = FALSE, ...)
disk2dsmooth(x, lambda, W = NULL, setup = FALSE, ...)
identity2dsmooth(x, lambda = 0, W = NULL, setup = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | numeric matrix giving the field to be smoothed. |
| lambda | single numeric giving the smoothing parameter. In the case of hoods2dsmooth, this is the neighborhood length so that the result is that each point is an average over the nearest lambda^2 neighbors. For gauss2dsmooth, this is the sigma parameter, and for disk2dsmooth, this is the radius of the disk. Not used by identity2dsmooth. |
| W | (optional) if the FFT of the kernel weights have already been calculated for the smooth, they can be passed here. Not used by identity2dsmooth. |
| setup | logical, should only the FFT of the kernel weights be returned (i.e., instead of the smoothed x field)? Should not be used if W is supplied, or you may not get what you want (i.e., precedence is given to W's being supplied). Not used by identity2dsmooth. |
| ... | optional arguments to the specific kernel type in the call to kernel2dsmooth, or really, to kernel2dmeitsjer. Not used by identity2dsmooth. Note that for gauss2dsmooth, both nx and ny must be provided. |

## Details

This is a wrapper function to `kernel2dsmooth`. See its help file for more details. These functions can be useful for functions that allow a user to smooth a field with a choice of smoothing functions. Makes use of the convolution theorem with the fast Fourier transform for computational efficiency (Ritter and Wilson, 2001; Barrett and Myers, 2004). Missing values are set to zero, and borders are zero-padded to an optimal amount. See Hastie and Tibshirani (1990) for smoothing functions in the context of statistical analysis.

`hoods2dsmooth` is a wrapper to `kernel2dsmooth` that employs the boxcar kernel with neighborhood length as the smoothing parameter, `lambda`. This is the type of kernel smoothing, for example, proposed by Roberts and Lean (2008) and used in Ebert (2008) in the context of spatial weather forecast verification (and used by the **SpatialVx** package). The smoothing parameter lambda should be an odd positive integer (though it need not be an actual integer recognized by R). If it is not, then one of several things will happen depending on its value, and a warning is generated. If it is not an integer (i.e., if floor(lambda) != lambda), the floor of lambda is taken. If it is even (possibly after flooring), one is subtracted from it. If it is less than one (possibly after flooring and/or subtracting one), it is set to one (note that if lambda = 1, the field x is returned untouched).

`gauss2dsmooth` is a wrapper to `kernel2dsmooth` that employs the Gaussian kernel with smoothing parameter, `lambda`, equal to the `sigma` parameter. This is the type of smoothing applied originally for the practically perfect hindcast method in the context of spatial weather forecast verification (see Ebert, 2008). If `W` is not specified, then one of either `h` or both `nx` and `ny` must be given (see the help file for `kernel2dmeitsjer`).

`disk2dsmooth` is a wrapper to `kernel2dsmooth` that calls the disk kernel with smoothing parameter `r` (the radius of the disk). This is the type of smoothing applied in Davis et al. (2006a, 2006b) in the context of feature-based spatial weather forecast verification.

`identity2dsmooth` simply returns the field without smoothing it. Provided for convenience.

## Value

If `W` is not supplied and `setup` is TRUE, then a matrix is returned with dimensions chosen to optimize computational efficiency. Otherwise, a matrix of the same dimension as `x` is returned giving the smoothed version of the field.

## Author(s)

Eric Gilleland

## References

Barrett, H. H. and Myers, K. J. (2004) *Foundations of Image Science*. Wiley Series in Pure and Applied Optics, Editor: B. E. A. Saleh, Hoboken, New Jersey, 1540pp.

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.

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. Available at http://www.ecmwf.int/newsevents/meetings/workshops/2007/jwgv/METspecialissueemail.pdf

Hastie, T. J. and Tibshirani, R. J. (1990) *Generalized Additive Models*. Chapman \& Hall/CRC Monographs on Statistics and Applied Probability 43, 335pp.

Ritter, G. X. and Wilson, J. N. (2001) *Handbook of Computer Vision Algorithms in Image Algebra*. 2nd Edition, CRC Press, Boca Raton, Florida, U.S.A., 417pp.

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

kernel2dsmooth, kernel2dmeitsjer, fft

## Examples

```
x <- y <- matrix( 0, 50, 50)
x[ sample(1:50,10), sample(1:50,10)] <- rexp( 100, 0.25)
y <- disk2dsmooth( x=x, lambda=6.5)
x <- gauss2dsmooth( x=x, lambda=3, nx=25, ny=25)
## Not run:
##
## The following examples are specific to the SpatialVx package.
##
par( mfrow=c(1,2))
image.plot( x, col=tim.colors(256))
image.plot( y, col=tim.colors(256))

hold <- make.SpatialVx(x, y, thresholds=c(0.1, 0.5))
look <- hoods2d( hold, which.methods=c("fss"), levels=c(1, 3, 20),
        smooth.fun="gauss2dsmooth", smooth.params=list(nx=601, ny=501))
plot( look)

data(pert000)
data(pert004)
data(ICPg240Locs)
# Do the neighborhood methods with averaging performed over
# a radius instead of the lambda^2 nearest neighbors.
# The smoothing parameters are determined by the levels argument,
# and the others are passed via smooth.params.
hold <- make.SpatialVx( pert000, pert004, thresholds=c(1,2,5,10,20,50),
            loc=ICPg240Locs, projection=TRUE, map=TRUE,
            field.type="Precipitation", units="mm/h",
            data.name=c("ICP Fake", "pert000", "pert004"))

look <- hoods2d(hold, levels=c(1, 3, 9, 17, 33, 65, 129, 257),
            smooth.fun="disk2dsmooth", verbose=TRUE)
look

## End(Not run)
```

kernel2dmeitsjer *Create a Kernel Matrix*

## Description

Create a kernel matrix to be used in a 2-D convolution smooth (e.g., using kernel2dsmooth).

## Usage

```
kernel2dmeitsjer(type = "gauss", ...)
```

## Arguments

| | |
|---|---|
| type | character name of the kernel to be created. |
| ... | Other arguments to the specific kernel type. See Details section below. |

## Details

The specific types of kernels that can be made are as follows. In each case, h=||x-x_c|| is the distance from the center of the kernel. Every kernel that requires single numerics nx and ny be specified returns an nx by ny metrix. Distances h are found by setting up a grid based on 1 to nx and 1 to ny denoting the points as (xgrid, ygrid), finding the center of the grid as (x.center, y.center)=(nx/2,ny/2), and then h = sqrt( (xgrid - x.center)^2 + (ygrid - y.center)^2). For kernels that better reflect distance (e.g., using great-circle distance, anisotropic distances, etc.), the matrix h can be passed instead of nx and ny, but only for those kernels that take h as an argument. In each case with sigma as an argument, sigma is the smoothing parameter. There are many kernel functions allowed here, and not all of them make sense for every purpose.

"average" gives a kernel that will give an average of the nearest neighbors in each direction (can take an average grid points further in the x- direction than the y-direction, and vice versa). Requires that nx and ny be specified, and the resulting kernel is defined by an nx by ny matrix with each element equal to 1/(nx*ny). If nx = ny, then the result is the same as the boxcar kernel below.

"boxcar" the boxcar kernel is an n by n matrix of 1/n^2. This results in a neighborhood smoothing when used with kernel2dsmooth giving the type of smoothed fields utilized, e.g., in Roberts and Lean (2008) and Ebert (2008). Requires that n be specified. Note that usually the boxcar is a square matrix of ones, which gives the sum of the nearest n^2 grid points. This gives the average.

"cauchy" The Cauchy kernel is given by K(sigma)=1/(1+h^2/sigma). Requires the arguments nx, ny and sigma. See Souza (2010) for more details.

"disk" gives a circular averaging (pill-box) kernel (aka, disk kernel). Similar to "average" or "boxcar", but this kernel accounts for a specific distance in all directions from the center (i.e., an average of grid squares within a circular radius of the central point). This results in the convolution radius smoothing applied in Davis et al. (2006a,2006b). Requires that r (the desired radius) be supplied, and a square matrix of appropriate dimension is returned.

"epanechnikov" The Epanechnikov kernel is defined by max(0, 3/4*(1-h/(sigma^2))). See, e.g., Hastie and Tibshirani (1990). Requires arguments nx, ny, and sigma.

"exponential" The exponential kernel is given by K(sigma) = a*exp(-h/(2*sigma)). Requires the arguments nx, ny and sigma, and optionally takes the argument a (default is a=1). An nx by ny matrix is returned. See Souza (2010) for more details.

"gauss" The Gaussian kernel defined by K(sigma) = 1/(2*pi*sigma^2)*exp(-h/(2*sigma)). Requires the arguments nx, ny and sigma be specified. The convolution with this kernel results in a Gaussian smoothed field as used in the practically perfect hindcast method of Brooks et al. (1998) (see also Ebert 2008) and studied by Sobash et al (2011) for spatial forecast verification purposes. Returns an nx by ny matrix.

"laplacian" Laplacian Operator kernel, which gives the sum of second partial derivatives for each direction. It is often used for edge detection because it identifies areas of rapid intensity change. Typically, it is first applied to a field that has been smoothed first by a Gaussian kernel smoother (or an approximation thereof; cf. type "LoG" below). This method optionally the parameter alpha, which controls the shape of the Laplacian kernel, which must be between 0 and 1 (inclusive), or else it will be set to 0 (if < 0) or 1 (if > 1). Returns a 3 by 3 kernel matrix.

"LoG" Laplacian of Gaussian kernel. This combines the Laplacian Operator kernel with that of a Gaussian kernel. The form is given by K(sigma) = -1/(pi*sigma^4)*exp(-h/(2*sigma^2))*(1-h/(2*sigma^2)). Requires the arguments nx, ny and sigma be specified. Returns an nx by ny matrix.

"minvar" A minimum variance kernel, which is given by 3/8*(3 - 5*h/sigma^2) if h <= 1, and zero otherwise (see, e.g., Hastie and Tibshirani, 1990). Requires the arguments nx, ny, and sigma be specified. Returns an nx by ny matrix.

"multiquad" The multiquadratic kernel is similar to the rational quadratic kernel, and is given by K(a) = sqrt(h + a^2). The inverse is given by 1/K(a). Requires the arguments nx, ny and a be specified. Optionally takes a logical named inverse determining whether to return the inverse multiquadratic kernel or not.

"prewitt" Prewitt filter kernel, which emphasizes horizontal (vertical) edges through approximation of a vertical (horizontal) gradient. Optionally takes a logical argument named transpose, which if FALSE (default) emphasis is on horizontal, and if TRUE emphasis is on vertical. Returns a 3 by 3 matrix whose first row is all ones, second row is all zeros, and third row is all negative ones for the transpose=FALSE case, and the transpose of this matrix in the transpose=TRUE case.

"power" The power kernel is defined by K(p) = -h^p. The log power kernel is similarly defined as K(p) = -log(h^p+1). Requires specification of the arguments nx, ny and p. Alternatively takes the logical do.log to determine whether the log power kernel should be returned (TRUE) or not (FALSE). Default if not passed is to do the power kernel. Returns an nx by ny matrix. See Souza (2010) for more details.

"radial" The radial kernel is returns a*|h|^(2*m-d)*log(|h|) if d is even and a*|h|^(2*m-d) otherwise. Requires arguments a, m, d nx and ny. Replaces any missing values with zero.

"ratquad" The rational quadratic kernel is used as an alternative to the Gaussian, and is given by K(a) = 1 - h/(h+a). Requires the arguments nx, ny and a, and returns an nx by ny matrix. See Souza (2010) for more details.

"sobel" Same as prewitt, except that the elements 1,2 and 3,2 are replaced by two and neative two, resp.

"student" The generalized Student's t kernel is defined by K(p)=1/(1+h^p). Requires the arguments nx, ny and p be specified. Returns an nx by ny matrix. See Souza (2010) for more details.

"unsharp" Unsharp contrast enhancement filter. This is simply given by a 3 by 3 matrix of al zeros, except for a one in the center subtracted by a laplacian operator kernel matrix. Requires the same

arguments as for "laplacian". Returns a 3 by 3 matrix.

"wave" The wave kernel is defined by K(phi) = phi/h * sin( h/phi). Requires arguments nx, ny and phi be specified. Returns an nx by ny matrix.

## Value

matrix of dimension determined by the specific type of kernel, and possibly user passed arguments giving the kernel to be used by `kernel2dsmooth`.

## Author(s)

Eric Gilleland

## References

Brooks, H. E., Kay, M., and Hart, J. A. (1998) Objective limits on forecasting skill of rare events. *19th Conf. Severe Local Storms*. Amer. Met. Soc., 552–555.

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.

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. Available at http://www.ecmwf.int/newsevents/meetings/workshops/2007/jwgv/METspecialissueemail.pdf

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.

Sobash, R. A., Kain, J. S. Bright, D. R. Dean, A. R. Coniglio, M. C. and Weiss, S. J. (2011) Probabilistic forecast guidance for severe thunderstorms based on the identification of extreme phenomena in convection-allowing model forecasts. *Wea. Forecasting*, **26**, 714–728.

Souza, C. R. (2010) *Kernel Functions for Machine Learning Applications*. 17 Mar 2010. Web. http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html.

## See Also

`fft`, `kernel2dsmooth`

## Examples

```
x <- matrix( 0, 10, 12)
x[4,5] <- 1
kmat <- kernel2dmeitsjer( "average", nx=7, ny=5)
kernel2dsmooth( x, K=kmat)

##
```

```
## Can also call 'kernel2dsmooth' directly.
##
kernel2dsmooth( x, kernel.type="boxcar", n=5)
kernel2dsmooth( x, kernel.type="cauchy", sigma=20, nx=10, ny=12)
kernel2dsmooth( x, kernel.type="disk", r=3)
kernel2dsmooth( x, kernel.type="epanechnikov", nx=10, ny=12, sigma=4)
kernel2dsmooth( x, kernel.type="exponential", a=0.1, sigma=4, nx=10, ny=12)
kernel2dsmooth( x, kernel.type="gauss", nx=10, ny=12, sigma=4)
kernel2dsmooth( x, kernel.type="laplacian", alpha=0)
kernel2dsmooth( x, kernel.type="LoG", nx=10, ny=12, sigma=1)
kernel2dsmooth( x, kernel.type="minvar", nx=10, ny=12, sigma=4)
kernel2dsmooth( x, kernel.type="multiquad", a=0.1, nx=10, ny=12)
kernel2dsmooth( x, kernel.type="power", p=0.5, nx=10, ny=12)
kernel2dsmooth( x, kernel.type="prewitt")
kernel2dsmooth( x, kernel.type="prewitt", transpose=TRUE)
kernel2dsmooth( x, kernel.type="radial", a=1, m=2, d=1, nx=10, ny=12)
kernel2dsmooth( x, kernel.type="ratquad", a=0.1, nx=10, ny=12)
kernel2dsmooth( x, kernel.type="sobel")
kernel2dsmooth( x, kernel.type="sobel", transpose=TRUE)
kernel2dsmooth( x, kernel.type="student", p=1.5, nx=10, ny=12)
kernel2dsmooth( x, kernel.type="unsharp", alpha=0)
kernel2dsmooth( x, kernel.type="wave", phi=45, nx=10, ny=12)

## Not run:
## the lennon image is in package 'fields'.
data(lennon)
kmat <- kernel2dmeitsjer( "average", nx=7, ny=5)
lennon.smAvg <- kernel2dsmooth( lennon, K=kmat)
## Can also just make a call to kernel2dsmooth, which
## will call this function.
lennon.smBox <- kernel2dsmooth( lennon, kernel.type="boxcar", n=7)
lennon.smDsk <- kernel2dsmooth( lennon, kernel.type="disk", r=5)
par( mfrow=c(2,2), mar=rep(0.1,4))
image.plot( lennon, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smAvg, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smBox, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smDsk, col=tim.colors(256), axes=FALSE)

lennon.smEpa <- kernel2dsmooth( lennon, kernel.type="epanechnikov", nx=10, ny=10, sigma=20)
lennon.smGau <- kernel2dsmooth( lennon, kernel.type="gauss", nx=10, ny=10, sigma=20)
lennon.smMvr <- kernel2dsmooth( lennon, kernel.type="minvar", nx=10, ny=10, sigma=20)
par( mfrow=c(2,2), mar=rep(0.1,4))
image.plot( lennon, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smEpa, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smGau, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smMvr, col=tim.colors(256), axes=FALSE)

lennon.smLa0 <- kernel2dsmooth( lennon, kernel.type="laplacian", alpha=0)
lennon.smLap <- kernel2dsmooth( lennon, kernel.type="laplacian", alpha=0.999)
lennon.smLoG <- kernel2dsmooth( lennon, kernel.type="LoG", nx=10, ny=10, sigma=20)
par( mfrow=c(2,2), mar=rep(0.1,4))
image.plot( lennon, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smLa0, col=tim.colors(256), axes=FALSE)
```

```
image.plot( lennon.smLap, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smLoG, col=tim.colors(256), axes=FALSE)

lennon.smPrH <- kernel2dsmooth( lennon, kernel.type="prewitt")
lennon.smPrV <- kernel2dsmooth( lennon, kernel.type="prewitt", transpose=TRUE)
lennon.smSoH <- kernel2dsmooth( lennon, kernel.type="sobel")
lennon.smSoV <- kernel2dsmooth( lennon, kernel.type="sobel", transpose=TRUE)
par( mfrow=c(2,2), mar=rep(0.1,4))
image.plot( lennon.smPrH, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smPrV, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smSoH, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smSoV, col=tim.colors(256), axes=FALSE)

lennon.smUsh <- kernel2dsmooth( lennon, kernel.type="unsharp", alpha=0.999)
par( mfrow=c(2,1), mar=rep(0.1,4))
image.plot( lennon, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smUsh, col=tim.colors(256), axes=FALSE)

lennon.smRad1 <- kernel2dsmooth( lennon, kernel.type="radial", a=2, m=2, d=1, nx=10, ny=10)
lennon.smRad2 <- kernel2dsmooth( lennon, kernel.type="radial", a=2, m=2, d=2, nx=10, ny=10)
par( mfrow=c(2,1), mar=rep(0.1,4))
image.plot( lennon.smRad1, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smRad2, col=tim.colors(256), axes=FALSE)

lennon.smRQd <- kernel2dsmooth( lennon, kernel.type="ratquad", a=0.5, nx=10, ny=10)
lennon.smExp <- kernel2dsmooth( lennon, kernel.type="exponential", a=0.5, sigma=20, nx=10, ny=10)
lennon.smMQd <- kernel2dsmooth( lennon, kernel.type="multiquad", a=0.5, nx=10, ny=10)
par( mfrow=c(2,2), mar=rep(0.1,4))
image.plot( lennon.smGau, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smRQd, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smExp, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smMQd, col=tim.colors(256), axes=FALSE)

lennon.smIMQ <- kernel2dsmooth( lennon, kernel.type="multiquad", a=0.5, nx=10, ny=10, inverse=TRUE)
par( mfrow=c(2,1), mar=rep(0.1,4))
image.plot( lennon.smMQd, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smIMQ, col=tim.colors(256), axes=FALSE)

lennon.smWav <- kernel2dsmooth( lennon, kernel.type="wave", phi=45, nx=10, ny=10)
par( mfrow=c(1,1), mar=rep(0.1,4))
image.plot( lennon.smWav, col=tim.colors(256), axes=FALSE)

lennon.smPow <- kernel2dsmooth( lennon, kernel.type="power", p=0.5, nx=10, ny=10)
lennon.smLpw <- kernel2dsmooth( lennon, kernel.type="power", p=0.5, nx=10, ny=10, do.log=TRUE)
par( mfrow=c(2,1), mar=rep(0.1,4))
image.plot( lennon.smPow, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smLpw, col=tim.colors(256), axes=FALSE)

lennon.smCau <- kernel2dsmooth( lennon, kernel.type="cauchy", sigma=20, nx=10, ny=10)
lennon.smStd <- kernel2dsmooth( lennon, kernel.type="student", p=1.5, nx=10, ny=10)
par( mfrow=c(2,1), mar=rep(0.1,4))
image.plot( lennon.smCau, col=tim.colors(256), axes=FALSE)
image.plot( lennon.smStd, col=tim.colors(256), axes=FALSE)
```

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

---

`kernel2dsmooth` *Convolution Smooth on a 2-d Field.*

---

## Description

Perform a onvolution smooth on a 2-d field. Default is to take an average over all neighbors within (n-1)/2 grid points from each grid point. Uses FFT with the convolution theorem for computational efficiency.

## Usage

```
kernel2dsmooth(x, kernel.type=NULL,
    K = NULL, W = NULL, X = NULL, xdim = NULL, Nxy = NULL,
    setup = FALSE, verbose = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | matrix to be smoothed. |
| `kernel.type` | (optional) character naming the kernel type accepted by `kernel2dmeitsjer`. One and only one of this argument, K or W must be supplied. |
| `K` | (optional) matrix defining a kernel to be applied. this function will expand, and flip the kernel about its center, so ideally it will have odd dimensions. |
| `W` | (optional) possibly complex matrix of scaled Fourier transformed kernel weights. If NULL, these will be computed, but if passed, it will save one FFT in computation time. This should not be given if setup is TRUE, or you will not get what you expect. The dimensions are determined by this function, so it is probably best to supply this matrix only from a previous call with setup=TRUE. The dimensions are chosen to optimize the FFT calculations (see Details section). |
| `X` | (optional) matrix giving the Fourier transformed `x`. Can be used to save an FFT in computation, if this has already been calculated. |
| `xdim` | (optional) numeric vector of length 2 giving the dimensions of `x`. Not really necessary, but as it will have already been calculated, seems silly to have to keep re-calculating it. If NULL, it will be calculated here. |
| `Nxy` | (optional) total number of grid points of x. Similar to xdim argument, not really necessary, and will be calculated if not passed. |
| `setup` | logical, should just the Fourier transformed kernel weights, W, be returned for subsequent calls to this function? |
| `verbose` | logical, should progress information be printed to the screen? |
| `...` | optional arguments to `kernel2dmeitsjer` as required by the specific kernel given to kernel.type |

## Details

This 2-d spatial kernel smoother applies a kernel smoother to a spatial field (see Hastie and Tibshirani, 1990 sec. 2.6; Ritter and Wilson, 2001, chapter 8; Barrett and Myers, 2004 for details about this type of convolution smoothing). Specifically, if `X` is a matrix of grid points, then the returned field, denoted by Ebert (2008) as <X>s, is a smoothed field such that the value at each grid point <X>s[i,j] is given by: <X>s[i,j] = sum_k sum_l X[i + k - 1, j + l - 1]*K[i, j], where k,l = 1, ..., n, and K[i, j] is the kernel matrix. In order to be fast, loops are avoided. Instead, the convolution theorem is applied with a Fast Fourier Transform (FFT). If the weights 'W' are supplied, then you will save one FFT in computation time.

The convolution theorem says that the Fourier transform of a convolution between two functions f and g is equal to the product of the Fourier transformed functions. That is, if F denotes the Fourier transform, and * the convolution operator, F( f*g ) = k F(f)F(g), where 'k' is a scaling factor. The neighborhood smooth is given by a convolution between the field and a boxcar kernel (i.e., a square around a point with constant value 1/n^2). Because of the FFT, this enables a fast way to compute this convolution.

In order to zero-pad the field, and perform a cyclic convolution, it is necessary to expand the field, 'x', and re-arrange the kernel (or else it will not be centered on the points). If zero padding is not desired, then a field that has been extrapolated to an appropriate size in another way should be used, and a subset going back to the original size could be used. Alternatively, a subset of an appropriate size could be taken from the resulting smoothed field in order to avoid edge effects. The latter is probably a wise move. The image is expanded to the nearest power of two above the dimension of N=x + dimension of K - 1 in each direction, if N <= 1024, and N is rounded up to the nearest multiple of 512 otherwise. This is to ensure that the FFT is fast.

In order to get the neighborhood type of smoothing of Roberts and Lean (2008) and Ebert (2008), use the boxcar kernel with the argument n giving the neighborhood length. The resulting kernel is n by n with elements 1/n^2 at each point. The result is that each grid point of the returned field is an average of the n^2 nearest neighbors. Alternatively, one might prefer to use a disk kernel, which takes the radius, r, as an argument. This gives a similar type of kernel, but ensures an average over a uniform distance from the center point. The disk kernel is also that which is used in the smoothing step of Davis et al (2006a,2006b). See the help file for `kernel2dmeitsjer` for other smoothing options.

## Value

If setup is FALSE, then a k by m matrix giving the neighborhood smoothed field is returned. Otherwise, a 2k by 2m possibly complex matrix giving the Fourier transformed kernel weights are returned, which can be used to save an FFT in computation time for subsequent calls to this function by supplying the W argument with this result.

## Note

If n is 1, then the field is returned without applying any smoothing.

## Author(s)

Eric Gilleland

## References

Barrett, H. H. and Myers, K. J. (2004) *Foundations of Image Science*. Wiley Series in Pure and Applied Optics, Editor: B. E. A. Saleh, Hoboken, New Jersey, 1540pp.

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.

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. Available at http://www.ecmwf.int/newsevents/meetings/workshops/2007/jwgv/METspecialissueemail.pdf

Hastie, T. J. and Tibshirani, R. J. (1990) *Generalized Additive Models*. Chapman \& Hall/CRC Monographs on Statistics and Applied Probability 43, 335pp.

Ritter, G. X. and Wilson, J. N. (2001) *Handbook of Computer Vision Algorithms in Image Algebra*. 2nd Edition, CRC Press, Boca Raton, Florida, U.S.A., 417pp.

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

fft, Fourier2d, kernel2dmeitsjer

## Examples

```
look <- matrix( 0, 10, 12)
look[4,7] <- 1
kernel2dsmooth( look, kernel.type="boxcar", n=3)
# The above returns the shape of the kernel applied, which
# is a square of length 3 centered on the grid point in look
# that has a value of 1.

# What happens if the 1 is on the edge?  the effect is zero padding:
look <- look*0
look[1,1] <- 1
kernel2dsmooth( look, kernel.type="boxcar", n=3)

# Suppose we want to do the above for several, say l, neighborhood lengths.
# We can save an FFT for l-1 of the convolutions.
look <- look*0
look[4,7] <- 1
lookFFT <- Fourier2d( look, kdim=c(3,3))
dim( lookFFT) # Note the dimension is twice that of look.
kernel2dsmooth( look, kernel.type="boxcar", n=3, X=lookFFT)

# Now, suppose we want to apply the same kernel smooth to different fields.
# We can save an FFT for each subsequent calculation as follows.
wg <- kernel2dsmooth( look, kernel.type="boxcar", n=3, setup=TRUE)
dim( wg) # Note the dimension is twice that of look.
kernel2dsmooth( look, kernel.type="boxcar", n=3, W=wg)
```

```
look <- look*0
look[8,5] <- 1
kernel2dsmooth( look, kernel.type="boxcar", n=3, W=wg)
look[5, 10] <- 1
kernel2dsmooth( look, kernel.type="boxcar", n=3, W=wg)
```

# Index