Package ‘raster’

October 16, 2018

Type Package
Title Geographic Data Analysis and Modeling
Version 2.7-15
Date 2018-10-12
Depends methods, sp (>= 1.2-0), R (>= 3.0.0)
Suggests rgdal (>= 0.9-1), rgeos (>= 0.3-8), ncdf4, igraph, tcltk, parallel, rasterVis, MASS, sf
LinkingTo Rcpp
Imports Rcpp
SystemRequirements C++11
Description Reading, writing, manipulating, analyzing and modeling of gridded spatial data. The package implements basic and high-level functions. Processing of very large files is supported.
License GPL (>= 3)
URL http://www.rspatial.org/
ByteCompile TRUE
NeedsCompilation yes
Author Robert J. Hijmans [cre, aut],
        Jacob van Etten [ctb],
        Joe Cheng [ctb],
        Matteo Mattiuzzi [ctb],
        Michael Sumner [ctb],
        Jonathan A. Greenberg [ctb],
        Oscar Perpinan Lamigueiro [ctb],
        Andrew Bevan [ctb],
        Roger Bivand [ctb],
        Mort Canty [ctb],
        David Forrest [ctb],
        Aniruddha Ghosh [ctb],
        Duncan Golicher [ctb],
        Josh Gray [ctb],
R topics documented:

- raster-package .......................................................... 6
- addLayer ................................................................. 14
- adjacent ................................................................. 15
- aggregate ................................................................. 16
- alignExtent .............................................................. 19
- animate ................................................................. 20
- approxNA ................................................................. 21
- area ................................................................. 22
- Arith-methods ........................................................ 23
- as.data.frame .......................................................... 24
- as.list ................................................................. 26
- as.logical ............................................................ 27
- as.matrix ............................................................. 27
- as.raster .............................................................. 29
- atan2 ................................................................. 29
- autocorrelation ..................................................... 30
- bands ................................................................. 30
- barplot ............................................................... 31
- bind ................................................................. 32
- blockSize ........................................................... 33
- boundaries .......................................................... 34
- boxplot ............................................................... 35
- brick ................................................................. 36
- buffer ................................................................. 37
- calc ................................................................. 39
- cellFrom ............................................................. 40
- cellsFromExtent ................................................ 42
- cellStats ............................................................ 43
- clamp ................................................................. 45
- clearValues ........................................................ 46
- click ................................................................. 47
R topics documented:
clump .......................................................... 50
cluster ......................................................... 51
colorable ....................................................... 53
Compare-methods ........................................... 54
compareCRS .................................................. 55
compareRaster ............................................... 56
contour ......................................................... 57
corLocal ........................................................ 58
cover ........................................................... 59
crop ............................................................ 60
crosstab ....................................................... 62
cut ............................................................... 63
cv ............................................................... 64
datasource ...................................................... 64
dataType ......................................................... 65
density .......................................................... 67
dim .............................................................. 67
direction ......................................................... 68
disaggregate .................................................. 69
distance ........................................................ 70
distanceFromPoints ........................................ 71
draw ............................................................. 73
drawExtent ..................................................... 73
erase ............................................................ 74
extend .......................................................... 75
extension ......................................................... 76
extent .......................................................... 77
Extent math .................................................... 78
Extent-class .................................................... 79
extract ........................................................ 80
Extract by index .............................................. 83
Extreme coordinates ........................................ 85
extremeValues ............................................... 86
factors .......................................................... 87
filename ......................................................... 89
filledContour .................................................. 89
flip .............................................................. 90
flowPath ......................................................... 91
focal ............................................................ 92
focalWeight .................................................... 94
freq ............................................................. 95
Gain and offset ............................................... 96
geom ........................................................... 97
getData ........................................................ 98
getValues ....................................................... 99
getValuesBlock ............................................... 100
getValuesFocal ............................................... 101
gridDistance .................................................. 102
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hdr</td>
<td>104</td>
</tr>
<tr>
<td>head</td>
<td>105</td>
</tr>
<tr>
<td>hillShade</td>
<td>105</td>
</tr>
<tr>
<td>hist</td>
<td>107</td>
</tr>
<tr>
<td>image</td>
<td>108</td>
</tr>
<tr>
<td>infile</td>
<td>109</td>
</tr>
<tr>
<td>initialize</td>
<td>109</td>
</tr>
<tr>
<td>interpolate</td>
<td>110</td>
</tr>
<tr>
<td>intersect</td>
<td>113</td>
</tr>
<tr>
<td>isLonLat</td>
<td>115</td>
</tr>
<tr>
<td>KML</td>
<td>116</td>
</tr>
<tr>
<td>layerize</td>
<td>117</td>
</tr>
<tr>
<td>layerStats</td>
<td>118</td>
</tr>
<tr>
<td>localFun</td>
<td>119</td>
</tr>
<tr>
<td>Logic-methods</td>
<td>121</td>
</tr>
<tr>
<td>mask</td>
<td>122</td>
</tr>
<tr>
<td>match</td>
<td>123</td>
</tr>
<tr>
<td>Math-methods</td>
<td>124</td>
</tr>
<tr>
<td>merge</td>
<td>125</td>
</tr>
<tr>
<td>metadata</td>
<td>126</td>
</tr>
<tr>
<td>modal</td>
<td>127</td>
</tr>
<tr>
<td>mosaic</td>
<td>128</td>
</tr>
<tr>
<td>movingFun</td>
<td>129</td>
</tr>
<tr>
<td>names</td>
<td>130</td>
</tr>
<tr>
<td>NAvalue</td>
<td>131</td>
</tr>
<tr>
<td>ncell</td>
<td>132</td>
</tr>
<tr>
<td>nlayers</td>
<td>133</td>
</tr>
<tr>
<td>Options</td>
<td>134</td>
</tr>
<tr>
<td>origin</td>
<td>136</td>
</tr>
<tr>
<td>overlay</td>
<td>136</td>
</tr>
<tr>
<td>pairs</td>
<td>139</td>
</tr>
<tr>
<td>persp</td>
<td>140</td>
</tr>
<tr>
<td>plot</td>
<td>141</td>
</tr>
<tr>
<td>plotRGB</td>
<td>144</td>
</tr>
<tr>
<td>pointDistance</td>
<td>145</td>
</tr>
<tr>
<td>predict</td>
<td>147</td>
</tr>
<tr>
<td>Programming</td>
<td>150</td>
</tr>
<tr>
<td>projection</td>
<td>152</td>
</tr>
<tr>
<td>projectRaster</td>
<td>153</td>
</tr>
<tr>
<td>properties</td>
<td>155</td>
</tr>
<tr>
<td>quantile</td>
<td>156</td>
</tr>
<tr>
<td>raster</td>
<td>157</td>
</tr>
<tr>
<td>Raster-class</td>
<td>160</td>
</tr>
<tr>
<td>rasterFromCells</td>
<td>162</td>
</tr>
<tr>
<td>rasterFromXYZ</td>
<td>163</td>
</tr>
<tr>
<td>rasterize</td>
<td>164</td>
</tr>
<tr>
<td>rasterTmpFile</td>
<td>168</td>
</tr>
<tr>
<td>rasterToContour</td>
<td>169</td>
</tr>
</tbody>
</table>
R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rasterToPoints</td>
<td>170</td>
</tr>
<tr>
<td>rasterToPolygons</td>
<td>171</td>
</tr>
<tr>
<td>Rcpp-class</td>
<td>172</td>
</tr>
<tr>
<td>readAll</td>
<td>172</td>
</tr>
<tr>
<td>reclassify</td>
<td>173</td>
</tr>
<tr>
<td>rectify</td>
<td>174</td>
</tr>
<tr>
<td>replacement</td>
<td>175</td>
</tr>
<tr>
<td>resample</td>
<td>175</td>
</tr>
<tr>
<td>resolution</td>
<td>176</td>
</tr>
<tr>
<td>RGB</td>
<td>177</td>
</tr>
<tr>
<td>rotate</td>
<td>178</td>
</tr>
<tr>
<td>rotated</td>
<td>179</td>
</tr>
<tr>
<td>round</td>
<td>180</td>
</tr>
<tr>
<td>rowFromCell</td>
<td>181</td>
</tr>
<tr>
<td>rowSums</td>
<td>182</td>
</tr>
<tr>
<td>SampleInt</td>
<td>183</td>
</tr>
<tr>
<td>sampleRandom</td>
<td>183</td>
</tr>
<tr>
<td>sampleRegular</td>
<td>184</td>
</tr>
<tr>
<td>sampleStratified</td>
<td>185</td>
</tr>
<tr>
<td>scale</td>
<td>186</td>
</tr>
<tr>
<td>scalebar</td>
<td>187</td>
</tr>
<tr>
<td>select</td>
<td>188</td>
</tr>
<tr>
<td>setExtent</td>
<td>190</td>
</tr>
<tr>
<td>setMinMax</td>
<td>191</td>
</tr>
<tr>
<td>setValues</td>
<td>191</td>
</tr>
<tr>
<td>shapefile</td>
<td>193</td>
</tr>
<tr>
<td>shift</td>
<td>194</td>
</tr>
<tr>
<td>Slope and aspect</td>
<td>195</td>
</tr>
<tr>
<td>sp</td>
<td>195</td>
</tr>
<tr>
<td>spplot</td>
<td>196</td>
</tr>
<tr>
<td>stack</td>
<td>197</td>
</tr>
<tr>
<td>stackApply</td>
<td>199</td>
</tr>
<tr>
<td>stackSave</td>
<td>200</td>
</tr>
<tr>
<td>stackSelect</td>
<td>201</td>
</tr>
<tr>
<td>stretch</td>
<td>202</td>
</tr>
<tr>
<td>subset</td>
<td>203</td>
</tr>
<tr>
<td>substitute</td>
<td>204</td>
</tr>
<tr>
<td>Summary</td>
<td>205</td>
</tr>
<tr>
<td>Summary-methods</td>
<td>206</td>
</tr>
<tr>
<td>symdif</td>
<td>207</td>
</tr>
<tr>
<td>terrain</td>
<td>208</td>
</tr>
<tr>
<td>text</td>
<td>210</td>
</tr>
<tr>
<td>transpose</td>
<td>211</td>
</tr>
<tr>
<td>trim</td>
<td>212</td>
</tr>
<tr>
<td>union</td>
<td>213</td>
</tr>
<tr>
<td>unique</td>
<td>214</td>
</tr>
<tr>
<td>unstack</td>
<td>215</td>
</tr>
<tr>
<td>update</td>
<td>216</td>
</tr>
</tbody>
</table>
raster-package

Overview of the functions in the raster package

Description

The raster package provides classes and functions to manipulate geographic (spatial) data in 'raster' format. Raster data divides space into cells (rectangles; pixels) of equal size (in units of the coordinate reference system). Such continuous spatial data are also referred to as 'grid' data, and be contrasted with discrete (object based) spatial data (points, lines, polygons).

The package should be particularly useful when using very large datasets that can not be loaded into the computer’s memory. Functions will work correctly, because they they process large files in chunks, i.e., they read, compute, and write blocks of data, without loading all values into memory at once.

Below is a list of some of the most important functions grouped by theme. See the vignette for more information and some examples (you can open it by running this command: vignette('Raster'))

Details

The package implements classes for Raster data (see Raster-class) and supports

- Creation of Raster* objects from scratch or from file
- Handling extremely large raster files
- Raster algebra and overlay functions
- Distance, neighborhood (focal) and patch functions
- Polygon, line and point to raster conversion
- Model predictions
- Summarizing raster values
- Easy access to raster cell-values
- Plotting (making maps)
• Manipulation of raster extent, resolution and origin
• Computation of row, col and cell numbers to coordinates and vice versa
• Reading and writing various raster file types

I. Creating Raster* objects

RasterLayer, RasterStack, and RasterBrick objects are, as a group, referred to as Raster* objects. Raster* objects can be created, from scratch, files, or from objects of other classes, with the following functions:

- **raster** To create a RasterLayer
- **stack** To create a RasterStack (multiple layers)
- **brick** To create a RasterBrick (multiple layers)
- **subset** Select layers of a RasterStack/Brick
- **addLayer** Add a layer to a Raster* object
- **dropLayer** Remove a layer from a RasterStack or RasterBrick
- **unstack** Create a list of RasterLayer objects from a RasterStack

II. Changing the spatial extent and/or resolution of Raster* objects

- **merge** Combine Raster* objects with different extents (but same origin and resolution)
- **mosaic** Combine RasterLayers with different extents and a function for overlap areas
- **crop** Select a geographic subset of a Raster* object
- **extend** Enlarge a Raster* object
- **trim** Trim a Raster* object by removing exterior rows and/or columns that only have NAs
- **aggregate** Combine cells of a Raster* object to create larger cells
- **disaggregate** Subdivide cells
- **resample** Warp values to a Raster* object with a different origin or resolution
- **projectRaster** Project values to a raster with a different coordinate reference system
- **shift** Move the location of Raster
- **flip** Flip values horizontally or vertically
- **rotate** Rotate values around the date-line (for lon/lat data)
- **t** Transpose a Raster* object

III. Raster algebra

- **Arith-methods** Arith functions (+, -, *, ^, %%, %/%, /)
- **Math-methods** Math functions like abs, sqrt, trunc, log, log10, exp, sin, round
- **Logic-methods** Logic functions (!, &, |)
Summary-methods

Summary functions (mean, max, min, range, prod, sum, any, all)

Compare-methods

Compare functions (==, !-, >, <, <=, >=)

IV. Cell based computation

calc  Computations on a single Raster* object
overlay Computations on multiple RasterLayer objects
cover First layer covers second layer except where the first layer is NA
mask Use values from first Raster except where cells of the mask Raster are NA
cut Reclassify values using ranges
subs Reclassify values using an 'is-becomes' matrix
reclassify Reclassify using a 'from-to-becomes' matrix
init Initialize cells with new values
stackApply Computations on groups of layers in Raster* object
stackSelect Select cell values from different layers using an index RasterLayer

V. Spatial contextual computation

distance Shortest distance to a cell that is not NA
gridDistance Distance when traversing grid cells that are not NA
distanceFromPoints Shortest distance to any point in a set of points
direction Direction (azimuth) to or from cells that are not NA
focal Focal (neighborhood; moving window) functions
localFun Local association (using neighborhoods) functions
boundaries Detection of boundaries (edges)
clump Find clumps (patches)
adjacent Identify cells that are adjacent to a set of cells on a raster
area Compute area of cells (for longitude/latitude data)
terrain Compute slope, aspect and other characteristics from elevation data
Moran Compute global or local Moran or Geary indices of spatial autocorrelation

VI. Model predictions

predict Predict a non-spatial model to a RasterLayer
interpolate  Predict a spatial model to a RasterLayer

VII. Data type conversion

You can coerce Raster* objects to Spatial* objects using as, as in `as(SpatialGridDataFrame)`

- `raster`: RasterLayer from SpatialGrid*, image, or matrix objects
- `rasterize`: Rasterizing points, lines or polygons
- `rasterToPoints`: Create points from a RasterLayer
- `rasterToPolygons`: Create polygons from a RasterLayer
- `rasterToContour`: Contour lines from a RasterLayer
- `rasterFromXYZ`: RasterLayer from regularly spaced points
- `rasterFromCells`: RasterLayer from a Raster object and cell numbers

VIII. Summarizing

- `cellStats`: Summarize a Raster cell values with a function
- `summary`: Summary of the values of a Raster object (quartiles and mean)
- `freq`: Frequency table of Raster cell values
- `crosstab`: Cross-tabulate two Raster objects
- `unique`: Get the unique values in a Raster object
- `zonal`: Summarize a Raster object by zones in a RasterLayer

IX. Accessing values of Raster* object cells

Apart from the function listed below, you can also use indexing with [ ] for cell numbers, and [ ] for row / column number combinations

- `getValues`: Get all cell values (fails with very large rasters), or a row of values (safer)
- `getValuesBlock`: Get values for a block (a rectangular area)
- `getValuesFocal`: Get focal values for one or more rows
- `as.matrix`: Get cell values as a matrix
- `as.array`: Get cell values as an array
- `extract`: Extract cell values from a Raster* object (e.g., by cell, coordinates, polygon)
- `sampleRandom`: Random sample
- `sampleRegular`: Regular sample
- `minValue`: Get the minimum value of the cells of a Raster* object (not always known)
maxValue
setMinMax

Get the maximum value of the cells of a Raster* object (not always known)
Compute the minimum and maximum value of a Raster* object if these are not known

X. Plotting

See the rasterVis package for additional plotting methods for Raster* objects using methods from 'lattice' and other packages.

Maps

plot
plotRGB
spplot
image
persp
contour
filledContour
text

Plot a Raster* object. The main method to create a map
Combine three layers (red, green, blue channels) into a single 'real color' image
Plot a Raster* with the spplot function (sp package)
Plot a Raster* with the image function
Perspective plot of a RasterLayer
Contour plot of a RasterLayer
Filled contour plot of a RasterLayer
Plot the values of a RasterLayer on top of a map

Interacting with a map

zoom
click
select
drawPoly
drawLine
drawExtent

Zoom in to a part of a map
Query values of Raster* or Spatial* objects by clicking on a map
Select a geometric subset of a Raster* or Spatial* object
Create a SpatialPolygons object by drawing it
Create a SpatialLines object by drawing it
Create an Extent object by drawing it

Other plots

plot
hist
barplot
density
pairs
boxplot

x-y scatter plot of the values of two RasterLayer objects
Histogram of Raster* object values
barplot of a RasterLayer
Density plot of Raster* object values
Pairs plot for layers in a RasterStack or RasterBrick
Box plot of the values of one or multiple layers

XI. Getting and setting Raster* dimensions

Basic parameters of existing Raster* objects can be obtained, and in most cases changed. If there are values associated with a RasterLayer object (either in memory or via a link to a file) these are lost when you change the number of columns or rows or the resolution. This is not the case when the extent is changed (as the number of columns and rows will not be affected). Similarly, with projection you can set the projection, but this does not transform the data (see projectRaster for that).

ncol
nrow

The number of columns
The number of rows
XII. Computing row, column, cell numbers and coordinates

Cell numbers start at 1 in the upper-left corner. They increase within rows, from left to right, and then row by row from top to bottom. Likewise, row numbers start at 1 at the top of the raster, and column numbers start at 1 at the left side of the raster.

xFromCol      x-coordinates from column numbers
yFromRow      y-coordinates from row numbers
xFromCell     x-coordinates from row numbers
yFromCell     y-coordinates from cell numbers
xyFromCell    x and y coordinates from cell numbers
colFromX      Column numbers from x-coordinates (or longitude)
rowFromY      Row numbers from y-coordinates (or latitude)
rowColFromCell Row and column numbers from cell numbers
cellFromXY    Cell numbers from x and y coordinates
cellFromRowCol Cell numbers from row and column numbers
cellsFromExtent Cell numbers from extent object
coordinates   x and y coordinates for all cells
validCell     Is this a valid cell number?
validCol      Is this a valid column number?
validRow      Is this a valid row number?

XIII. Writing files

Basic
setValues  Put new values in a Raster* object
writeRaster  Write all values of Raster* object to disk
KML  Save raster as KML file

Advanced
blockSize  Get suggested block size for reading and writing
writeStart  Open a file for writing
writeValues  Write some values
writeStop  Close the file after writing
update  Change the values of an existing file

XIV. Manipulation of SpatialPolygons* and other vector type Spatial* objects

Some of these functions are in the sp package. The name in bold is the equivalent command in ArcGIS. These functions build on the geometry ("spatial features") manipulation functions in package rgeos. These functions are extended here by also providing automated attribute data handling.

bind  append combine Spatial* objects of the same (vector) type
erase or "-"  erase parts of a SpatialPolygons* object
intersect or "&"  intersect SpatialPolygons* objects
union or "+"  union SpatialPolygons* objects
cover  update and identity a SpatialPolygons object with another one
symdif  symmetrical difference of two SpatialPolygons* objects
aggregate  dissolve smaller polygons into larger ones
disaggregate  explode: turn polygon parts into separate polygons (in the sp package)
crop  clip a Spatial* object using a rectangle (Extent object)
select  select - interactively select spatial features
click  identify attributes by clicking on a map
merge
over
extract
as.data.frame

XV. Extent objects

extent  Create an extent object
intersect  Intersect two extent objects
union  Combine two extent objects
round  round/floor/ceiling of the coordinates of an Extent object
alignExtent  Align an extent with a Raster* object
drawExtent  Create an Extent object by drawing it on top of a map (see plot)
XVI. Miscellaneous

- `rasterOptions` : Show, set, save or get session options
- `getData` : Download and geographic data
- `pointDistance` : Distance between points
- `readIniFile` : Read a (windows) ‘ini’ file
- `hdr` : Write header file for a number of raster formats
- `trim` : Remove leading and trailing blanks from a character string
- `extension` : Get or set the extension of a filename
- `cv` : Coefficient of variation
- `modal` : Modal value
- `sampleInt` : Random sample of (possibly very large) range of integer values
- `showTmpFiles` : Show temporary files
- `removeTmpFiles` : Remove temporary files

XVII. For programmers

- `canProcessInMemory` : Test whether a file can be created in memory
- `pbCreate` : Initialize a progress bar
- `pbStep` : Take a progress bar step
- `pbClose` : Close a progress bar
- `readStart` : Open file connections for efficient multi-chunck reading
- `readStop` : Close file connections
- `rasterTmpFile` : Get a name for a temporary file
- `inMemory` : Are the cell values in memory?
- `fromDisk` : Are the cell values read from a file?

Acknowledgements

Extensive contributions were made by Jacob van Etten, Jonathan Greenberg, Matteo Mattiuzzi, and Michael Sumner. Significant help was also provided by Phil Heilman, Agustin Lobo, Oscar Perpinan Lamigueiro, Stefan Schlaffer, Jon Olav Skoien, Steven Mosher, and Kevin Ummel. Contributions were also made by Jochen Albrecht, Neil Best, Andrew Bevan, Roger Bivand, Isabelle Boulangeat, Lyndon Estes, Josh Gray, Tim Haering, Herry Herry, Paul Hiemstra, Ned Hornig, Mayeul Kauffmann, Bart Kranstauber, Rainer Krug, Alice Laborte, John Lewis, Lennon Li, Justin
addLayer

McGrath, Babak Naimi, Carsten Neumann, Joshua Perlman, Richard Plant, Edzer Pebesma, Etienne Racine, David Ramsey, Shaun Walbridge, Julian Zeidler and many others.

Author(s)

Except where indicated otherwise, the functions in this package were written by Robert J. Hijmans

addLayer Add or drop a layer

Description

Add a layer to a Raster* object or drop a layer from a RasterStack or RasterBrick. The object returned is always a RasterStack (unless nothing to add or drop was provided, in which case the original object is returned).

Usage

addLayer(x, ...) dropLayer(x, i, ...)

Arguments

x Raster* object
i integer. Indices of the layers to be dropped
... Additional arguments. The layers to add for addLayer. None for dropLayer

Value

RasterStack

See Also

subset

Examples

file <- system.file("external/test.grd", package="raster")
s <- stack(file, file, file)
r <- raster(file)
s <- addLayer(s, r/2, r*2)
s
s <- dropLayer(s, c(3, 5))
nlayers(s)
adjacent

Description

Identify cells that are adjacent to a set of cells on a raster.

Usage

adjacent(x, cells, directions=4, pairs=TRUE, target=NULL, sorted=FALSE, include=FALSE, id=FALSE)

Arguments

x
Raster* object

cells
vector of cell numbers for which adjacent cells should be found. Cell numbers start with 1 in the upper-left corner and increase from left to right and from top to bottom

directions
the number of directions in which cells should be connected: 4 (rook’s case), 8 (queen’s case), 16 (knight and one-cell queen moves), or ‘bishop’ to connect cells with one-cell diagonal moves. Or a neighborhood matrix (see Details)
pairs
logical. If TRUE, a matrix of pairs of adjacent cells is returned. If FALSE, a vector of cells adjacent to cells is returned
target
optional vector of target cell numbers that should be considered. All other adjacent cells are ignored
sorted
logical. Should the results be sorted?
include
logical. Should the focal cells be included in the result?
id
logical. Should the id of the cells be included in the result? (numbered from 1 to length(cells)

Details

A neighborhood matrix identifies the cells around each cell that are considered adjacent. The matrix should have one, and only one, cell with value 0 (the focal cell); at least one cell with value 1 (the adjacent cell(s)); All other cells are not considered adjacent and ignored.

Value

matrix or vector with adjacent cells.

Author(s)

Robert J. Hijmans and Jacob van Etten
Examples

r <- raster(nrows=10, ncols=10)
adjacent(r, cells=c(1, 55), directions=8, pairs=TRUE)

a <- adjacent(r, cell = c(155,90), directions=4, sorted=TRUE)
a

r[c(1,55,90)] <- 1
r[a] <- 2
plot(r)

# same result as above
rook <- matrix(c(NA, 1, NA,
1, 0, 1,
NA, 1, NA), ncol=3, byrow=TRUE)
adjacent(r, cells = c(1,55,90), directions=rook, sorted=TRUE)

# Count the number of times that a cell with a certain value
# occurs next to a cell with a certain value
set.seed(0)
r <- raster(ncol=10, nrow=10)
r[] <- round(runif(ncell(r)) * 5)
a <- adjacent(r, 1:ncell(r), 4, pairs=TRUE)
tb <- table(r[a[,1]], r[a[,2]])
tb

# make a matrix out of the 'table' object
tb <- unclass(tb)
plot(raster(tb, xmn=-0.5, xmx=5.5, ymn=-0.5, ymx=5.5))

---

aggregate

Aggregate raster cells or SpatialPolygons/Lines

Description

Raster* objects:

Aggregate a Raster* object to create a new RasterLayer or RasterBrick with a lower resolution (larger cells). Aggregation groups rectangular areas to create larger cells. The value for the resulting cells is computed with a user-specified function.

SpatialPolygon*:

Aggregate a SpatialPolygon* object, optionally by combining polygons that have the same attributes for one or more variables. If the polygons touch or overlap, internal boundaries are optionally "dissolved".
Usage

```r
## S4 method for signature 'Raster'
aggregate(x, fact=2, fun=mean, expand=TRUE, na.rm=TRUE, filename='', ...)

## S4 method for signature 'SpatialPolygons'
aggregate(x, by, sums, dissolve=TRUE, vars=NULL, ...)
```

Arguments

- `x`: Raster* object or SpatialPolygons* object
- `fact`: integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor) or three integers (when also aggregating over layers). See Details
- `fun`: function used to aggregate values
- `expand`: logical. If TRUE the output Raster* object will be larger then the input Raster* object if a division of the number of columns or rows with `fact` is not an integer
- `na.rm`: logical. If TRUE, NA cells are removed from calculations
- `filename`: character. Output filename (optional)
- `...`: if `x` is a Raster* object, additional arguments as for `writeRaster`
- `by`: character or integer. The variables (column names or numbers) that should be used to aggregate (dissolve) the SpatialPolygons by only maintaining unique combinations of these variables. The default setting is to use no variables and aggregate all polygons. You can also supply a vector with a length of length(x)
- `sums`: list with function(s) and variable(s) to summarize. This should be a list of lists in which each element of the main lists has two items. The first item is function (e.g. mean), the second element is a vector of column names (or indices) that need to summarize with that function. Be careful with character and factor variables (you can use, e.g. 'first' function(x)x[1] or 'last' function(x)x[length(x)] or modal for these variables
- `vars`: deprecated. Same as by
- `dissolve`: logical. If TRUE borders between touching or overlapping polygons are removed (requires package rgeos)

Details

Aggregation of a `x` will result in a Raster* object with fewer cells. The number of cells is the number of cells of `x` divided by `fact*fact` (when `fact` is a single number) or `prod(fact)` (when `fact` consists of 2 or 3 numbers). If necessary this number is adjusted according to the value of `expand`. For example, `fact=2` will result in a new Raster* object with `2*2=4` times fewer cells. If two numbers are supplied, e.g., `fact=c(2, 3)`, the first will be used for aggregating in the horizontal direction, and the second for aggregating in the vertical direction, and the returned object will have `2*3=6` times fewer cells. Likewise, `fact=c(2, 3, 4)` aggregates cells in groups of 2 (rows) by 3 (columns) and 4 (layers).
Aggregation starts at the upper-left end of a raster (you can use `flip` if you want to start elsewhere). If a division of the number of columns or rows with `factor` does not return an integer, the extent of the resulting Raster object will either be somewhat smaller or somewhat larger then the original RasterLayer. For example, if an input RasterLayer has 100 columns, and `fact=12`, the output Raster object will have either 8 columns (expand=FALSE) (using $8 \times 12 = 96$ of the original columns) or 9 columns (expand=TRUE). In both cases, the maximum x coordinate of the output RasterLayer would, of course, also be adjusted.

The function `fun` should take multiple numbers, and return a single number. For example `mean`, `modal`, `min` or `max`. It should also accept a `na.rm` argument (or ignore it as one of the `dots` arguments).

**Value**

RasterLayer or RasterBrick, or a SpatialPolygons* object

**Author(s)**

Robert J. Hijmans and Jacob van Etten

**See Also**

`disaggregate`, `resample`. For SpatialPolygons* `disaggregate`

**Examples**

```r
r <- raster()
# a new aggregated raster, no values
ra <- aggregate(r, fact=10)
r <- setValues(r, runif(ncell(r)))
# a new aggregated raster, max of the values
ra <- aggregate(r, fact=10, fun=max)

# multiple layers
s <- stack(r, r*2)
x <- aggregate(s, 2)

#SpatialPolygons
if (require(rgdal) & require(rgeos)) {
p <- shapefile(system.file("external/lux.shp", package="raster"))
p
pa0 <- aggregate(p)
pa0
pal <- aggregate(p, by="NAME_1", sums=list(list(mean, 'ID_2')))}
```
alignExtent

**Align an extent (object of class Extent)**

**Description**

Align an Extent object with the (boundaries of the) cells of a Raster* object.

**Usage**

```r
alignExtent(extent, object, snap='near')
```

**Arguments**

- `extent`: Extent object
- `object`: Raster* object
- `snap`: Character. One of 'near', 'in', or 'out', to determine in which direction the extent should be aligned. To the nearest border, inwards or outwards.

**Details**

Aligning an Extent object to another object assures that it gets the same origin and resolution. This should only be used to adjust objects because of imprecision in the data. `alignExtent` should not be used to force data to match that really does not match (use e.g. `resample` or (dis)aggregate for this).

**Value**

Extent object

**See Also**

`extent, drawExtent, Extent-class`

**Examples**

```r
r <- raster()
e <- extent(-10.1, 9.9, -20.1, 19.9)
ea <- alignExtent(e, r)
e
extent(r)
ea
```
animate

Animate layers of a Raster* object

Description

Animate (sequentially plot) the layers of a RasterStack or RasterBrick* object to create a movie

Usage

## S4 method for signature 'RasterStackBrick'
animate(x, pause=0.25, main, zlim, maxpixels=50000, n=10, ...)

Arguments

- **x**: Raster* object
- **pause**: numeric. How long should be the pause be between layers?
- **main**: title for each layer. If not supplied the z-value is used if available. Otherwise the names are used.
- **zlim**: numeric vector of length 2. Range of values to plot
- **maxpixels**: integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), sampleRegular is used before plotting
- **n**: integer > 0. Number of loops
- **...**: Additional arguments passed to `plot`

Value

None

See Also

`plot`, `spplot`, `plotRGB`

Examples

```r
b <- brick(system.file("external/rlogo.grd", package="raster"))
animate(b, n=1)
```
approxNA

Estimate values for cell values that are NA by interpolating between layers

Description

approxNA uses the stats function approx to estimate values for cells that are NA by interpolation across layers. Layers are considered equidistant, unless an argument 'z' is used, or getZ returns values, in which case these values are used to determine distance between layers.

For estimation based on neighbouring cells see focal

Usage

## S4 method for signature 'RasterStackBrick'
approxNA(x, filename="", method="linear", yleft, yright, rule=1, f=0, ties=mean, z=NULL, NArule=1, ...)

Arguments

- **x**: RasterStack or RasterBrick object
- **filename**: character. Output filename (optional)
- **method**: specifies the interpolation method to be used. Choices are "linear" or "constant" (step function; see the example in approx
- **yleft**: the value to be returned before a non-NA value is encountered. The default is defined by the value of rule given below
- **yright**: the value to be returned after the last non-NA value is encountered. The default is defined by the value of rule given below
- **rule**: an integer (of length 1 or 2) describing how interpolation is to take place at for the first and last cells (before or after any non-NA values are encountered). If rule is 1 then NAs are returned for such points and if it is 2, the value at the closest data extreme is used. Use, e.g., rule = 2:1, if the left and right side extrapolation should differ
- **f**: for method = "constant" a number between 0 and 1 inclusive, indicating a compromise between left- and right-continuous step functions. If y0 and y1 are the values to the left and right of the point then the value is y0*(1-f)+y1*f so that f = 0) is right-continuous and f = 1 is left-continuous
- **ties**: Handling of tied 'z' values. Either a function with a single vector argument returning a single number result or the string "ordered"
- **z**: numeric vector to indicate the distance between layers (e.g., time, depth). The default is 1:nlayers(x)
- **NArule**: single integer used to determine what to do when only a single layer with a non-NA value is encountered (and linear interpolation is not possible). The default value of 1 indicates that all layers will get this value for that cell; all other values do not change the cell values
- **...**: additional arguments as for writeRaster
Value

**RasterBrick**

See Also

- *focal*

Examples

```r
r <- raster(ncols=5, nrows=5)
r1 <- setValues(r, runif(ncell(r)))
r2 <- setValues(r, runif(ncell(r)))
r3 <- setValues(r, runif(ncell(r)))
r4 <- setValues(r, runif(ncell(r)))
r5 <- setValues(r, NA)
r6 <- setValues(r, runif(ncell(r)))
r1[6:10] <- NA
r2[5:15] <- NA
r3[8:25] <- NA
s <- stack(r1, r2, r3, r4, r5, r6)
s[1:5] <- NA
x1 <- approxNA(s)
x2 <- approxNA(s, rule=2)
x3 <- approxNA(s, rule=2, z=c(1,2,3,5,14,15))
```

---

**area**

*Size of cells*

Description

Raster objects: Compute the approximate surface area of cells in an unprojected (longitude/latitude) Raster object. It is an approximation because area is computed as the height (latitudinal span) of a cell (which is constant among all cells) times the width (longitudinal span) in the (latitudinal) middle of a cell. The width is smaller at the poleward side than at the equator-ward side of a cell. This variation is greatest near the poles and the values are thus not very precise for very high latitudes.

SpatialPolygons: Compute the area of the spatial features. Works for both planar and angular (lon/lat) coordinate reference systems

Usage

```r
## S4 method for signature 'RasterLayer'
area(x, filename="", na.rm=FALSE, weights=FALSE, ...)

## S4 method for signature 'RasterStackBrick'
area(x, filename="", na.rm=FALSE, weights=FALSE, ...)

## S4 method for signature 'SpatialPolygons'
area(x, ...)
```
Arith-methods

Arguments

- `x`: Raster* or SpatialPolygons object
- `filename`: character. Filename for the output Raster object (optional)
- `na.rm`: logical. If TRUE, cells that are NA are ignored
- `weights`: logical. If TRUE, the area of each cells is divided by the total area of all cells that are not NA
- `...`: additional arguments as for `writeRaster`

Details

If `x` is a RasterStack/Brick, a RasterBrick will be returned if `na.rm=TRUE`. However, if `na.rm=FALSE`, a RasterLayer is returned, because the values would be the same for all layers.

Value

- If `x` is a Raster* object: RasterLayer or RasterBrick. Cell values represent the size of the cell in km², or the relative size if `weights=TRUE`
- If `x` is a SpatialPolygons* object: area if each spatial object in squared meters if the CRS is longitude/latitude, or in squared map units (typically meter)

Examples

```r
r <- raster(nrow=18, ncol=36)
a <- area(r)

if (require(rgdal) & require(rgeos)) {
  p <- shapefile(system.file("external/lux.shp", package="raster"))
p$area <- round(area(p) / 1000000,1)
p$area
}
```

Description

Standard arithmetic operators for computations with Raster* objects and numeric values. The following operators are available: `+`, `-`, `*`, `/`, `^`, `%%`, `%/`

The input Raster* objects should have the same extent, origin and resolution. If only the extent differs, the computation will continue for the intersection of the Raster objects. Operators are applied on a cell by cell basis. For a RasterLayer, numeric values are recycled by row. For a RasterStack or RasterBrick, recycling is done by layer. RasterLayer objects can be combined RasterStack/Brick objects, in which case the RasterLayer is 'recycled'. When using multiple RasterStack or RasterBrick objects, the number of layers of these objects needs to be the same.

In addition to arithmetic with Raster* objects, the following operations are supported for SpatialPolygons* objects. Given SpatialPolygon objects `x` and `y`:
$x+y$ is the same as $\text{union}(x, y)$. For SpatialLines* and SpatialPoints* it is equivalent to $\text{bind}(x, y)$

$x*y$ is the same as $\text{intersect}(x, y)$

$x-y$ is the same as $\text{erase}(x, y)$

**Details**

If the values of the output Raster* cannot be held in memory, they will be saved to a temporary file. You can use `options` to set the default file format, datatype and progress bar.

**Value**

A Raster* object, and in some cases the side effect of a new file on disk.

**See Also**

Math-methods, overlay, calc

**Examples**

```r
r1 <- raster(ncols=10, nrows=10)
r1[] <- runif(ncell(r1))
r2 <- setValues(r1, 1:ncell(r1) / ncell(r1) )
r3 <- r1 + r2
r2 <- r1 / 10
r3 <- r1 * (r2 - 1 + r1^2 / r2)

# recycling by row
r4 <- r1 * 0 + 1:ncol(r1)

# multi-layer object multiplication, no recycling
b1 <- brick(r1, r2, r3)
b2 <- b1 * 10

# recycling by layer
b3 <- b1 + c(1, 5, 10)

# addition of the cell-values of two RasterBrick objects
b3 <- b2 + b1

# summing two RasterBricks and one RasterLayer. The RasterLayer is 'recycled'
b3 <- b1 + b2 + r1
```

---

**as.data.frame**

Get a data.frame with raster cell values, or coerce SpatialPolygons, Lines, or Points to a data.frame
Description

`as.matrix` returns all values of a Raster* object as a matrix. For RasterLayers, rows and columns in the matrix represent rows and columns in the RasterLayer object. For other Raster* objects, the matrix returned by `as.matrix` has columns for each layer and rows for each cell.

`as.array` returns an array of matrices that are like those returned by `as.matrix` for a RasterLayer.

If there is insufficient memory to load all values, you can use `getValues` or `getValuesBlock` to read chunks of the file. You could also first use `sampleRegular`.

The methods for Spatial* objects allow for easy creation of a data.frame with the coordinates and attributes; the default method only returns the attributes data.frame.

Usage

```r
## S4 method for signature 'Raster'
as.data.frame(x, row.names=NULL, optional=FALSE, xy=FALSE,
               na.rm=FALSE, long=FALSE, ...)

## S4 method for signature 'SpatialPolygons'
as.data.frame(x, row.names=NULL, optional=FALSE,
               xy=FALSE, centroids=TRUE, sepNA=FALSE, ...)

## S4 method for signature 'SpatialLines'
as.data.frame(x, row.names=NULL, optional=FALSE,
               xy=FALSE, sepNA=FALSE, ...)
```

Arguments

- `x` : Raster* object
- `row.names` : NULL or a character vector giving the row names for the data frame. Missing values are not allowed
- `optional` : logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional
- `xy` : logical. If TRUE, also return the spatial coordinates
- `na.rm` : logical. If TRUE, remove rows with NA values. This can be particularly useful for very large datasets with many NA values
- `long` : logical. If TRUE, values are reshaped from a wide to a long format
- `centroids` : logical. If TRUE return the centroids instead of all spatial coordinates (only relevant if `xy`=TRUE)
- `sepNA` : logical. If TRUE the parts of the spatial objects are separated by lines that are NA (only if `xy`=TRUE and, for polygons, if `centroids`=FALSE)
- `...` : Additional arguments (none)

Value

`data.frame`
Examples

```r
r <- raster(ncol=3, nrow=3)
r[] <- sqrt(1:ncell(r))
r[3:5] <- NA
as.data.frame(r)
s <- stack(r, r*2)
as.data.frame(s)
as.data.frame(s, na.rm=TRUE)
```

as.list  

Create a list of RasterLayer objects

Description

Create a list of RasterLayer objects from Raster* objects

Usage

```r
## S4 method for signature 'Raster'
as.list(x, ...)
```

Arguments

- `x`: Raster* object
- `...`: additional Raster* objects

Value

list

Examples

```r
r <- raster(ncol=3, nrow=3)
r[] <- 1:ncell(r)
as.list(r)

s <- stack(r,r*2,r*3)
as.list(s, r)
```
as.logical  Change cell values to logical or integer values

Description

Change values of a Raster* object to logical or integer values. With as.logical, zero becomes FALSE, all other values become TRUE. With as.integer values are truncated.

Usage

```r
## S4 method for signature 'Raster'
as.logical(x, filename='', ...)  
## S4 method for signature 'Raster'
as.integer(x, filename='', ...)  
```

Arguments

- `x` Raster* object
- `filename` character. Output filename (optional)
- `...` additional optional arguments as for `writeRaster`

See Also

- `as.logical`, `as.integer`

Examples

```r
r <- raster(nrow=10, ncol=10)
set.seed(0)
r[] <- runif(ncell(r)) * 10
r
r <- as.integer(r)
r
as.logical(r)
```

as.matrix  Get a vector, matrix, or array with raster cell values
as.matrix returns all values of a Raster* object as a matrix. For RasterLayers, rows and columns in the matrix represent rows and columns in the RasterLayer object. For other Raster* objects, the matrix returned by as.matrix has columns for each layer and rows for each cell.

as.array returns an array of matrices that are like those returned by as.matrix for a RasterLayer.

If there is insufficient memory to load all values, you can use getValues or getValuesBlock to read chunks of the file.

as.matrix and as.vector can also be used to obtain the coordinates from an Extent object.

Usage

as.matrix(x, ...)

as.array(x, ...)

## S4 method for signature 'Extent'
as.vector(x, mode='any')

## S4 method for signature 'Raster'
as.vector(x, mode='any')

Arguments

x Raster* or (for as.matrix and as.vector) Extent object
mode Character string giving an atomic mode (such as "numeric" or "character") or "list", or "any". Note: this argument is currently ignored!
... additional arguments:
maxpixels Integer. To regularly subsample very large objects
transpose Logical. Transpose the data? (for as.array only)

Value

matrix, array, or vector

Examples

r <- raster(ncol=3, nrow=3)
r[] <- 1:ncell(r)
as.matrix(r)
s <- stack(r,r)
as.array(s)
as.vector(extent(s))
as.raster

Coerce to a 'raster' object

Description
Implementation of the generic `as.raster` function to create a 'raster' (small r) object. NOT TO BE CONFUSED with the Raster* (big R) objects defined by the raster package! Such objects can be used for plotting with the `rasterImage` function.

Usage
`as.raster(x, ...)`

Arguments
- `x` RasterLayer object
- `...` Additional arguments.
  - `maxpixels` Integer. To regularly subsample very large objects
  - `col` Vector of colors. Default is `col=rev(terrain.colors(255))`

Value
'raster' object

Examples
```
r <- raster(ncol=3, nrow=3)
r[] <- 1:ncell(r)
as.raster(r)
```

atan2

Two argument arc-tangent

Description
For Raster* objects x and y, `atan2(y, x)` returns the angle in radians for the tangent y/x, handling the case when x is zero. See `Trig`

See `Math-methods` for other trigonometric and mathematical functions that can be used with Raster* objects.

Usage
`atan2(y, x)`
autocorrelation

Arguments

\( \text{y} \) Raster* object
\( \text{x} \) Raster* object

See Also

Math-methods

Examples

\[
\begin{align*}
\text{r1} & \leftarrow \text{r2} \leftarrow \text{raster(nrow=10, ncol=10)} \\
\text{r1[,]} & \leftarrow (\text{runif(ncell(r1)})-0.5) \times 10 \\
\text{r2[,]} & \leftarrow (\text{runif(ncell(r1)})-0.5) \times 10 \\
\text{atan2(r1, r2)}
\end{align*}
\]

autocorrelation Spatial autocorrelation

Description

Compute Moran’s I or Geary’s C measures of global spatial autocorrelation in a RasterLayer, or compute the the local Moran or Geary index (Anselin, 1995).

Usage

\[
\begin{align*}
\text{Geary(x, w=matrix(c(1,1,1,1,0,1,1,1,1), 3,3))} \\
\text{M Moran(x, w=matrix(c(1,1,1,1,0,1,1,1,1), 3,3))} \\
\text{M MoranLocal(x, w=matrix(c(1,1,1,1,0,1,1,1,1), 3,3))} \\
\text{G GearyLocal(x, w=matrix(c(1,1,1,1,0,1,1,1,1), 3,3))}
\end{align*}
\]

Arguments

\( \text{x} \) RasterLayer
\( \text{w} \) Spatial weights defined by or a rectangular matrix with odd length (3, 5, ...) sides (as in focal)

Details

The default setting uses a 3x3 neighborhood to compute "Queen’s case" indices. You can use a filter (weights matrix) to do other things, such as "Rook’s case", or different lags.

Value

A single value (Moran’s I or Geary’s C) or a RasterLayer (Local Moran or Geary values)

Author(s)

Robert J. Hijmans and Babak Naimi
**bands**

**References**


Geary, R.C., 1954. The contiguity ratio and statistical mapping. The Incorporated Statistician 5: 115-145


**See Also**

The spdep package for additional and more general approaches for computing indices of spatial autocorrelation

**Examples**

```r
r <- raster(nrows=10, ncols=10)
r[] <- 1:ncell(r)

Moran(r)
# Rook's case
f <- matrix(c(0,1,0,1,0,1,0,1,0), nrow=3)
Moran(r, f)

Geary(r)

x1 <- MoranLocal(r)

# Rook's case
x2 <- MoranLocal(r, w=f)
```

<table>
<thead>
<tr>
<th>bands</th>
<th>Number of bands</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

A ‘band’ refers to a single layer for a possibly multi-layer file. Most RasterLayer objects will refer to files with a single layer. The term 'band' is frequently used in remote sensing to refer to a variable (layer) in a multi-variable dataset as these variables typically represent reflection in different bandwidths in the electromagnetic spectrum. But in that context, bands could be stored in a single or in separate files. In the context of the raster package, the term band is equivalent to a layer in a raster file.

`nbands` returns the number of bands of the file that a RasterLayer points to (and 1 if it does not point at any file). This functions also works for a RasterStack for which it is equivalent to `nlayers`. `band` returns the specific band the RasterLayer refers to (1 if the RasterLayer points at single layer file or does not point at any file).
Bar plot of a RasterLayer

Description

Create a barplot of the values of a RasterLayer. For large datasets a regular sample with a size of approximately \( \text{maxpixels} \) is used.

Usage

```r
## S4 method for signature 'RasterLayer'
barplot(height, maxpixels=1000000, digits=0, breaks=NULL, col=rainbow, ...)
```

Arguments

- `height`: RasterLayer
- `maxpixels`: integer. To regularly subsample very large objects
- `digits`: integer used to determine how to round the values before tabulating. Set to NULL or to a large number if you do not want any rounding
- `breaks`: breaks used to group the data as in `cut`
- `col`: a color generating function such as `rainbow`, or a vector of colors
- `...`: additional arguments for plotting as in `barplot`
**Value**

A numeric vector (or matrix, when `beside = TRUE`) of the coordinates of the bar midpoints, useful for adding to the graph. See `barplot`

**See Also**

`hist`, `boxplot`

**Examples**

```r
f <- system.file("external/test.grd", package="raster")
r <- raster(f)
barplot(r, digits=-2, las=2, ylab='Frequency')

op <- par(no.readonly = TRUE)
par(mai = c(1, 2, .5, .5))
barplot(r, breaks=10, col=c('red', 'blue'), horiz=TRUE, digits=NULL, las=1)
par(op)
```

---

**bind**

*Bind Spatial* objects

**Description**

Bind (append) Spatial* objects into a single object. All objects must be of the same vector type base class (SpatialPoints, SpatialLines, or SpatialPolygons)

**Usage**

```r
## S4 method for signature 'SpatialPolygons,SpatialPolygons'
bind(x, y, ..., keepnames=FALSE)

## S4 method for signature 'SpatialLines,SpatialLines'
bind(x, y, ..., keepnames=FALSE)

## S4 method for signature 'SpatialPoints,SpatialPoints'
bind(x, y, ..., keepnames=FALSE)

## S4 method for signature 'data.frame,data.frame'
bind(x, y, ..., variables=NULL)

## S4 method for signature 'list,missing'
bind(x, y, ..., keepnames=FALSE)
```
Arguments

- **x**: Spatial* object or data.frame, or a list of Spatial* objects
- **y**: Spatial* object or data.frame, or missing
- **...**: Additional Spatial* objects
- **keepnames**: Logical. If TRUE the row.names are kept (if unique)
- **variables**: character. Variable (column) names to keep. If NULL, all variables are kept

Value

Spatial* object

See Also

merge

Examples

```r
if (require(rgdal) & require(rgdal)) {
  p <- shapefile(system.file("external/lux.shp", package="raster"))
  mersch <- p[p$NAME_2=="Mersch",]
  diekirch <- p[p$NAME_2=="Diekirch",]
  remich <- p[p$NAME_2=="Remich",]
  remich$NAME_1 <- NULL
  x <- bind(mersch, diekirch, remich)
  plot(x)
  data.frame(x)
}
```

blockSize

**Block size for writing files**

Description

This function can be used to suggest chunk sizes (always a number of entire rows), and corresponding row numbers, to be used when processing Raster* objects in chunks. Normally used together with `writeValues`.

Usage

```r
blockSize(x, chunksize, n=nlayers(x), minblocks=4, minrows=1)
```
boundaries

Arguments

x Raster* object
chunksize Integer, normally missing. Can be used to set the block size; unit is number of cells. Block size is then computed in units of number of rows (always >= 1)
n Integer. number of layers to consider. The function divides chunksize by n to determine blocksize
minblocks Integer. Minimum number of blocks
minrows Integer. Minimum number of rows in each block

Value

A list with three elements:

rows, the suggested row numbers at which to start the blocks for reading and writing,
nrows, the number of rows in each block, and,
n, the total number of blocks

See Also

writeValues

Examples

r <- raster(system.file("external/test.grd", package="raster"))
blockSize(r)

boundary detection

Description

Detect boundaries (edges). boundaries are cells that have more than one class in the 4 or 8 cells surrounding it, or, if classes=FALSE, cells with values and cells with NA.

Usage

## S4 method for signature 'RasterLayer'
boundaries(x, type='inner', classes=FALSE, directions=8, asNA=FALSE, filename="", ...)
Arguments

x  RasterLayer object

  type  character. 'inner' or 'outer'

  classes  character. Logical. If TRUE all different values are (after rounding) distinguished, as well as NA. If FALSE (the default) only edges between NA and non-NA cells are considered

  directions  integer. Which cells are considered adjacent? Should be 8 (Queen’s case) or 4 (Rook’s case)

  asNA  logical. If TRUE, non-edges are returned as NA instead of zero

  filename  character. Filename for the output RasterLayer (optional)

  ...  additional arguments as for writeRaster

Value

  RasterLayer. Cell values are either 1 (a border) or 0 (not a border), or NA

Note

  'edge' is obsolete and should not be used. It will be removed from this package

See Also

  focal, clump

Examples

  r <- raster(nrow=18, ncol=36, xmin=0)
  r[150:250] <- 1
  r[251:450] <- 2
  plot( boundaries(r, type='inner') )
  plot( boundaries(r, type='outer') )
  plot( boundaries(r, classes=TRUE) )
A RasterBrick is a multi-layer raster object. They are typically created from a multi-layer (band) file; but they can also exist entirely in memory. They are similar to a RasterStack (that can be created with the `stack` function), but processing time should be shorter when using a RasterBrick. Yet they are less flexible as they can only point to a single file.

A RasterBrick can be created from RasterLayer objects, from a RasterStack, or from a (multi-layer) file. The can also be created from SpatialPixels*, SpatialGrid*, and Extent objects, and from a three-dimensional array.

Usage

```r
## S4 method for signature 'character'
brick(x, ...)  
## S4 method for signature 'RasterStack'
brick(x, values=TRUE, nl, filename='', ...)  
## S4 method for signature 'RasterBrick'
brick(x, nl, ...)  
```
## S4 method for signature 'RasterLayer'

`brick(x, ..., values=TRUE, nl=1, filename='')`

## S4 method for signature 'missing'

`brick(nrows=180, ncols=360, xmn=-180, xmx=180, ymn=-90, ymx=90, nl=1, crs)`

## S4 method for signature 'Extent'

`brick(x, nrows=10, ncols=10, crs=NA, nl=1)`

## S4 method for signature 'array'

`brick(x, xmn=0, xmx=1, ymn=0, ymx=1, crs=NA, transpose=FALSE)`

## S4 method for signature 'big.matrix'

`brick(x, template, filename='', ...)`

## S4 method for signature 'SpatialGrid'

`brick(x)`

## S4 method for signature 'SpatialPixels'

`brick(x)`

### Arguments

- **x**: character (filename, see Details); Raster* object; missing; array; SpatialGrid*; SpatialPixels*; Extent; or list of Raster* objects. Supported file types are the ‘native’ raster package format and those that can be read via rgdal (see `readGDAL`), and NetCDF files (see details)
- **...**: see Details
- **values**: logical. If TRUE, the cell values of ‘x’ are copied to the RasterBrick object that is returned
- **nl**: integer > 0. How many layers should the RasterBrick have?
- **filename**: character. Filename if you want the RasterBrick to be saved on disk
- **nrows**: integer > 0. Number of rows
- **ncols**: integer > 0. Number of columns
- **xmn**: minimum x coordinate (left border)
- **xmx**: maximum x coordinate (right border)
- **ymn**: minimum y coordinate (bottom border)
- **ymx**: maximum y coordinate (top border)
- **crs**: character or object of class CRS. PROJ4 type description of a Coordinate Reference System (map projection). If this argument is missing, and the x coordinates are within -360 .. 360 and the y coordinates are within -90 .. 90, "+proj=longlat +datum=WGS84" is used
- **transpose**: if TRUE, the values in the array are transposed
- **template**: Raster* object used to set the extent, number of rows and columns and CRS
buffer

Details
If x is a RasterLayer, the additional arguments can be used to pass additional Raster* objects. If there is a filename argument, the additional arguments are as for writeRaster. The big.matrix most have rows representing cells and columns representing layers. If x represents a filename there is the following additional argument:
native: logical. If TRUE (not the default), reading and writing of IDRISI, BIL, BSQ, BIP, and Arc ASCII files is done with native (raster package) drivers, rather then via rgdal.
In addition, if x is a NetCDF filename there are the following additional arguments:
varname: character. The variable name (e.g. ‘altitude’ or ‘precipitation’. If not supplied and the file has multiple variables are a guess will be made (and reported))
1var: integer > 0 (default=3). To select the ‘level variable’ (3rd dimension variable) to use, if the file has 4 dimensions (e.g. depth instead of time)
1level: integer > 0 (default=1). To select the ‘level’ (4th dimension variable) to use, if the file has 4 dimensions, e.g. to create a RasterBrick of weather over time at a certain height.
To use NetCDF files the ncdf4 package needs to be available. It is assumed that these files follow, or are compatible with the CF-1 convention.

Value
RasterBrick

See Also
raster

Examples
b <- brick(system.file("external/rlogo.grd", package="raster"))
b
nlayers(b)
names(b)
extract(b, 870)

Description
Calculate a buffer around all cells that are not NA or around SpatialPoints, Lines, or Polygons.
Note that the distance unit of the buffer width parameter is meters if the RasterLayer is not projected (+proj=longlat), and in map units (typically also meters) when it is projected.
Except for SpatialLines and SpatialPolygons that are currently handled by rgeos, and can only deal with planar coordinate reference systems.
Usage

```r
## S4 method for signature 'RasterLayer'
buffer(x, width=0, filename='', doEdge=FALSE, ...)

## S4 method for signature 'Spatial'
buffer(x, width=1, dissolve=TRUE, ...)
```

Arguments

- **x**: RasterLayer or Spatial* object
- **width**: numeric > 0. Unit is meter if x has a longitude/latitude CRS, or mapunits in other cases
- **filename**: character. Filename for the output RasterLayer (optional)
- **doEdge**: logical. If TRUE, the `boundaries` function is called first. This may be efficient in cases where you compute a buffer around very large areas because boundaries determines the edge cells that matter for distance computation
- **dissolve**: logical. If TRUE, buffer geometries of overlapping polygons are dissolved and all geometries are aggregated and attributes (the data.frame) are dropped
- **...**: Additional arguments as for `writeRaster`

Value

RasterLayer or SpatialPolygons* object

See Also

`distance`, `gridDistance`, `pointDistance`

Examples

```r
r <- raster(ncol=36,nrow=18)
r[] <- NA
r[500] <- 1
b <- buffer(r, width=5000000)
#plot(b)
```

### Description

Calculate values for a new Raster* object from another Raster* object, using a formula.

If x is a RasterLayer, fun is typically a function that can take a single vector as input, and return a vector of values of the same length (e.g. `sqrt`). If x is a RasterStack or RasterBrick, fun should operate on a vector of values (one vector for each cell). `calc` returns a RasterLayer if fun returns
calc

a single value (e.g. sum) and it returns a RasterBrick if fun returns more than one number, e.g.,
fun=quantile.

In many cases, what can be achieved with calc, can also be accomplished with a more intuitive
'raster-algebra' notation (see Arith-methods). For example, r <- r * 2 instead of
r <- calc(r, fun=function(x){x * 2}, or r <- sum(s) instead of
r <- calc(s, fun=sum). However, calc should be faster when using complex formulas on large
datasets. With calc it is possible to set an output filename and file type preferences.
See (overlay) to use functions that refer to specific layers, like (function(a,b,c){a + sqrt(b) / c})

Usage

## S4 method for signature 'Raster, function'
calc(x, fun, filename='', na.rm, forcefun=FALSE, forceapply=FALSE, ...)

Arguments

x Raster* object
fun function
filename character. Output filename (optional)
na.rm Remove NA values, if supported by 'fun' (only relevant when summarizing a
multilayer Raster object into a RasterLayer)
forcefun logical. Force calc to not use fun with apply; for use with ambiguous functions
and for debugging (see Details)
forceapply logical. Force calc to use fun with apply; for use with ambiguous functions and
for debugging (see Details)
... Additional arguments as for writeRaster

Details

The intent of some functions can be ambiguous. Consider:
library(raster)
r <- raster(volcano)
calc(r, function(x) x * 1:10)

In this case, the cell values are multiplied in a vectorized manner and a single layer is returned
where the first cell has been multiplied with one, the second cell with two, the 11th cell with one
again, and so on. But perhaps the intent was to create 10 new layers (x*1, x*2, ...)? This can
be achieved by using argument forceapply=TRUE

calc(r, function(x) x * 1:10), forceapply=TRUE

Value

a Raster* object
Note

For large objects `calc` will compute values chunk by chunk. This means that for the result of `fun` to be correct it should not depend on having access to _all_ values at once. For example, to scale the values of a `Raster*` object by subtracting its mean value (for each layer), you would _not_ do, for `Raster` object `x`:

```r
calc(x, function(x)scale(x, scale=FALSE))
```

Because the mean value of each chunk will likely be different. Rather do something like

```r
m <- cellStats(x, 'mean')
x <- m
```

Author(s)

Robert J. Hijmans and Matteo Mattiuzzi

See Also

`overlay`, `reclassify`, `Arith-methods`, `Math-methods`

Examples

```r
r <- raster(ncols=36, nrows=18)
r[] <- 1:ncell(r)

# multiply values with 10
fun <- function(x) { x * 10 }
rc1 <- calc(r, fun)

# set values below 100 to NA.
fun <- function(x) { x[x<100] <- NA; return(x) }
rc2 <- calc(r, fun)

# set NA values to -9999
fun <- function(x) { x[is.na(x)] <- -9999; return(x) }
rc3 <- calc(rc2, fun)

# using a RasterStack as input
s <- stack(r, r*r, sqrt(r))
# return a RasterLayer
rs1 <- calc(s, sum)

# return a RasterBrick
rs2 <- calc(s, fun=function(x)(x * 10))
# recycling by layer
rs3 <- calc(s, fun=function(x)(x * c(1, 5, 10)))

# use overlay when you want to refer to individual layer in the function
# but it can be done with calc:
rs4 <- calc(s, fun=function(x)(x[1]+x[2]*x[3]))
```
# Some regression examples

##

## create data

```r
r <- raster(nrow=10, ncol=10)
s1 <- lapply(1:12, function(i) setValues(r, rnorm(ncell(r), i, 3)))
s2 <- lapply(1:12, function(i) setValues(r, rnorm(ncell(r), i, 3)))
s1 <- stack(s1)
s2 <- stack(s2)
```

# regression of values in one brick (or stack) with another

```r
s <- stack(s1, s2)
# s1 and s2 have 12 layers; coefficients[2] is the slope
x1 <- calc(s, fun)
```

# regression of values in one brick (or stack) with 'time'

```r
time <- 1:nlayers(s)
fun <- function(x) { lm(x ~ time)$coefficients[2] }
x2 <- calc(s, fun)
```

# get multiple layers, e.g. the slope _and_ intercept

```r
fun <- function(x) { lm(x ~ time)$coefficients }
x3 <- calc(s, fun)
```

### A much (> 100 times) faster approach is to directly use
### linear algebra and pre-compute some constants

```r
# add 1 for a model with an intercept
X <- cbind(1, time)

# pre-computing constant part of least squares
invXtX <- solve(t(X) %% X) %*% t(X)

# much reduced regression model; [2] is to get the slope
quickfun <- function(y) (invXtX %*% y)[2]
x4 <- calc(s, quickfun)
```

---

**cellFrom**

*Get cell, row, or column number*

**Description**

Get cell number(s) of a Raster* object from row and/or column numbers. Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the Raster* object.
Usage

cellFromRowCol(object, rownr, colnr)
cellFromRowColCombine(object, rownr, colnr)
cellFromRow(object, rownr)
cellFromCol(object, colnr)
colFromX(object, x)
rowFromY(object, y)
cellFromXY(object, xy)
cellFromLine(object, lns)
cellFromPolygon(object, p, weights=FALSE)
fourCellsFromXY(object, xy, duplicates=TRUE)

Arguments

object Raster* object (or a SpatialPixels* or SpatialGrid* object)
colnr column number; or vector of column numbers
rownr row number; or vector of row numbers
x x coordinate(s)
y y coordinate(s)
xy matrix of x and y coordinates, or a SpatialPoints or SpatialPointsDataFrame object
lns SpatialLines object
p SpatialPolygons object
weights Logical. If TRUE, the fraction of each cell that is covered is also returned
duplicates Logical. If TRUE, the same cell number can be returned twice (if the point in the middle of a division between two cells) or four times (if a point is in the center of a cell)

Details

cellFromRowCol returns the cell numbers obtained for each row / col number pair. In contrast, cellFromRowColCombine returns the cell numbers obtained by the combination of all row and column numbers supplied as arguments.
fourCellsFromXY returns the four cells that are nearest to a point (if the point falls on the raster). Also see adjacent.

Value

vector of row, column or cell numbers. cellFromLine and cellFromPolygon return a list, fourCellsFromXY returns a matrix.

See Also

xyFromCell, cellsFromExtent, rowColFromCell
cellsFromExtent

Examples

```r
r <- raster(ncols=10, nrows=10)
cellFromRowCol(r, 5, 5)
cellFromRowCol(r, 1:2, 1:2)
cellFromRowColCombine(r, 1:3, 1:2)
cellFromCol(r, 1)
cellFromRow(r, 1)

colFromX(r, 0.5)
rowFromY(r, 0.5)
cellFromXY(r, cbind(c(0.5,5), c(15, 88)))
fourCellsFromXY(r, cbind(c(0.5,5), c(15, 88)))

cds1 <- rbind(c(-180,-20), c(-160,5), c(-60,0), c(-160,-60), c(-180,-20))
cds2 <- rbind(c(80,0), c(100,60), c(120,0), c(120,-55), c(80,0))
pols <- SpatialPolygons(list(Polygons(list(Polygon(cds1)), 1), Polygons(list(Polygon(cds2)), 2)))
cellFromPolygon(r, pols)
```

---

**cellsFromExtent**  
*Cells from extent, and vice versa*

**Description**

cellsFromExtent returns the cell numbers for a Raster* object that are within a specified extent (rectangular area), supply an object of class Extent, or another Raster* object.

extentFromCells returns an Extent object from a Raster* object and cell numbers. All cells are within the returned Extent.

**Usage**

cellsFromExtent(object, extent, expand=FALSE)
extentFromCells(object, cells)

**Arguments**

- `object`  
  A Raster* object

- `extent`  
  An object of class Extent (which you can create with newExtent(), or another Raster* object)

- `expand`  
  Logical. If TRUE, NA is returned for (virtual) cells implied by `bbox`, that are outside the RasterLayer (object). If FALSE, only cell numbers for the area where object and `bbox` overlap are returned (see `intersect`)

- `cells`  
  numeric. A vector of cell numbers

**Value**

- a vector of cell numbers
cellStats

Statistics across cells

Description

Compute statistics for the cells of each layer of a Raster* object. In the raster package, functions such as max, min, and mean, when used with Raster* objects as argument, return a new Raster* object (with a value computed for each cell). In contrast, cellStats returns a single value, computed from the all the values of a layer. Also see layerStats

Usage

## S4 method for signature 'RasterLayer'
cellStats(x, stat='mean', na.rm=TRUE, asSample=TRUE, ...)

## S4 method for signature 'RasterStackBrick'
cellStats(x, stat='mean', na.rm=TRUE, asSample=TRUE, ...)

Arguments

- **x**: Raster* object
- **stat**: The function to be applied. See Details
- **na.rm**: Logical. Should NA values be removed?
- **asSample**: Logical. Only relevant for stat=sd in which case, if TRUE, the standard deviation for a sample (denominator is n-1) is computed, rather than for the population (denominator is n)
- **...**: Additional arguments

Details

cellStats will fail (gracefully) for very large Raster* objects except for a number of known functions: sum, mean, min, max, sd, 'skew' and 'rms'. 'skew' (skewness) and 'rms' (Root Mean Square) must be supplied as a character value (with quotes), the other known functions may be supplied with or without quotes. For other functions you could perhaps use a sample of the RasterLayer that can be held in memory (see sampleRegular)
clamp

Value

Numeric

See Also

freq, quantile, minValue, maxValue, setMinMax

Examples

r <- raster(nrow=18, ncol=36)
r[] <- runif(nccell(r)) * 10
# works for large files
cellStats(r, 'mean')
# same, but does not work for very large files
cellStats(r, mean)
# multi-layer object
cellStats(brick(r, r), mean)

clamp

Clamp values

Description

Clamp values to a minimum and maximum value. That is, all values below the lower clamp value and the upper clamp value become NA (or the lower/upper value if useValues=TRUE)

Usage

```r
## S4 method for signature 'Raster'
clamp(x, lower=-Inf, upper=Inf, useValues=TRUE, filename='', ...) 
## S4 method for signature 'numeric'
clamp(x, lower=-Inf, upper=Inf, ...) 
```

Arguments

- `x` RasterLayer, or numeric vector
- `lower` numeric, lowest value
- `upper` numeric, highest value
- `useValues` logical. If FALSE values outside the clamping range become NA, if TRUE, they get the extreme values
- `filename` character. Filename for the output RasterLayer (optional)
- `...` additional arguments as for `writeRaster`

Value

Raster object
clearValues

See Also

reclassify

Examples

```r
r <- raster(ncols=12, nrows=12)
values(r) <- 1:ncell(r)
rc <- clamp(r, 25, 75)
rc
```

clearValues  Clear values

Description

Clear cell values of a Raster* object from memory

Usage

clearValues(x)

Arguments

x  Raster* object

Value

a Raster* object

See Also

values, replacement

Examples

```r
r <- raster(ncol=10, nrow=10)
r[] <- 1:ncell(r)
r <- clearValues(r)
```
Description

Click on a map (plot) to get values of a Raster* or Spatial* object at that location; and optionally the coordinates and cell number of the location. For SpatialLines and SpatialPoints you need to click twice (draw a box).

Usage

```r
## S4 method for signature 'Raster'
click(x, n=Inf, id=FALSE, xy=FALSE, cell=FALSE, type="n", show=TRUE, ...)

## S4 method for signature 'SpatialGrid'
click(x, n=1, id=FALSE, xy=FALSE, cell=FALSE, type="n", ...)

## S4 method for signature 'SpatialPolygons'
click(x, n=1, id=FALSE, xy=FALSE, cell=FALSE, type="n", ...)

## S4 method for signature 'SpatialLines'
click(x, ...)

## S4 method for signature 'SpatialPoints'
click(x, ...)
```

Arguments

- `x` Raster*, or Spatial* object (or missing)
- `n` number of clicks on the map
- `id` Logical. If `TRUE`, a numeric ID is shown on the map that corresponds to the row number of the output
- `xy` Logical. If `TRUE`, xy coordinates are included in the output
- `cell` Logical. If `TRUE`, cell numbers are included in the output
- `type` One of "n", "p", "l" or "o". If "p" or "o" the points are plotted; if "l" or "o" they are joined by lines. See ?locator
- `show` logical. Print the values after each click?
- `...` additional graphics parameters used if type != "n" for plotting the locations. See ?locator

Value

The value(s) of `x` at the point(s) clicked on (or touched by the box drawn).
**Note**

The plot only provides the coordinates for a spatial query, the values are read from the Raster* or Spatial* object that is passed as an argument. Thus you can extract values from an object that has not been plotted, as long as it spatially overlaps with the extent of the plot.

Unless the process is terminated prematurely values at at most $n$ positions are determined. The identification process can be terminated by clicking the second mouse button and selecting 'Stop' from the menu, or from the 'Stop' menu on the graphics window.

**See Also**

`select`, `drawExtent`

**Examples**

```r
## Not run:
r <- raster(system.file("external/test.grd", package="raster"))
plot(r)
click(r)
# now click on the plot (map)

## End(Not run)
```

---

**clump**

Detect clumps (patches) of connected cells. Each clump gets a unique ID. NA and zero are used as background values (i.e. these values are used to separate clumps). You can use queen’s or rook’s case, using the `directions` argument. For larger files that are processed in chunks, the highest clump number is not necessarily equal to the number of clumps (unless you use argument `gaps`=FALSE).

**Usage**

```r
## S4 method for signature 'RasterLayer'
clump(x, filename="", directions=8, gaps=TRUE, ...)
```

**Arguments**

- `x`  
  RasterLayer
- `filename`  
  Character. Filename for the output RasterLayer (optional)
- `directions`  
  Integer. Which cells are considered adjacent? Should be 8 (Queen’s case) or 4 (Rook’s case)
- `gaps`  
  Logical. If TRUE (the default), there may be 'gaps' in the chunk numbers (e.g. you may have clumps with IDs 1, 2, 3 and 5, but not 4). If it is FALSE, these numbers will be recoded from 1 to n (4 in this example)
- `...`  
  Additional arguments as for `writeRaster`
cluster

Value

RasterLayer

Note

This function requires that the igraph package is available.

Author(s)

Robert J. Hijmans and Jacob van Etten

Examples

```r
r <- raster(ncols=12, nrows=12)
set.seed(0)
r[] <- round(runif(ncell(r)) * 0.7)
rc <- clump(r)
freq(rc)
plot(rc)
```

cluster

Use a multi-core cluster

Description

beginCluster creates, and endCluster deletes a 'snow' cluster object. This object can be used for multi-core computing with those 'raster' functions that support it.

beginCluster determines the number of nodes (cores) that are available and uses all of them (unless the argument n is used).

NOTE: beginCluster may fail when the package 'nws' is installed. You can fix that by removing the 'nws' package, or by setting the cluster type manually, e.g. beginCluster(type="SOCK")

endCluster closes the cluster and removes the object.

The use of the cluster is automatic in these functions: projectRaster, resample and in extract when using polygons.

clusterR is a flexible interface for using cluster with other functions. This function only works with functions that have a Raster* object as first argument and that operate on a cell by cell basis (i.e., there is no effect of neighboring cells) and return an object with the same number of cells as the input raster object. The first argument of the function called must be a Raster* object. There can only be one Raster* object argument. For example, it works with calc and it also works with overlay as long as you provide a single RasterStack or RasterBrick as the first argument.

This function is particularly useful to speed up computations in functions like predict, interpolate, and perhaps calc.

Among other functions, it does _not_ work with merge, crop, mosaic, (dis)aggregate, resample, projectRaster, focal, distance, buffer, direction. But note that projectRaster has a build-in capacity for clustering that is automatically used if beginCluster() has been called.
Usage

beginCluster(n, type='SOCK', nice, exclude)
endCluster()
clusterR(x, fun, args=NULL, export=NULL, filename='', cl=NULL, m=2, ...)

Arguments

n
  Integer. The number of nodes to be used (optional)
type
  Character. The cluster type to be used
nice
  Integer. To set the priority for the workers, between -20 and 20 (UNIX like platforms only)
exclude
  Character. Packages to exclude from loading on the nodes (because they may fail there) but are required/loaded on the master
x
  Raster* object
fun
  function that takes x as its first argument
args
  list with the arguments for the function (excluding x, which should always be the first argument
export
  character. Vector of variable names to export to the cluster nodes such that the are visible to fun (e.g. a parameter that is not passed as an argument)
filename
  character. Output filename (optional)
cl
  cluster object (do not use it if beginCluster() has been called
m
  tuning parameter to determine how many blocks should be used. The number is rounded and multiplied with the number of nodes.
...  
  additional arguments as for writeRaster

Value

beginCluster and endCluster: None. The side effect is to create or delete a cluster object.
clusterR: as for the function called with argument fun

Note

If you want to write your own cluster-enabled functions see getCluster, returnCluster, and the vignette about writing functions.

Author(s)

Matteo Mattiuzzi and Robert J. Hijmans

Examples

## Not run:
# set up the cluster object for parallel computing
beginCluster()
r <- raster()
r[] <- 1:ncell(r)

x <- clusterR(r, sqrt, verbose=T)

f1 <- function(x) calc(x, sqrt)
y <- clusterR(r, f1)

s <- stack(r, r*2, r*3)
f2 <- function(d,e,f) (d + e) / (f * param)
param <- 122
ov <- clusterR(s, overlay, args=list(fun=f2), export='param')

pts <- matrix(c(0,0, 45,45), ncol=2, byrow=T)
d <- clusterR(r, distanceFromPoints, args=list(xy=pts))

values(r) <- runif(ncell(r))
m <- c(0, 0.25, 1, 0.25, 0.5, 2, 0.5, 1, 3)
m <- matrix(m, ncol=3, byrow=TRUE)
rc1 <- clusterR(r, reclassify, args=list(rcl=m, right=FALSE),
                filename='rc1test.grd', datatype='INT2S', overwrite=TRUE)

# equivalent to:
rc2 <- reclassify(r, rcl=m, right=FALSE, filename='rc2test.grd', datatype='INT2S', overwrite=TRUE)

# example with the calc function
a <- 10
f3 <- function(x) sum(x)+a

z1 <- clusterR(s, calc, args=list(fun=f3), export='a')

# for some raster functions that use another function as an argument
# you can write your own parallel function instead of using clusterR
# get cluster object created with beginCluster
cl <- getCluster()
library(parallel)
clusterExport(cl, "a")
z2 <- calc(s, fun=function(x){ parApply(cl, x, 1, f3) })
# set flag that cluster is available again
returnCluster()

# done with cluster object
endCluster()

## End(Not run)
**Description**

Get or set the colortable of a RasterLayer. A colortable is a vector of 256 colors in the RGB triple format as returned by the `rgb` function (e.g. "#C4CDDA").

When setting the colortable, it is assumed that the values are integers in the range [0,255]

**Usage**

```r
colortable(x)
colortable(x) <- value
```

**Arguments**

- `x`: RasterLayer object
- `value`: vector of 256 character values

**See Also**

`plotRGB`

**Examples**

```r
r <- raster(ncol=10, nrow=10)
values(r) <- sample(0:255, ncell(r), replace=TRUE)
ctab <- sample(rainbow(256))
colortable(r) <- ctab
plot(r)
head(colortable(r))
```

---

**Compare-methods**

**Compare Raster* objects**

**Description**

These methods compare the location and resolution of Raster* objects. That is, they compare their spatial extent, projection, and number of rows and columns.

For `basicRaster` objects you can use `==` and `!`, the values returned is a single logical value `TRUE` or `FALSE`

For RasterLayer objects, these operators also compare the values associated with the objects, and the result is a RasterLayer object with logical (Boolean) values.

The following methods have been implemented for RasterLayer objects:

```r
==, !=, >, <, <=, >=
```

**Value**

A logical value or a RasterLayer object, and in some cases the side effect of a new file on disk.
Examples

```r
r1 <- raster()
r1 <- setValues(r1, round(10 * runif(ncell(r1))))
r2 <- setValues(r1, round(10 * runif(ncell(r1))))
as(r1, 'BasicRaster') == as(r2, 'BasicRaster')

r3 <- r1 == r2

b <- extent(0, 360, 0, 180)
r4 <- setExtent(r2, b)
as(r2, 'BasicRaster') != as(r4, 'BasicRaster')
# The following would give an error. You cannot compare RasterLayer
# that do not have the same BasicRaster properties.
# r3 <- r1 > r4
```

**compareCRS**

*Partially compare two CRS objects*

Description

Compare CRS objects

Usage

```r
compareCRS(x, y, unknown=FALSE, verbatim=FALSE, verbose=FALSE)
```

Arguments

- `x` CRS object, or object from which it can be extracted with `projection`, or PROJ.4 format character string
- `y` same as `x`
- `unknown` logical. Return TRUE if `x` or `y` is TRUE
- `verbatim` logical. If TRUE compare `x` and `y`, verbatim (not partially)
- `verbose` logical. If TRUE, messages about the comparison may be printed

Value

logical

See Also

crs
Examples

```r
compareCRS("+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84", 
  "+proj=longlat +datum=WGS84")
compareCRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0", 
  "+proj=longlat +datum=WGS84")
compareCRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0", 
  "+proj=longlat +datum=WGS84", verbatim=TRUE)
compareCRS("+proj=longlat +datum=WGS84", NA)
compareCRS("+proj=longlat +datum=WGS84", NA, unknown=TRUE)
```

**compareRaster**  
*Compare Raster objects*

**Description**

Evaluate whether two or more Raster* objects have the same extent, number of rows and columns, projection, resolution, and origin (or a subset of these comparisons).

all.equal is a wrapper around compareRaster with options values=TRUE, stopiffalse=FALSE and showwarning=TRUE.

**Usage**

```r
compareRaster(x, ..., extent=TRUE, rowcol=TRUE, crs=TRUE, res=FALSE, orig=FALSE, 
  rotation=TRUE, values=FALSE, tolerance, stopiffalse=TRUE, showwarning=FALSE)
```

**Arguments**

- **x**: Raster* object
- **...**: Raster* objects
- **extent**: logical. If TRUE, bounding boxes are compared
- **rowcol**: logical. If TRUE, number of rows and columns of the objects are compared
- **crs**: logical. If TRUE, coordinate reference systems are compared.
- **res**: logical. If TRUE, resolutions are compared (redundant when checking extent and rowcol)
- **orig**: logical. If TRUE, origins are compared
- **rotation**: logical. If TRUE, rotations are compared
- **values**: logical. If TRUE, cell values are compared
- **tolerance**: numeric between 0 and 0.5. If not supplied, the default value is used (see `rasterOptions`). It sets difference (relative to the cell resolution) that is permissible for objects to be considered 'equal', if they have a non-integer origin or resolution. See all.equal.
- **stopiffalse**: logical. If TRUE, an error will occur if the objects are not the same
- **showwarning**: logical. If TRUE, an warning will be given if objects are not the same. Only relevant when stopiffalse is TRUE
Examples

```r
r1 <- raster()
r2 <- r1
r3 <- r1
compareRaster(r1, r2, r3)
nrow(r3) <- 10

# compareRaster(r1, r3)
compareRaster(r1, r3, stopiffalse=FALSE)
compareRaster(r1, r3, rowcol=FALSE)

all.equal(r1, r2)
all.equal(r1, r3)
```

Description

Contour plot of a RasterLayer.

Usage

```r
## S4 method for signature 'RasterLayer'
contour(x, maxpixels=100000, ...)
```

Arguments

- `x`: Raster* object
- `maxpixels`: maximum number of pixels used to create the contours
- `...`: any argument that can be passed to `contour` (graphics package)

See Also

`persp`, `filledContour`, `rasterToContour`

Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))
plot(r)
contour(r, add=TRUE)
```
Description

Local correlation coefficient for two RasterLayer objects (using a focal neighborhood) or for two RasterStack or Brick objects (with the same number of layers (> 2))

Usage

```r
## S4 method for signature 'RasterLayer,RasterLayer'
corLocal(x, y, ngb=5,
    method=c("pearson", "kendall", "spearman"), test=FALSE, filename='', ...)  

## S4 method for signature 'RasterStack,RasterStack'
corLocal(x, y,
    method=c("pearson", "kendall", "spearman"), test=FALSE, filename='', ...)  
```

Arguments

- `x`: RasterLayer or RasterStack/RasterBrick
- `y`: object of the same class as `x`, and with the same number of layers
- `ngb`: neighborhood size. Either a single integer or a vector of two integers `c(nrow, ncol)`
- `method`: character indicating which correlation coefficient is to be used. One of "pearson", "kendall", or "spearman"
- `test`: logical. If TRUE, return a p-value
- `filename`: character. Output filename (optional)
- `...`: additional arguments as for `writeRaster`

Value

RasterLayer

Note

NA values are omitted

See Also

`cor`, `cor.test`
Examples

```r
b <- stack(system.file("external/rlogo.grd", package="raster"))
b <- aggregate(b, 2, mean)

set.seed(0)
b[[2]] <- flip(b[[2]], 'y') + runif(ncell(b))
b[[1]] <- b[[1]] + runif(ncell(b))

x <- corLocal(b[[1]], b[[2]], test=TRUE )
# plot(x)

# only cells where the p-value < 0.1
xm <- mask(x[[1]], x[[2]] < 0.1, maskvalue=FALSE)
plot(xm)

# for global correlation, use the cor function
x <- as.matrix(b)
cor(x, method="spearman")

# use sampleRegular for large datasets
x <- sampleRegular(b, 1000)
cor.test(x[,1], x[,2])

# RasterStack or Brick objects
y <- corLocal(b, flip(b, 'y'))
```

---

Replace NA values with values of other layers

Description

For Raster* objects: Replace NA values in the first Raster object (x) with the values of the second (y), and so forth for additional Rasters. If x has multiple layers, the subsequent Raster objects should have the same number of layers, or have a single layer only (which will be recycled).

For SpatialPolygons* objects: Areas of x that overlap with y are replaced by (or intersected with) y.

Usage

```r
## S4 method for signature 'RasterLayer,RasterLayer'
cover(x, y, ..., filename='')

## S4 method for signature 'RasterStackBrick,Raster'
cover(x, y, ..., filename='')

## S4 method for signature 'SpatialPolygons,SpatialPolygons'
cover(x, y, ..., identity=FALSE)
```
Arguments

x  Raster* or SpatialPolygons* object
y  Same as x
filename  character. Output filename (optional)

...  Same as x. If x is a Raster* object, also additional arguments as for writeRaster
identity  logical. If TRUE overlapping areas are intersected rather than replaced

Value

RasterLayer or RasterBrick object, or SpatialPolygons object

Examples

C raster objects

r1 <- raster(ncols=36, nrows=18)
r1[] <- 1:ncell(r1)
r2 <- setValues(r1, runif(ncell(r1)))
r2[r2 < 0.5] <- NA
r3 <- cover(r2, r1)

Cspatialpolygons

if (require(rgdal) & require(rgeos)) {
  p <- shapefile(system.file("external/lux.shp", package="raster"))
  b <- as(extent(6, 6.4, 49.75, 50), 'SpatialPolygons')
  crs(b) <- crs(p)
  b <- SpatialPolygonsDataframe(b, data.frame(ID_1=9))
  cv1 <- cover(p, b)
  cv2 <- cover(p, b, identity=TRUE)
}

crop  Crop

Description

crop returns a geographic subset of an object as specified by an Extent object (or object from which an extent object can be extracted/created). If x is a Raster* object, the Extent is aligned to x. Areas included in y but outside the extent of x are ignored (see extend if you want a larger area).

Usage

## S4 method for signature 'Raster'
crop(x, y, filename="", snap='near', datatype=NULL, ...)

## S4 method for signature 'Spatial'
crop(x, y, ...)
crop

Arguments

x  
Raster* object or SpatialPolygons*, SpatialLines*, or SpatialPoints* object

y  
Extent object, or any object from which an Extent object can be extracted (see Details)

filename  
Character, output filename. Optional

ap  
Character. One of 'near', 'in', or 'out', for use with alignExtent

datatype  
Character. Output dataType (by default it is the same as the input datatype)

...  
Additional arguments as for writeRaster

Details

Objects from which an Extent can be extracted/created include RasterLayer, RasterStack, RasterBrick and objects of the Spatial* classes from the sp package. You can check this with the extent function. New Extent objects can be also be created with function extent and drawExtent by clicking twice on a plot.

To crop by row and column numbers you can create an extent like this (for Raster x, row 5 to 10, column 7 to 12) crop(x, extent(x, 5, 10, 7, 15))

Value

RasterLayer or RasterBrick object; or SpatialLines or SpatialPolygons object.

Note

values within the extent of a Raster* object can be set to NA with mask

See Also

extend, merge

Examples

r <- raster(nrow=45, ncol=90)
r[] <- 1:ncell(r)
e <- extent(-160, 10, 30, 60)
rc <- crop(r, e)

# use row and column numbers:
rc2 <- crop(r, extent(r, 5, 10, 7, 15))

# crop Raster* with Spatial* object
b <- as(extent(6, 6.4, 49.75, 50), 'SpatialPolygons')
crs(b) <- crs(r)
rb <- crop(r, b)

# crop a SpatialPolygon* object with another one
if (require(rgdal) & require(rgeos)) {
  p <- shapefile(system.file("external/lux.shp", package="raster"))
pb <- crop(p, b)
Cross-tabulate two RasterLayer objects, or multiple layers in a RasterStack or RasterBrick to create a contingency table.

Usage

```r
## S4 method for signature 'Raster,Raster'
crosstab(x, y, digits=0, long=FALSE, useNA=FALSE, progress='', ...)

## S4 method for signature 'RasterStackBrick,missing'
crosstab(x, digits=0, long=FALSE, useNA=FALSE, progress='', ...)
```

Arguments

- `x`: Raster* object
- `y`: Raster* object if `x` is a RasterLayer; Can be missing if `x` is a RasterStack or RasterBrick
- `digits`: integer. The number of digits for rounding the values before cross-tabulation
- `long`: logical. If TRUE the results are returned in `long` format data.frame instead of a table
- `useNA`: logical, indicating if the table should include counts of NA values
- `progress`: character. "text", "window", or "" (the default, no progress bar), only for large files that cannot be processed in one step
- `...`: additional arguments. None implemented

Value

A table or data.frame

See Also

freq, zonal
**Examples**

```r
r <- raster(nc=5, nr=5)
r[] <- runif(nccell(r)) * 2
s <- setValues(r, runif(nccell(r)) * 3)
crosstab(r, s)

rs <- r/s
r[1:5] <- NA
s[20:25] <- NA
x <- stack(r, s, rs)
crosstab(x, useNA=TRUE, long=TRUE)
```

**Description**
Cut uses the base function `cut` to classify the values of a Raster* object according to which interval they fall in. The intervals are defined by the argument `breaks`. The leftmost interval corresponds to level one, the next leftmost to level two and so on.

**Usage**
```r
cut(x, ...)
```

**Arguments**
- `x`: A Raster* object
- `...`: additional arguments. See `cut`

**Value**
Raster* object

**See Also**
`subs`, `reclassify`, `calc`

**Examples**
```r
r <- raster(ncols=36, nrows=18)
r[] <- rnorm(nccell(r))
breaks <- -2:2 * 3
rc <- cut(r, breaks=breaks)
```
**Coefficient of variation**

**Description**

Compute the coefficient of variation (expressed as a percentage). If there is only a single value, sd is NA and cv returns NA if aszero=FALSE (the default). However, if (aszero=TRUE), cv returns 0.

**Usage**

```r
## S4 method for signature 'ANY'
cv(x, ..., aszero=FALSE, na.rm = FALSE)
```

```r
## S4 method for signature 'Raster'
cv(x, ..., aszero=FALSE, na.rm = FALSE)
```

**Arguments**

- `x` A vector of numbers (typically integers for modal), or a Raster* object
- `...` additional (vectors of) numbers, or Raster objects
- `aszero` logical. If TRUE, a zero is returned (rather than an NA) if the cv of single value is computed
- `na.rm` Remove (ignore) NA values

**Value**

vector or RasterLayer

**Examples**

```r
data <- c(0,1,2,3,3,3,4,4,4,5,5,6,7,7,8,9,NA)
cv(data, na.rm=TRUE)
```

---

**Are values in memory and/or on disk?**

**Description**

These are helper functions for programmers and for debugging that provide information about whether a Raster object has associated values, and if these are in memory or on disk.

`fromDisk` is TRUE if the data source is a file on disk; and FALSE if the object only exists in memory.

`inMemory` is TRUE if all values are currently in memory (RAM); and FALSE if not (in which case they either are on disk, or there are no values).

`hasValues` is TRUE if the object has cell values.
**Usage**

```r
dataType(x)
```

**Arguments**

- `x` Raster* object

**Value**

Logical value

**Examples**

```r
rs <- raster(system.file("external/test.grd", package="raster"))
inMemory(rs)
fromDisk(rs)
rs <- readAll(rs)
inMemory(rs)
fromDisk(rs)
rs <- rs + 1
inMemory(rs)
fromDisk(rs)
rs <- raster(rs)
inMemory(rs)
fromDisk(rs)
rs <- setValues(rs, 1:.ncell(rs))
inMemory(rs)
fromDisk(rs)
rs <- writeRaster(rs, filename='test', overwrite=TRUE)
inMemory(rs)
fromDisk(rs)
```

dataType

<table>
<thead>
<tr>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Description**

Get the datatype of a RasterLayer object. The datatype determines the interpretation of values written to disk. Changing the datatype of a Raster* object does not directly affect the way they are stored in memory. For native file formats (.grd/.gri files) it does affect how values are read from file. This is not the case for file formats that are read via rgdal (such as .tif and .img files) or netcdf.

If you change the datatype of a RasterLayer and then read values from a native format file these may be completely wrong, so only do this for debugging or when the information in the header file was wrong. To set the datatype of a new file, you can give a 'datatype' argument to the functions that write values to disk (e.g. `writeRaster`).
Usage

dataType(x)
dataType(x) <- value

Arguments

x A RasterLayer object
value A data type (see below)

Details

Setting the data type is useful if you want to write values to disk. In other cases use functions such as round()

Datatypes are described by 5 characters. The first three indicate whether the values are integers, decimal number or logical values. The fourth character indicates the number of bytes used to save the values on disk, and the last character indicates whether the numbers are signed (i.e. can be negative and positive values) or not (only zero and positive values allowed)

The following datatypes are available:

<table>
<thead>
<tr>
<th>Datatype definition</th>
<th>minimum possible value</th>
<th>maximum possible value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG1S</td>
<td>FALSE (0)</td>
<td>TRUE (1)</td>
</tr>
<tr>
<td>INT1S</td>
<td>-127</td>
<td>127</td>
</tr>
<tr>
<td>INT1U</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>INT2S</td>
<td>-32,767</td>
<td>32,767</td>
</tr>
<tr>
<td>INT2U</td>
<td>0</td>
<td>65,534</td>
</tr>
<tr>
<td>INT4S</td>
<td>-2,147,483,647</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>INT4U</td>
<td>0</td>
<td>4,294,967,296</td>
</tr>
<tr>
<td>FLT4S</td>
<td>-3.4e+38</td>
<td>3.4e+38</td>
</tr>
<tr>
<td>FLT8S</td>
<td>-1.7e+308</td>
<td>1.7e+308</td>
</tr>
</tbody>
</table>

For all integer types, except the single byte types, the lowest (signed) or highest (unsigned) value is used to store NA. Single byte files do not have NA values. Logical values are stored as signed single byte integers, they do have an NA value (-127)

INT4U is available but they are best avoided as R does not support 32-bit unsigned integers.

Value

Raster* object

Examples

r <- raster(system.file("external/test.grd", package="raster"))
dataType(r)
s <- writeRaster(r, 'new.grd', datatype='INT2U', overwrite=TRUE)
dataType(s)
density

Density plot

Description
Create density plots of values in a Raster object

Usage

```
## S4 method for signature 'Raster'
density(x, layer, maxpixels=100000, plot=TRUE, main, ...)
```

Arguments

- `x` Raster object
- `layer` numeric. Can be used to subset the layers to plot in a multilayer object (RasterBrick or RasterStack)
- `maxpixels` the maximum number of (randomly sampled) cells to be used for creating the plot
- `plot` if TRUE produce a plot, else return a density object
- `main` main title for each plot (can be missing)
- `...` Additional arguments passed to `plot`

Value
density plot (and a density object, returned invisibly if `plot=TRUE`)

Examples

```
logo <- stack(system.file("external/rlogo.grd", package="raster"))
density(logo)
```

dim

Dimensions of a Raster* object

Description
Get or set the number of rows, columns, and layers of a Raster* object. You cannot use this function to set the dimensions of a RasterStack object.

When setting the dimensions, you can provide a row number, or a vector with the row and the column number (for a RasterLayer and a RasterBrick), or a row and column number and the number of layers (only for a RasterBrick)
Usage

```r
## S4 method for signature 'BasicRaster'
dim(x)
```

Arguments

- `x` Raster* object

Value

Integer or Raster* object

See Also

- `ncell`, `extent`, `res`

Examples

```r
r <- raster()
dim(r)
dim(r) <- c(18)
dim(r)
dim(r) <- c(18, 36)
dim(r)
b <- brick(r)
dim(b)
dim(b) <- c(10, 10, 5)
dim(b)
```

<table>
<thead>
<tr>
<th>direction</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

The direction (azimuth) to or from the nearest cell that is not NA. The direction unit is in radians, unless you use argument `degrees=TRUE`.

Usage

```r
## S4 method for signature 'RasterLayer'
direction(x, filename='', degrees=FALSE, from=FALSE, doEdge=FALSE, ...)```
### disaggregate

**Description**

Disaggregate a RasterLayer to create a new RasterLayer with a higher resolution (smaller cells). The values in the new RasterLayer are the same as in the larger original cells unless you specify method="bilinear", in which case values are locally interpolated (using the resample function).

**Usage**

```r
## S4 method for signature 'Raster'
disaggregate(x, fact=NULL, method='', filename='', ...)```

**Arguments**

- `x` RasterLayer object
- `filename` Character. Output filename (optional)
- `degrees` Logical. If FALSE (the default) the unit of direction is radians.
- `from` Logical. Default is FALSE. If TRUE, the direction from (instead of to) the nearest cell that is not NA is returned
- `doEdge` Logical. If TRUE, the boundaries function is called first. This may be efficient in cases where you compute the distance to large blobs. Calling boundaries determines the edge cells that matter for distance computation
- `...` Additional arguments as for `writeRaster`

**Value**

RasterLayer

**See Also**

`distance, gridDistance`

For the direction between (longitude/latitude) points, see the azimuth function in the geosphere package

**Examples**

```r
r <- raster(ncol=36, nrow=18)
r[] <- NA
r[306] <- 1
b <- direction(r)
#plot(b)
```
Arguments

x  
a Raster object

fact  
integer. Amount of disaggregation expressed as number of cells (horizontally and vertically). This can be a single integer or two integers \( c(x, y) \), in which case the first one is the horizontal disaggregation factor and \( y \) the vertical disaggregation factor. If a single integer value is supplied, cells are disaggregated with the same factor in \( x \) and \( y \) direction.

method  
Character. `'` or `'bilinear'`. If `'bilinear'`, values are locally interpolated (using the `resample` function.

filename  
Character. Output filename (optional)

...  
Additional arguments as for `writeRaster`.

Value

Raster object

Author(s)

Robert J. Hijmans and Jim Regetz

See Also

`aggregate`

Examples

```r
r <- raster(ncols=10, nrows=10)
rnd <- disaggregate(r, fact=c(10, 2))
ncol(rnd)
nrow(rnd)
r[1] <- 1:ncell(r)
rnd <- disaggregate(r, fact=c(4, 2), method='bilinear')
```

Description

For a single `RasterLayer` (\( y \) is missing) this method computes the distance, for all cells that are `NA`, to the nearest cell that is not `NA`. The distance unit is in meters if the `RasterLayer` is not projected (`+proj=longlat`) and in map units (typically also meters) when it is projected.

If two `RasterLayer` objects are provided, the cell-value distances are computed. If two `Spatial` vector type objects are provided, the distances between pairs of geographic object are computed.
distanceFromPoints

Usage

```r
## S4 method for signature 'RasterLayer,missing'
distance(x, y, filename='', doEdge=TRUE, ...)
## S4 method for signature 'RasterLayer,RasterLayer'
distance(x, y, ...)
## S4 method for signature 'Spatial,Spatial'
distance(x, y, ...)
```

Arguments

- `x`: RasterLayer object
- `y`: missing, RasterLayer or Spatial object
- `filename`: Character. Filename for the output RasterLayer (optional)
- `doEdge`: Logical. If TRUE, the boundaries function is called first. This may be efficient in cases where you compute the distance to large blobs. Calling boundaries determines the edge cells that matter for distance computation
- `...`: Additional arguments as for `writeRaster`

Value

RasterLayer

See Also

distanceFromPoints, gridDistance, pointDistance

See the gdistance package for more advanced distances, and the geosphere package for great-circle distances (and more) between points in longitude/latitude coordinates.

Examples

```r
r <- raster(ncol=36, nrow=18)
r[] <- NA
r[500] <- 1
dist <- distance(r)
#plot(dist / 1000)
```


distanceFromPoints Distance from points

Description

The function calculates the distance from a set of points to all cells of a Raster* object.
The distance unit is in meters if the coordinate reference system (crs) of the Raster* object is (+proj=longlat) or assumed to be if the crs is NA. In all other cases it is in the units defined by the crs (which typically is meters).
distanceFromPoints

Usage

distanceFromPoints(object, xy, filename='', ...)

Arguments

object Raster object
xy matrix of x and y coordinates, or a SpatialPoints* object.
filename character. Optional filename for the output RasterLayer
... Additional arguments as for writeRaster

Details

Distances for longlat data are computed on the WGS84 spheroid using GeographicLib (Karney, 2013)

Value

RasterLayer

References


See Also

crs, distance, gridDistance, pointDistance

Examples

```
r <- raster(ncol=36,nrow=18)
xy <- c(0,0)
d1 <- distanceFromPoints(r, xy)
crs(r) = '+proj=utm +zone=12 +datum=WGS84'
d2 <- distanceFromPoints(r, xy)
par(mfrow=c(1,2))
plot(d1)
plot(d2)
```
**draw**  

*Draw a line or polygon*

**Description**

Draw a line or polygon on a plot (map) and save it for later use. After calling the function, start clicking on the map. To finish, right-click and select ‘stop’.

**Usage**

```r
drawPoly(sp=TRUE, col='red', lwd=2, ...)
drawLine(sp=TRUE, col='red', lwd=2, ...)
```

**Arguments**

- `sp` logical. If TRUE, the output will be a sp object (SpatialPolygons or SpatialLines). Otherwise a matrix of coordinates is returned.
- `col` the color of the lines to be drawn.
- `lwd` the width of the lines to be drawn.
- `...` additional arguments padded to locator.

**Value**

If `sp==TRUE` a SpatialPolygons or SpatialLines object; otherwise a matrix of coordinates.

**See Also**

- `locator`
  - `drawextent`

---

**drawExtent**  

*Create an Extent object by drawing on a map*

**Description**

Click on two points of a plot (map) to obtain an object of class Extent (‘bounding box’).

**Usage**

```r
drawExtent(show=TRUE, col="red")
```

**Arguments**

- `show` logical. If TRUE, the extent will be drawn on the map.
- `col` sets the color of the lines of the extent.
Erase parts of a SpatialPolygons* or SpatialLines* object. The inverse of this can be done with intersect

Description

Erase parts of a SpatialPolygons* or SpatialLines* object with a SpatialPolygons* object

Usage

## S4 method for signature 'SpatialPolygons,SpatialPolygons'
erase(x, y, ...)

## S4 method for signature 'SpatialLines,SpatialPolygons'
erase(x, y, ...)

Arguments

- **x**: SpatialPolygons or SpatialLines object
- **y**: SpatialPolygons object
- **...**: Additional arguments (none)

Value

Spatial*

Author(s)

Robert J. Hijmans

See Also

The equivalent for raster data is mask
Examples

```r
if (require(rgdal) & require(rgeos)) {
  # erase parts of polygons with other polygons
  p <- shapefile(system.file("external/lux.shp", package="raster"))
  b <- as(extent(6, 6.4, 49.75, 50), 'SpatialPolygons')
  projection(b) <- projection(p)
  e <- erase(p, b)
  plot(e)

  # erase parts of lines with polygons
  r <- raster(extent(p) +c(-.1,.1,-.1,.1), crs=crs(p))
  start <- xyFromCell(r, cellFromCol(r, 1))
  end <- xyFromCell(r, cellFromCol(r, ncol(r)))
  lines <- do.call(splines, lapply(1:10, function(i) rbind(start[i,], end[i,])))
  crs(lines) <- crs(p)
  e2 <- erase(lines, p)
  plot(p)
  lines(lines, col='blue', lwd=4, lty=3)
  lines(e2, col='red', lwd=2)
}
```

Description

Extend returns an Raster* object with a larger spatial extent. The output Raster object has the outer minimum and maximum coordinates of the input Raster and Extent arguments. Thus, all of the cells of the original raster are included. See crop if you (also) want to remove rows or columns. There is also an extend method for Extent objects to enlarge (or reduce) an Extent. You can also use algebraic notation to do that (see examples)

This function has replaced function "expand" (to avoid a name conflict with the Matrix package).

Usage

```r
## S4 method for signature 'Raster'
extend(x, y, value=NA, filename='', ...)  

## S4 method for signature 'Extent'
extend(x, y, ...)  
```

Arguments

- `x`: Raster or Extent object
If \( x \) is a Raster object, \( y \) should be an Extent object, or any object that is or has an Extent object, or an object from which it can be extracted (such as sp objects). Alternatively, you can provide a numeric vector of length 2 indicating the number of rows and columns that need to be added (or a single number when the number of rows and columns is equal).

If \( x \) is an Extent object, \( y \) should be a numeric vector of 1, 2, or 4 elements.

value

value to assign to new cells

filename

Character (optional)

Additional arguments as for writeRaster

Value

RasterLayer or RasterBrick, or Extent

Author(s)

Robert J. Hijmans and Etienne B. Racine (Extent method)

See Also

crop, merge

Examples

```r
r <- raster(xmin=-150, xmax=-120, ymin=60, ymax=30, ncol=36, nrow=18)
r[] <- 1:ncell(r)
e <- extent(-180, 0, 0, 90)
re <- extend(r, e)

# extend with a number of rows and columns (at each side)
re2 <- extend(r, c(2,10))

# Extent object
e <- extent(r)
e
extend(e, 10)
extend(e, 10, -10, 0, 20)
e + 10
e * 2
```

---

**extension**

*Filename extensions*

**Description**

Get or change a filename extension
Usage

extension(filename, value=NULL, maxchar=10)
extension(filename) <- value

Arguments

filename A filename, with or without the path
value A file extension with or without a dot, e.g., ".txt" or "txt"
maxchar Maximum number of characters after the last dot in the filename, for that string to be considered a filename extension

Value

A file extension, filename or path.
If ext(filename) is used without a value argument, it returns the file extension; otherwise it returns the filename (with new extensions set to value)

Examples

fn <- "c:/temp folder/filename.extension"
extension(fn)
extension(fn) <- ".txt"
extension(fn)
fn <- extension(fn, ".document")
extension(fn)
extension(fn, maxchar=4)

Description

This function returns an Extent object of a Raster* or Spatial* object (or an Extent object), or creates an Extent object from a 2x2 matrix (first row: xmin, xmax; second row: ymin, ymax), vector (length=4; order= xmin, xmax, ymin, ymax) or list (with at least two elements, with names 'x' and 'y')
bbox returns a sp package like 'bbox' object (a matrix)

Usage

extent(x, ...)

extent
Arguments

- **x**: Raster* or Extent object, a matrix, or a vector of four numbers
- **...**: Additional arguments. When x is a single number representing 'xmin’, you can pass three additional numbers (xmax, ymin, ymax)
  When x is a Raster* object, you can pass four additional arguments to crop the extent: r1, r2, c1, c2, representing the first and last row and column number.

Value

- Extent object

Author(s)

- Robert J. Hijmans; Etienne Racine wrote the extent function for a list

See Also

- `extent`, `drawExtent`

Examples

```r
r <- raster()
extent(r)
extent(c(0, 20, 0, 20))
# is equivalent to
extent(0, 20, 0, 20)
extent(matrix(c(0, 0, 20, 20), nrow=2))
x <- list(x=c(0,1,2), y=c(-3,5))
extent(x)

# crop the extent by row and column numbers
extent(r, 1, 20, 10, 30)
```

Description

- Use `round(x, digits=0)` to round the coordinates of an Extent object to the number of digits specified. This can be useful when dealing with a small imprecision in the data (e.g. 179.9999 instead of 180). floor and ceiling move the coordinates to the outer or inner whole integer numbers.
- It is also possible to use Arithmetic functions with Extent objects (but these work perhaps unexpectedly!)
- See `Math-methods` for these (and many more) methods with Raster* objects.
Usage

```r
## S4 method for signature 'Extent'
floor(x)
## S4 method for signature 'Extent'
ceiling(x)
```

Arguments

- `x` : Extent object

See Also

- `Math-methods`

Examples

```r
e <- extent(c(0.999999, 10.000011, -60.4, 60))
round(e)
ceiling(e)
floor(e)
```

Description

Objects of class `Extent` are used to define the spatial extent (extremes) of objects of the BasicRaster and Raster* classes.

Objects from the Class

You can use the `extent` function to create Extent objects, or to extract them from Raster* and Spatial* objects.

Slots

- `xmin` : minimum x coordinate
- `xmax` : maximum x coordinate
- `ymin` : minimum y coordinate
- `ymax` : maximum y coordinate

Methods

- `show` : display values of a Extent object

See Also

- `extent`, `setExtent`
Examples

```r
ext <- extent(-180, 180, -90, 90)
ext
```

## Descriptions

Extract values from a Raster* object at the locations of other spatial data. You can use coordinates (points), lines, polygons or an Extent (rectangle) object. You can also use cell numbers to extract values.

If `y` represents points, `extract` returns the values of a Raster* object for the cells in which a set of points fall. If `y` represents lines, the `extract` method returns the values of the cells of a Raster* object that are touched by a line. If `y` represents polygons, the `extract` method returns the values of the cells of a Raster* object that are covered by a polygon. A cell is covered if its center is inside the polygon (but see the `weights` option for considering partly covered cells; and argument `small` for getting values for small polygons anyway).

It is also possible to extract values for point locations from SpatialPolygons.

### Usage

```r
## S4 method for signature 'Raster, matrix'
extract(x, y, method='simple', buffer=NULL, small=FALSE, cellnumbers=FALSE, fun=NULL, na.rm=TRUE, layer, nl, df=FALSE, factors=FALSE, ...)

## S4 method for signature 'Raster, SpatialLines'
extract(x, y, fun=NULL, na.rm=FALSE, cellnumbers=FALSE, df=FALSE, layer, nl, factors=FALSE, along=FALSE, sp=FALSE, ...)

## S4 method for signature 'Raster, SpatialPolygons'
extract(x, y, fun=NULL, na.rm=FALSE, weights=FALSE, normalizeWeights=TRUE, cellnumbers=FALSE, small=TRUE, df=FALSE, layer, nl, factors=FALSE, sp=FALSE, ...)

## S4 method for signature 'SpatialPolygons, SpatialPoints'
extract(x, y, ...)
```

### Arguments

- `x`: Raster* object
- `y`: points represented by a two-column matrix or data.frame, or SpatialPoints*: SpatialPolygons*: SpatialLines: Extent; or a numeric vector representing cell numbers
method character. 'simple' or 'bilinear'. If 'simple' values for the cell a point falls in are returned. If 'bilinear' the returned values are interpolated from the values of the four nearest raster cells.

buffer numeric. The radius of a buffer around each point from which to extract cell values. If the distance between the sampling point and the center of a cell is less than or equal to the buffer, the cell is included. The buffer can be specified as a single value, or as a vector of the length of the number of points. If the data are not projected (latitude/longitude), the unit should be meters. Otherwise it should be in map-units (typically also meters).

small logical. If TRUE and y represents points and a buffer argument is used, the function always return a number, also when the buffer does not include the center of a single cell. The value of the cell in which the point falls is returned if no cell center is within the buffer. If y represents polygons, a value is also returned for relatively small polygons (e.g. those smaller than a single cell of the Raster* object), or polygons with an odd shape, for which otherwise no values are returned because they do not cover any raster cell centers. In some cases, you could alternatively use the centroids of such polygons, for example using extract(x, coordinates(y)) or extract(x, coordinates(y), method='bilinear').

fun function to summarize the values (e.g. mean). The function should take a single numeric vector as argument and return a single value (e.g. mean, min or max), and accept a na.rm argument. Thus, standard R functions not including an na.rm argument must be wrapped as in this example: fun=function(x,...)length(x). If y represents points, fun is only used when a buffer is used (and hence multiple values per spatial feature would otherwise be returned).

na.rm logical. Only useful when an argument fun is supplied. If na.rm=TRUE (the default value), NA values are removed before fun is applied. This argument may be ignored if the function used has a ... argument and ignores an additional na.rm argument.

cellnumbers logical. If cellnumbers=TRUE, cell-numbers will also be returned (if no fun argument is supplied, and when extracting values with points, if buffer is NULL).

df logical. If df=TRUE, results will be returned as a data.frame. The first column is a sequential ID, the other column(s) are the extracted values.

weights logical. If TRUE and normalizeWeights=FALSE, the function returns, for each polygon, a matrix with the cell values and the approximate fraction of each cell that is covered by the polygon (rounded to 1/100). If TRUE and normalizeWeights=TRUE the weights are normalized such that they add up to one. The weights can be used for averaging; see examples. This option can be useful (but slow) if the polygons are small relative to the cells size of the Raster* object.

normalizeWeights logical. If TRUE, weights are normalized such that they add up to one for each polygon.

factors logical. If TRUE, factor values are returned, else their integer representation is returned.

layer integer. First layer for which you want values (if x is a multilayer object).

nl integer. Number of layers for which you want values (if x is a multilayer object).
along boolean. Should returned values be ordered to go along the lines?

sp boolean. Should the extracted values be added to the data.frame of the Spatial* object y? This only applies if y is a Spatial* object and, for SpatialLines and SpatialPolygons, if fun is not NULL. In this case the returned value is the expanded Spatial object

... additional arguments (none implemented)

Value

A vector for RasterLayer objects, and a matrix for RasterStack or RasterBrick objects. A list (or a data.frame if df=TRUE) if y is a SpatialPolygons* or SpatialLines* object or if a buffer argument is used (but not a fun argument). If sp=TRUE and y is a Spatial* object and fun is not NULL a Spatial* object is returned. The order of the returned values corresponds to the order of object y. If df=TRUE, this is also indicated in the first variable ('ID').

See Also

getvaluesL, getValuesFocal

Examples

r <- raster(ncmp=36, nrow=18, vals=1:(18*36))

# extract values by cell number
# extract values with points
# examples with a buffer
# illustrating the varying size of a buffer (expressed in meters)
# on a longitude/latitude raster
# compare with raster that is not longitude/latitude

sp <- SpatialPoints(xy)
extract(r, sp, method='bilinear')

z <- extract(r, xy, buffer=1000000)
s <- raster(r)
for (i in 1:length(z)) { s[z[i]] <- i }

projection(r) <- "+proj=utm +zone=17"
```r
xy[,1] <- 50
z <- extract(r, xy, buffer=8)
for (i in 1:length(z)) { s[z[[i]]] <- i }
plot(s)
# library(maptools)
# data(wrld_simpl)
# plot(wrld_simpl, add=TRUE)

# extract values with lines
r <- raster(ncol=36, nrow=18, vals=1:(18*36))
cds1 <- rbind(c(-50, 0), c(0, 60), c(40, 5), c(15, -45), c(-10, -25))
cds2 <- rbind(c(80, 20), c(140, 60), c(160, 0), c(140, -55))
lines <- splines(cds1, cds2)

extract(r, lines)

# extract values with polygons
cds1 <- rbind(c(-100, -20), c(-160, 5), c(-60, 0), c(-160, -60), c(-180, -20))
cds2 <- rbind(c(80, 0), c(100, 60), c(120, 0), c(120, -55), c(80, 0))
polys <- spPolygons(cds1, cds2)

v <- extract(r, polys)
# mean for each polygon
unlist(lapply(v, function(x) if (!is.null(x)) mean(x, na.rm=TRUE) else NA ))

# v <- extract(r, polys, cellnumbers=TRUE)

# weighted mean
# v <- extract(r, polys, weights=TRUE, fun=mean)
# equivalent to:
# v <- extract(r, polys, weights=TRUE)
# sapply(v, function(x) if (!is.null(x)) {sum(apply(x, 1, prod)) / sum(x[,2])} else NA)

# extract values with an extent
e <- extent(150, 170, -60, -40)
extract(r, e)
#plot(e)
#plot(e, add=T)
```

**Extract by index**  
*Indexing to extract values of a Raster* object
Description

These are shorthand methods that call other methods that should normally be used, such as `getValues`, `extract`, `crop`.

`object[i]` can be used to access values of a Raster* object, using cell numbers. You can also use row and column numbers as index, using `object[i,j]` or `object[i,]` or `object[,]`. In addition you can supply an Extent, SpatialPolygons, SpatialLines or SpatialPoints object.

If `drop=TRUE` (the default) cell values are returned (a vector for a RasterLayer, a matrix for a RasterStack or RasterBrick). If `drop=FALSE` a Raster* object is returned that has the extent covering the requested cells, and with all other non-requested cells within this extent set to `NA`.

If you supply a RasterLayer, its values will be used as logical (TRUE/FALSE) indices if both Raster objects have the same extent and resolution; otherwise the cell values within the extent of the RasterLayer are returned.

Double brackes `[[ ]]` can be used to extract one or more layers from a multi-layer object.

Methods

- `x[i]`
- `x[i,j]`

Arguments

- `x` a Raster* object
- `i` cell number(s), row number(s), a (logical) RasterLayer, Spatial* object
- `j` column number(s) (only available if `i` is (are) a row number(s))
- `drop` If TRUE, cell values are returned. Otherwise, a Raster* object is returned

See Also

`getValues`, `setValues`, `extract`, `crop`, `rasterize`

Examples

```r
r <- raster(ncol=10, nrow=5)
r[] <- 1:ncell(r)

r[1]
r[1:10]
r[1,]
r[,] r[1:2, 1:2]

s <- stack(r, sqrt(r))
s[1:3]
s[[2]]
```
Extreme coordinates

Coordinates of the Extent of a Raster object

Description

These functions return or set the extreme coordinates of a Raster* object; and return them for Spatial* objects.

Usage

\[
\begin{align*}
x_{\text{min}}(x) \\
x_{\text{max}}(x) \\
y_{\text{min}}(x) \\
y_{\text{max}}(x)
\end{align*}
\]

\[
\begin{align*}
x_{\text{min}}(x) & \leftarrow \text{value} \\
x_{\text{max}}(x) & \leftarrow \text{value} \\
y_{\text{min}}(x) & \leftarrow \text{value} \\
y_{\text{max}}(x) & \leftarrow \text{value}
\end{align*}
\]

Arguments

- \text{x} \quad \text{A Raster* object}
- \text{value} \quad \text{A new x or y coordinate}

Value

a single number

See Also

extent, dimensions

Examples

\[
\begin{align*}
r & \leftarrow \text{raster}(\text{vmin}=-0.5, \text{vmax}=9.5, \text{nrows}=10) \\
x_{\text{min}}(r) \\
x_{\text{max}}(r) \\
y_{\text{min}}(r) \\
y_{\text{max}}(r) \\
x_{\text{min}}(r) & \leftarrow -180 \\
x_{\text{max}}(r) & \leftarrow 180
\end{align*}
\]
extremeValues  Minimum and maximum values

Description

Returns the minimum or maximum value of a RasterLayer or layer in a RasterStack

Usage

minValue(x, ...)
maxValue(x, ...)

Arguments

x RasterLayer or RasterStack object
... Additional argument: layer number (for RasterStack or RasterBrick objects)

Details

If a Raster* object is created from a file on disk, the min and max values are often not known (depending on the file format). You can use setMinMax to set them in the Raster* object.

Value

a number

Examples

r <- raster()
r <- setValues(r, 1:ncell(r))
minValue(r)
maxValue(r)
r <- setValues(r, round(100 * runif(ncell(r)) + 0.5))
minValue(r)
maxValue(r)
r <- raster(system.file("external/test.grd", package="raster"))
minValue(r)
maxValue(r)
Factors

Description

These functions allow for defining a RasterLayer as a categorical variable. Such a RasterLayer is linked to other values via a "Raster Attribute Table" (RAT). Thus the cell values are an index, whereas the actual values of interest are in the RAT. The RAT is a data.frame. The first column in the RAT ("ID") has the unique cell values of the layer; this column should normally not be changed. The other columns can be of any basic type (factor, character, integer, numeric or logical). The functions documented here are mainly available such that files with a RAT can be read and processed; currently there is not too much further support. Whether a layer is defined as a factor or not is currently ignored by almost all functions. An exception is the 'extract' function (when used with option df=TRUE).

Function 'levels' returns the RAT for inspection. It can be modified and set using levels <= value (but use caution as it is easy to mess things up).

as.factor and ratify create a layer with a RAT table. Function 'deratify' creates a single layer for a (or each) variable in the RAT table.

Usage

is.factor(x)
as.factor(x)
levels(x)

factorValues(x, v, layer=1, att=NULL, append.names=FALSE)

ratify(x, filename='', count=FALSE, ...)
deratify(x, att=NULL, layer=1, complete=FALSE, drop=TRUE, fun='mean', filename='', ...)
asFactor(x, ...)

Arguments

x            Raster* object
v            integer cell values
layer        integer > 0 indicating which layer to use (in a RasterStack or RasterBrick)
att          numeric or character. Which variable(s) in the RAT table should be used. If NULL, all variables are extracted. If using a numeric, skip the first two default columns
append.names logical. Should names of data.frame returned by a combination of the name of the layer and the RAT variables? (can be useful for multilayer objects
filename     character. Optional
count        logical. If TRUE, a columns with frequencies is added
... additional arguments as for `writeRaster`

- `complete` logical. If TRUE, the layer returned is no longer a factor
- `drop` logical. If TRUE a factor is converted to a numerical value if possible
- `fun` character. Used to get a single value for each class for a weighted RAT table. 'mean', 'min', 'max', 'smallest', or 'largest'

### Value

- Raster* object; list (levels); boolean (is.factor); matrix (factorValues)

### Note

asFactor is deprecated and should not be used

### Examples

```r
set.seed(0)
r <- raster(nrow=10, ncol=10)
r[] <- runif(ncell(r)) * 10
is.factor(r)

r <- round(r)
f <- as.factor(r)
is.factor(f)

x <- levels(f)[[1]]
x
x$code <- letters[10:20]
levels(f) <- x
levels(f)
f

r <- raster(nrow=10, ncol=10)
r[] = 1
r[51:100] = 2
r[3:6, 1:5] = 3
r <- ratify(r)

rat <- levels(r)[[1]]
rat$landcover <- c('Pine', 'Oak', 'Meadow')
rat$code <- c(12,25,30)
levels(r) <- rat
r

# extract values for some cells
i <- extract(r, c(1,2, 25,100))
i
# get the attribute values for these cells
factorValues(r, i)

# write to file:
```
```r
rr <- writeRaster(r, 'test.grd', overwrite=TRUE)
rr

# create a single-layer factor
x <- deratify(r, 'landcover')
x
is.factor(x)
levels(x)
```

### filename

<table>
<thead>
<tr>
<th>Filename</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Description**

Get the filename of a Raster* object. You cannot set the filename of an object (except for Raster-Stack objects); but you can provide a 'filename= ' argument to a function that creates a new Raster-Layer or RasterBrick* object.

**Usage**

`filename(x)`

**Arguments**

- `x`: A Raster* object

**Value**

- a Raster* object

**Examples**

```r
r <- raster( system.file("external/test.grd", package="raster") )
filename(r)
```

### filledContour

**Filled contour plot**

<table>
<thead>
<tr>
<th>Filled contour plot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Description**

Filled contour plot of a RasterLayer. This is a wrapper around `filled.contour` for RasterLayer objects.

**Usage**

`filledContour(x, y=1, maxpixels=100000, ...)`
Arguments

x A Raster* object
y Integer. The layer number of x (if x has multiple layers)
maxpixels The maximum number of pixels
... Any argument that can be passed to filled.contour (graphics package)

See Also

filled.contour, persp, plot

Examples

r <- raster(system.file("external/test.grd", package="raster"))
filledContour(r)

flip

Description

Flip the values of a Raster* object by inverting the order of the rows (direction=y) or the columns direction='x'.

Usage

flip(x, direction, ...)

Arguments

x Raster* object
direction Character. 'y' or 'x'; or 1 (=x) or 2 (=y)
... Additional arguments as for writeRaster

Value

RasterLayer or RasterBrick

See Also

t, rotate

Examples

r <- raster(nrow=18, ncol=36)
m <- matrix(1:ncell(r), nrow=18)
r[] <- as.vector(t(m))
rx <- flip(r, direction='x')
r[] <- as.vector(m)
ry <- flip(r, direction='y')
Description

Compute the flow path (drainage path) starting at a given point. See package gdistance for more path computations.

Usage

flowPath(x, p, ...)

Arguments

x  
RasterLayer of flow direction (as can be created with terrain)

p  
starting point. Either two numbers: x (longitude) and y (latitude) coordinates; or a single cell number

...  
additional arguments (none implemented)

Value

numeric (cell numbers)

Author(s)

Ashton Shortridge

Examples

data(volcano)
v <- raster(volcano, xmn=2667400, xmx=2668010, ymn=6478700, ymx=6479570, crs="+init=epsg:27200")
fd <- terrain(v, opt = "flowdir")

path <- flowPath(fd, 2407)
xy <- xyFromCell(fd, path)
plot(v)
lines(xy)
Description

Calculate focal ("moving window") values for the neighborhood of focal cells using a matrix of weights, perhaps in combination with a function.

Usage

```r
## S4 method for signature 'RasterLayer'
focal(x, w, fun, filename='', na.rm=FALSE, pad=FALSE, padValue=NA, NAonly=FALSE, ...)
```

Arguments

- `x` RasterLayer
- `w` matrix of weights (the moving window), e.g. a 3 by 3 matrix with values 1; see Details. The matrix does not need to be square, but the sides must be odd numbers. If you need even sides, you can add a column or row with weights of zero
- `fun` function (optional). The function fun should take multiple numbers, and return a single number. For example mean, modal, min or max. It should also accept a na.rm argument (or ignore it, e.g. as one of the 'dots' arguments. For example, length will fail, but `function(x, ...)(na.omit(length(x)))` works.
- `filename` character. Filename for a new raster (optional)
- `na.rm` logical. If TRUE, NA will be removed from focal computations. The result will only be NA if all focal cells are NA. Except for some special cases (weights of 1, functions like min, max, mean), using `na.rm=TRUE` is generally not a good idea in this function because it will unbalance the effect of the weights
- `pad` logical. If TRUE, additional 'virtual' rows and columns are padded to `x` such that there are no edge effects. This can be useful when a function needs to have access to the central cell of the filter
- `padValue` numeric. The value of the cells of the padded rows and columns
- `NAonly` logical. If TRUE, only cell values that are NA are replaced with the computed focal values
- `...` Additional arguments as for `writeRaster`

Details

`focal` uses a matrix of weights for the neighborhood of the focal cells. The default function is `sum`. It is computationally much more efficient to adjust the weights-matrix than to use another function through the `fun` argument. Thus while the following two statements are equivalent (if there are no NA values), the first one is faster than the second one:

```r
a <- focal(x, w=matrix(1/9, nc=3, nr=3))
```
There is, however, a difference if NA values are considered. One can use the na.rm=TRUE option which may make sense when using a function like mean. However, the results would be wrong when using a weights matrix. 

Laplacian filter: `filter=matrix(c(0,1,0,1,-4,1,0,1,0), nrow=3)`

Sobel filter: `filter=matrix(c(1,2,1,0,0,0,-1,-2,-1) / 4, nrow=3)`

see the `focalWeight` function to create distance based circular, rectangular, or Gaussian filters.

**Value**

RasterLayer

**See Also**

`focalWeight`

**Examples**

```r
b <- focal(x, w=matrix(1,3,3), fun=mean)
```

```
# focal can be used to create a cellular automaton
```

```r
# Conway's Game of Life
e <- vector()
gameOfLife <- function(x) {
f <- focal(x, w=e, pad=TRUE, padValue=0)
# cells with less than two or more than three live neighbours die
x[f<2 | f>3] <- 0
# cells with three live neighbours become alive
focalWeight

Description

Calculate focal ("moving window") weight matrix for use in the focal function. The sum of the values adds up to one.

Usage

focalWeight(x, d, type=c('circle', 'Gauss', 'rectangle'))

Arguments

x Raster* object
d numeric. If type=circle, the radius of the circle (in units of the CRS). If type=rectangle the dimension of the rectangle (one or two numbers). If type=Gauss the size of sigma, and optionally another number to determine the size of the matrix returned (default is 3 times sigma)
type character indicating the type of filter to be returned
Value

matrix that can be used in `focal`

Examples

```r
r <- raster(ncols=36, nrows=18, xmn=0)
# Gaussian filter for square cells
gf <- focalWeight(r, 2, "Gauss")
```

freq  
Frequency table

Description

Frequency table of the values of a RasterLayer.

Usage

```r
## S4 method for signature 'RasterLayer'
freq(x, digits=0, value=NULL, useNA='ifany', progress='', ...)

## S4 method for signature 'RasterStackBrick'
freq(x, digits=0, value=NULL, useNA='ifany', merge=FALSE, progress='', ...)
```

Arguments

- `x`  
  RasterLayer

- `digits`  
  non-negative integer for rounding the cell values. Argument is passed to `round`

- `value`  
  numeric, logical or NA. An optional single value to only count the number of cells with that value

- `useNA`  
  character. What to do with NA values? Options are "no", "ifany", "always". See to `table`

- `progress`  
  character to specify a progress bar. Choose from 'text', 'window', or " (the default, no progress bar)

- `merge`  
  logical. If TRUE the list will be merged into a single data.frame

- `...`  
  additional arguments (none implemented)

Value

matrix (RasterLayer). List of matrices (one for each layer) or data.frame (if merge=TRUE) (RasterStack or RasterBrick)

See Also

crosstab and zonal
Gain and offset

Examples

```r
r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r))
r[1:5] <- NA
r <- r * r * r * 5
d <- freq(r)
d <- freq(r, value=2)
s <- stack(r, r*2, r*3)
d <- freq(s, merge=TRUE)
```

Description

These functions can be used to get or set the gain and offset parameters used to transform values when reading them from a file. The gain and offset parameters are applied to the raw values using the formula below:

\[ \text{value} \leftarrow \text{value} \times \text{gain} + \text{offset} \]

The default value for gain is 1 and for offset is 0. ‘gain’ is sometimes referred to as ‘scale’.

Note that setting gain and/or offset are intended to be used with values that are stored in a file. For a Raster* object with values in memory, assigning gain or offset values will lead to the immediate computation of new values; in such cases it would be clearer to use Arith-methods.

Usage

```r
gain(x)
gain(x) <- value
offs(x)
offs(x) <- value
```

Arguments

- `x` Raster* object
- `value` Single numeric value

Value

Raster* object or numeric value(s)
Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))
gain(r)
offs(r)
r[1505:1510]
gain(r) <- 10
offs(r) <- 5
r[1505:1510]
```

Description

Extract the coordinates of a Spatial* object

Usage

```r
## S4 method for signature 'SpatialPolygons'
geom(x, sepNA=FALSE, ...)
```

Arguments

- `x` SpatialPolygons*, SpatialLines*, or SpatialPoints* object
- `sepNA` logical. If TRUE, geometries are separated by a row with NA values
- `...` additional arguments, none implemented

Value

Matrix with 6, (5 SpatialLines), or 3 (SpatialPoints) columns. object (sequential object number) part (sequential part number within the object; not for SpatialPoints), cump (cumulative part number; not for SpatialPoints), hole (is this a hole or not; only for SpatialPolygons), x (x coordinate or longitude), y (y coordinate or latitude)

See Also

coordinates, geometry

Examples

```r
if (require(rgdal) & require(rgeos)) {
p <- shapefile(system.file("external/lux.shp", package="raster"))
x <- geom(p)
}
```
### Description

Get geographic data for anywhere in the world. Data are read from files that are first downloaded if necessary. Function ccodes returns country names and the ISO codes.

### Usage

```
getData(name, download=TRUE, path=' ', ...)  
ccodes()
```

### Arguments

- `name`: Data set name, currently supported are 'GADM', 'countries', 'SRTM', 'alt', and 'worldclim'. See Details for more info.
- `download`: Logical. If TRUE data will be downloaded if not locally available.
- `path`: Character. Path name indicating where to store the data. Default is the current working directory.
- `...`: Additional required (!) parameters. These are data set specific. See Details.

### Details

- **'alt'** stands for altitude (elevation); the data were aggregated from SRTM 90 m resolution data between -60 and 60 latitude. 'GADM' is a database of global administrative boundaries. 'worldclim' is a database of global interpolated climate data. 'SRTM' refers to the hole-filled CGIAR-SRTM (90 m resolution). 'countries' has polygons for all countries at a higher resolution than the 'wrld_simpl' data in the maptools package.

If `name` is 'alt' or 'GADM' you must provide a 'country=' argument. Countries are specified by their 3 letter ISO codes. Use `getData('ISO3')` to see these codes. In the case of GADM you must also provide the level of administrative subdivision (0=country, 1=first level subdivision). In the case of alt you can set 'mask' to FALSE. If it is TRUE values for neighbouring countries are set to NA. For example:

```
getData('GADM', country='FRA', level=1)  
getData('alt', country='FRA', mask=TRUE)
```

If `name` is 'SRTM' you must provide 'lon' and 'lat' arguments (longitude and latitude). These should be single numbers somewhere within the SRTM tile that you want.

```
getData('SRTM', lon=5, lat=45)
```

If `name`='worldclim' you must also provide arguments `var` and a resolution `res`. Valid variables names are 'tmin', 'tmax', 'prec' and 'bio'. Valid resolutions are 0.5, 2.5, 5, and 10 (minutes of a degree). In the case of `res=0.5`, you must also provide a `lon` and `lat` argument for a tile; for the lower resolutions global data will be downloaded. In all cases there are 12 (monthly) files for each variable except for 'bio' which contains 19 files.
getValues

getData('worldclim', var='tmin', res=0.5, lon=5, lat=45)
getData('worldclim', var='bio', res=10)

To get (projected) future climate data (CMIP5), you must provide arguments `var` and `res` as above. Only resolutions 2.5, 5, and 10 are currently available. In addition, you need to provide `model`, `rcp` and `year`. For example,

getData('CMIP5', var='tmin', res=10, rcp=85, model='AC', year=70)

function (var, model, rcp, year, res, lon, lat, path, download = TRUE)

'`model`' should be one of "AC", "BC", "CC", "CE", "CN", "GF", "GD", "GS", "HD", "HG", "HE", "IN", "IP", "MI", "MR", "MC", "MP", "MG", or "NO".

'`rcp`' should be one of 26, 45, 60, or 85.

'`year`' should be 50 or 70

Not all combinations are available. See www.worldclim.org for details.

Value

A spatial object (Raster* or Spatial*)

References

http://www.worldclim.org
https://gadm.org
http://srtm.csi.cgiar.org/
http://diva-gis.org/gdata

getValues

Get raster cell values

Description

getValues returns all values or the values for a number of rows of a Raster* object. Values returned for a RasterLayer are a vector. The values returned for a RasterStack or RasterBrick are always a matrix, with the rows representing cells, and the columns representing layers

values is a shorthand version of getValues (for all rows).

Usage

getValues(x, row, nrow, ...)

values(x, ...)
getValuesBlock

Arguments

- **x**  
  Raster* object

- **row**  
  Numeric. Row number, should be between 1 and nrow(x), or missing in which case all values are returned

- **nrows**  
  Numeric. Number of rows. Should be an integer > 0, or missing

- **...**  
  Additional arguments. When x is a RasterLayer: format to specify the output format. Either "matrix" or, the default "", in which case a vector is returned

Value

vector or matrix of raster values

See Also

getValuesBlock, getValuesFocal, setValues

Examples

```r
  r <- raster(system.file("external/test.grd", package="raster"))
  r
  v <- getValues(r)
  length(v)
  head(v)
  getValues(r, row=10)
```

---

getValuesBlock  
*Get a block of raster cell values*

Description

getValuesBlock returns values for a block (rectangular area) of values of a Raster* object.

Usage

```r
  ## S4 method for signature 'RasterLayer'
  getValuesBlock(x, row=1, nrows=1, col=1, ncols=(ncol(x)-col+1), format='', ...)

  ## S4 method for signature 'RasterBrick'
  getValuesBlock(x, row=1, nrows=1, col=1, ncols=(ncol(x)-col+1), lyrs, ...)

  ## S4 method for signature 'RasterStack'
  getValuesBlock(x, row=1, nrows=1, col=1, ncols=(ncol(x)-col+1), lyrs, ...)
```
getValuesFocal

Arguments

- `x` Raster* object
- `row` positive integer. Row number to start from, should be between 1 and nrow(x)
- `nrows` positive integer. How many rows? Default is 1
- `col` positive integer. Column number to start from, should be between 1 and ncol(x)
- `ncols` positive integer. How many columns? Default is the number of columns left after the start column
- `format` character. When `x` is a RasterLayer, if `format='matrix'` or `format='m'`, a matrix is returned instead of a vector. If `format='matrix'`, it is a nrow x ncol matrix. If `format='m'` it is a 1 column matrix (the benefit is that the type of output is now the same for all Raster objects)
- `lyrs` integer (vector). Which layers? Default is all layers (1:nlayers(x))
- `...` additional arguments (none implemented)

Value

matrix or vector (if `x`=RasterLayer, unless `format='matrix'`)

See Also

getValues

Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))
b <- getValuesBlock(r, row=100, nrows=3, col=10, ncols=5)
b b <- matrix(b, nrow=3, ncol=5, byrow=TRUE)
b

logo <- brick(system.file("external/logo.grd", package="raster"))
getValuesBlock(logo, row=35, nrows=3, col=50, ncol=3, lyrs=2:3)
```

getValuesFocal

Get focal raster cell values

Description

This function returns a matrix (or matrices) for all focal values of a number of rows of a Raster* object

Usage

```r
## S4 method for signature 'Raster'
getValuesFocal(x, row, nrows, ngb, names=FALSE, padValue=NA, array=FALSE, ...)
```
gridDistance

Arguments

- **x**: Raster* object
- **row**: Numeric. Row number, should be between 1 and nrow(x). Can be omitted to get all rows
- **nrows**: Numeric. Number of rows, should be a positive integer smaller than row+nrow(x). Should be omitted if row is omitted
- **ngb**: Neighbourhood size. Either a single integer or a vector of two integers c(nrow, ncol)
- **names**: logical. If TRUE, the matrix returned has row and column names
- **padValue**: numeric. The value of the cells of the "padded" rows and columns. That is 'virtual' values for cells within a neighbourhood, but outside the raster
- **array**: logical. If TRUE and x has multiple layers, an array is returned instead of a list of matrices
- ... additional arguments (none implemented)

Value

If x has a single layer, a matrix with one row for each focal cell, and one column for each neighbourhood cell around it.

If x has multiple layers, an array (if array=TRUE) or a list of such matrices (one list element (matrix) for each layer)

See Also

g Values, focal

Examples

```r
r <- raster(nr=5, nc=5, crs='+proj=utm +zone=12')
r[] <- 1:25
as.matrix(r)
getValuesFocal(r, row=1, nrows=2, ngb=3, names=TRUE)
getValuesFocal(stack(r,r), row=1, nrows=1, ngb=3, names=TRUE, array=TRUE)
```

Description

The function calculates the distance to cells of a RasterLayer when the path has to go through the centers of neighboring raster cells (currently only implemented as a 'queen' case in which cells have 8 neighbors).

The distance is in meters if the coordinate reference system (CRS) of the RasterLayer is longitude/latitude (+proj=longlat) and in the units of the CRS (typically meters) in other cases.

Distances are computed by summing local distances between cells, which are connected with their neighbours in 8 directions.
Usage

```r
## S4 method for signature 'RasterLayer'
gridDistance(x, origin, omit=NULL, filename="", ...)```

Arguments

- `x`: RasterLayer
- `origin`: value(s) of the cells from which the distance is calculated
- `omit`: value(s) of the cells which cannot be traversed (optional)
- `filename`: character. output filename (optional)
- `...`: additional arguments as for `writeRaster`

Details

If the RasterLayer to be processed is big, it will be processed in chunks. This may lead to errors in the case of complex objects spread over different chunks (meandering rivers, for instance). You can try to solve these issues by varying the chunk size, see function `setOptions()`.

Value

- `RasterLayer`

Author(s)

- Jacob van Etten and Robert J. Hijmans

See Also

- See `distance` for 'as the crow flies' distance. Additional distance measures and options (directions, cost-distance) are available in the 'gdistance' package.

Examples

```r
#world lon/lat raster
r <- raster(ncol=10,nrow=10)
r[] <- 1
r[48] <- 2
r[66:68] <- 3
d <- gridDistance(r,origin=2,omit=3)
plot(d)

#UTM small area
projection(r) <- "+proj=utm +zone=15 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
d <- gridDistance(r,origin=2,omit=3)
plot(d)```
hdr  

Header files

Description

Write header files to use together with raster binary files to read the data in other applications.

Usage

hdr(x, format, extension='wld', filename='')

Arguments

x RasterLayer or RasterBrick object associated with a binary values file on disk
format Type of header file: 'VRT', 'BIL', 'ENVI', 'ErdasRaw', 'IDRISI', 'SAGA', 'RASTER', 'WORLDFILE', 'PRJ'
extension File extension, only used with an ESRI worldfile (format='WORLDFILE')
filename character. Need to be provided if x is not associated with a file

Details

The RasterLayer object must be associated with a file on disk.

You can use writeRaster to save a existing file in another format. But if you have a file in a 'raster' format (or similar), you can also only export a header file, and use the data file (.gri) that already exists. The function can write a VRT (GDAL virtual raster) header (.vrt); an ENVI or BIL header (.hdr) file; an Erdas Raw (.raw) header file; an IDRISI (.rdc) or SAGA (.sgrd). This (hopefully) allows for reading the binary data (.gri), perhaps after changing the file extension, in other programs such as ENVI or ArcGIS.

See Also

writeRaster, writeGDAL

Examples

r <- raster(system.file("external/test.grd", package="raster"))
r <- writeRaster(r, filename="export.grd", overwrite=TRUE)
hdr(r, format="ENVI")
head

Show the head (first rows/columns) or tail (last rows/columns) of the cell values of a Raster* object.

Usage

head(x, ...)
tail(x, ...)

Arguments

x Raster* object

... Additional arguments: rows=10 and cols=20, to set the maximum number of rows and columns that are shown. For RasterStack and RasterBrick objects there is an additional argument lyr

Value

matrix

See Also

getValuesBlock

Examples

r <- raster(nrow=25, ncol=25)
r[] = 1:ncell(r)
head(r)
tail(r, cols=10, rows=5)

hillShade

Hill shading

Description

Compute hill shade from slope and aspect layers (both in radians). Slope and aspect can be computed with function terrain.

A hill shade layer is often used as a backdrop on top of which another, semi-transparent, layer is drawn.
Usage

hillShade(slope, aspect, angle=45, direction=0, filename='', normalize=FALSE, ...)

Arguments

- **slope**: RasterLayer object with slope values (in radians)
- **aspect**: RasterLayer object with aspect values (in radians)
- **angle**: The elevation angle of the light source (sun), in degrees
- **direction**: The direction (azimuth) angle of the light source (sun), in degrees
- **filename**: Character. Optional filename
- **normalize**: Logical. If TRUE, values below zero are set to zero and the results are multiplied with 255
- **...**: Standard additional arguments for writing RasterLayer files

Author(s)

Andrew Bevan, Robert J. Hijmans

References


See Also

- **terrain**

Examples

```r
## Not run:
alt <- getData('alt', country='CHE')
slope <- terrain(alt, opt='slope')
aspect <- terrain(alt, opt='aspect')
hill <- hillShade(slope, aspect, 40, 270)
plot(hill, col=grey(0:100/100), legend=FALSE, main='Switzerland')
plot(alt, col=rainbow(25, alpha=0.35), add=TRUE)

## End(Not run)
```
Description

Create a histogram of the values of a RasterLayer. For large datasets a sample is used.

Usage

```r
## S4 method for signature 'Raster'
hist(x, layer, maxpixels=100000, plot=TRUE, main, ...)
```

Arguments

- `x`: Raster* object
- `layer`: integer (or character) to indicate layer number (or name). Can be used to subset the layers to plot in a multilayer Raster* object
- `maxpixels`: integer. To regularly subsample very large objects
- `plot`: logical. Plot the histogram or only return the histogram values
- `main`: character. Main title(s) for the plot. Default is the value of `names`
- `...`: Additional arguments. See under Methods and at `hist`

Value

This function is principally used for the side-effect of plotting a histogram, but it also returns an S3 object of class `"histogram"` (invisibly if `plot=TRUE`).

See Also

`pairs`, `boxplot`

Examples

```r
r1 <- raster(nrows=50, ncols=50)
r1 <- setValues(r1, runif(ncell(r1)))
r2 <- setValues(r1, runif(ncell(r1)))
rs <- r1 + r2
rp <- r1 * r2
par(mfrow=c(2,2))
plot(rs, main='sum')
plot(rp, main='product')
hist(rs)
a = hist(rp)
a
```
Description

Create an "image" type plot of a RasterLayer. This is an implementation of a generic function in the graphics package. In most cases the `plot` function would be preferable because it produces a legend (and has some additional options).

Usage

```r
image(x, ...)  
## S4 method for signature 'RasterLayer'
image(x, maxpixels=500000, useRaster=TRUE, ...)

## S4 method for signature 'RasterStackBrick'
image(x, y=1, maxpixels=100000, useRaster=TRUE, main, ...)
```

Arguments

- **x**  
  Raster* object
- **maxpixels**  
  integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), `sampleRegular` is used before plotting
- **useRaster**  
  If TRUE, the rasterImage function is used for plotting. Otherwise the image function is used. This can be useful if rasterImage does not work well on your system (see note)
- **main**  
  character. Main plot title
- **...**  
  Any argument that can be passed to `image` (graphics package)
- **y**  
  If x is a RasterStack or RasterBrick: integer, character (layer name(s)), or missing to select which layer(s) to plot

Note

raster uses `rasterImage` from the graphics package. For unknown reasons this does not work on Windows Server and on a few versions of Windows XP. On that system you may need to use argument `useRaster=FALSE` to get a plot.

See Also

`plot, image, contour`

Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))
image(r)
```
**inifile**

*Read a .ini file*

**Description**

This function reads '.ini' files. These are text file databases that are organized in [sections] containing pairs of "name = value".

**Usage**

```r
readIniFile(filename, token='=', commenttoken=';', aslist=FALSE, case)
```

**Arguments**

- **filename**: Character. Filename of the .ini file
- **token**: Character. The character that separates the "name" (variable name) from the "value"
- **commenttoken**: Character. This token and everything that follows on the same line is considered a 'comment' that is not for machine consumption and is ignored in processing
- **aslist**: Logical. Should the values be returned as a list
- **case**: Optional. Function that operates on the text, such as `toupper` or `tolower`

**Details**

This function allows for using inistrings that have "=" as part of a value (but the token cannot be part of the 'name' of a variable!). Sections can be missing.

**Value**

A n*3 matrix of characters with columns: section, name, value; or a list if aslist=TRUE.

---

**initialize**

*Initialize a Raster object with values*

**Description**

Create a new RasterLayer with values reflecting a cell property: 'x', 'y', 'col', 'row', or 'cell'. Alternatively, a function can be used. In that case, cell values are initialized without reference to pre-existing values. E.g., initialize with a random number (`fun=runif`). While there are more direct ways of achieving this for small objects (see examples) for which a vector with all values can be created in memory, the init function will also work for Raster* objects with many cells.

**Usage**

```r
init(x, fun, filename="", ...)
```
Arguments

\textbf{x} \hspace{1cm} \text{Raster* object}

\textbf{fun} \hspace{1cm} \text{function to be applied. This must be a function that can take the number of cells as a single argument to return a vector of values with a length equal to the number of cells, such as \texttt{fun=runif}. You can also supply one of the following character values: 'x', 'y', 'row', 'col', or 'cell' to get the x or coordinate, row, col or cell number}

\textbf{filename} \hspace{1cm} \text{character. Optional output filename}

\textbf{...} \hspace{1cm} \text{Additional arguments as for \texttt{writeRaster}}

Value

\textbf{RasterLayer}

Note

For backwards compatibility, the character values valid for \texttt{fun} can also be passed as named argument \texttt{v}

Examples

\begin{verbatim}
r <- raster(ncols=36, nrows=18)
x <- init(r, fun='cell')
y <- init(r, fun=runif)

# there are different ways to set all values to 1
# for large rasters:
s1f <- function(x)(rep(1, x))
z1 <- init(r, fun=s1f, filename='test.grd', overwrite=TRUE)

# This is equivalent to (but not memory safe):
z2 <- setValues(r, rep(1, ncell(r)))
# or
r[] <- rep(1, ncell(r))
# or
r[] <- 1
\end{verbatim}
interpolate

Description

Make a RasterLayer with interpolated values using a fitted model object of classes such as 'gstat' (gstat package) or 'Krige' (fields package). That is, these are models that have location ('x' and 'y', or 'longitude' and 'latitude') as independent variables. If x and y are the only independent variables provide an empty (no associated data in memory or on file) RasterLayer for which you want predictions. If there are more spatial predictor variables provide these as a Raster* object in the first argument of the function. If you do not have x and y locations as implicit predictors in your model you should use `predict` instead.

Usage

```r
## S4 method for signature 'Raster'
interpolate(object, model, filename="", fun=predict, xyOnly=TRUE,
            xyNames=c('x', 'y'), ext=NULL, const=NULL, index=1, na.rm=TRUE, debug.level=1, ...)
```

Arguments

- `object`: Raster* object
- `model`: model object
- `filename`: character. Output filename (optional)
- `fun`: function. Default value is 'predict', but can be replaced with e.g. 'predict.se' (depending on the class of the model object)
- `xyOnly`: logical. If TRUE, values of the Raster* object are not considered as co-variables; and only x and y (longitude and latitude) are used. This should match the model
- `xyNames`: character. variable names that the model uses for the spatial coordinates. E.g., c('longitude', 'latitude')
- `ext`: Extent object to limit the prediction to a sub-region of x
- `const`: data.frame. Can be used to add a constant for which there is no Raster object for model predictions. This is particularly useful if the constant is a character-like factor value
- `index`: integer. To select the column if 'predict.model' returns a matrix with multiple columns
- `na.rm`: logical. Remove cells with NA values in the predictors before solving the model (and return NA for those cells). In most cases this will not affect the output. This option prevents errors with models that cannot handle NA values
- `debug.level`: for gstat models only. See ?
- `...`: additional arguments passed to the predict.'model' function

Value

Raster* object

See Also

`predict`, `predict.gstat`, `Tps`
Examples

## Not run:
## Thin plate spline interpolation with x and y only
# some example data
r <- raster(system.file("external/test.grd", package="raster"))
ra <- aggregate(r, 10)
xy <- data.frame(xyFromCell(ra, 1:ncell(ra)))
v <- getValues(ra)

#### Thin plate spline model
library(fields)
tps <- Tps(xy, v)
p <- raster(r)

# use model to predict values at all locations
p <- interpolate(p, tps)
p <- mask(p, r)

plot(p)
## change the fun from predict to fields::predictSE to get the TPS standard error
se <- interpolate(p, tps, fun=predictSE)
se <- mask(se, r)
plot(se)

## another variable; let’s call it elevation
elevation <- (init(r, 'x') * init(r, 'y')) / 10000000
names(elevation) <- 'elev'
elevation <- mask(elevation, r)

z <- extract(elevation, xy)

# add as another independent variable
xyz <- cbind(xy, z)
tps2 <- Tps(xyz, v)
p2 <- interpolate(elevation, tps2, xyOnly=FALSE)

# as a linear covariate
tps3 <- Tps(xy, v, Z=z)

# Z is a separate argument in Krig.predict, so we need a new function
# Internally (in interpolate) a matrix is formed of x, y, and elev (Z)
pfun <- function(model, x, ...) {
  predict(model, x[,1:2], Z=x[,3], ...)
}
p3 <- interpolate(elevation, tps3, xyOnly=FALSE, fun=pfun)

#### gstat examples
library(gstat)
data(meuse)

## inverse distance weighted (IDW)
It depends on the classes of the x and y what is returned.

If x is a Raster* object the extent of y is used, irrespective of the class of y, and a Raster* is returned. This is equivalent to `crop`.

```R
r <- raster(system.file("external/test.grd", package="raster"))
data(meuse)
mg <- gstat(id = "zinc", formula = zinc~1, locations = ~x+y, data=meuse,
         nmax=7, set=list(idp = .5))
z <- interpolate(r, mg)
z <- mask(z, r)

## kriging
coordinates(meuse) <- ~x+y
projection(meuse) <- projection(r)

## ordinary kriging
v <- fit.variogram(log(zinc)-1, meuse)
m <- fit.variogram(v, vgm(1, "Sph", 300, 1))
gOK <- gstat(NULL, "log.zinc", log(zinc)-1, meuse, model=m)
OK <- interpolate(r, gOK)

# examples below provided by Maurizio Marchi
## universal kriging
vu <- variogram(log(zinc)-elev, meuse)
uv <- fit.variogram(vu, vgm(1, "Sph", 300, 1))
gUK <- gstat(NULL, "log.zinc", log(zinc)-elev, meuse, model=mu)
names(r) <- 'elev'
UK <- interpolate(r, gUK, xyOnly=FALSE)

## co-kriging
gCoK <- gstat(NULL, 'log.zinc', log(zinc)-1, meuse)
gCoK <- gstat(gCoK, 'elev', elev-1, meuse)
gCoK <- gstat(gCoK, 'cadmium', cadmium-1, meuse)
gCoK <- gstat(gCoK, 'copper', copper-1, meuse)

plot(coV, type='b', main='Co-variogram')
coV.fit <- fit.lmc(coV, gCoK, vgm(model='Sph', range=1000))
plot(coV, coV.fit, main='Fitted Co-variogram')
coK <- interpolate(r, coV.fit)
plot(coK)

## End(Not run)
```
If \( x \) is a Spatial* object, a new Spatial* object is returned. If \( x \) or \( y \) has a data.frame, these are also returned (after merging if necessary) as part of a Spatial*DataFrame, and this is how \texttt{intersect} is different from \texttt{rgeos::gIntersection} on which it depends.

Intersecting SpatialPoints* with SpatialPoints* uses the extent (bounding box) of \( y \) to get the intersection. Intersecting of SpatialPoints* and SpatialLines* is not supported because of numerical inaccuracies with that. You can use \texttt{buffer}, to create SpatialPolygons* from SpatialLines* and use that in intersect. Or try \texttt{gIntersection}.

### Usage

```r
## S4 method for signature 'Extent,ANY'
intersect(x, y)

## S4 method for signature 'Raster,ANY'
intersect(x, y)

## S4 method for signature 'SpatialPoints,ANY'
intersect(x, y)

## S4 method for signature 'SpatialPolygons,SpatialPolygons'
intersect(x, y)

## S4 method for signature 'SpatialPolygons,SpatialLines'
intersect(x, y)

## S4 method for signature 'SpatialPolygons,SpatialPoints'
intersect(x, y)

## S4 method for signature 'SpatialLines,SpatialPolygons'
intersect(x, y)

## S4 method for signature 'SpatialLines,SpatialLines'
intersect(x, y)
```

### Arguments

- **x**: Extent, Raster*, SpatialPolygons*, SpatialLines* or SpatialPoints* object
- **y**: same as for \( x \)

### Value

- if \( x \) is an Extent object: Extent
- if \( x \) is a Raster* object: Raster*
- if \( x \) is a SpatialPoints* object: SpatialPoints*
- if \( x \) is a SpatialPolygons* object: SpatialPolygons*
- if \( x \) is a SpatialLines* object and if \( y \) is a SpatialLines* object: SpatialPoints*
- if \( x \) is a SpatialLines* object and if \( y \) is a SpatialPolygons* object: SpatialLines*
isLonLat

See Also

union, extent, crop

Examples

e1 <- extent(-10, 10, -20, 20)
e2 <- extent(0, 20, -40, 5)
intersect(e1, e2)

#SpatialPolgons
if (require(rgdal) & require(rgeos)) {
p <- shapefile(system.file("external/lux.shp", package="raster"))
b <- as(extent(6, 6.4, 49.75, 50), 'SpatialPolygons')
projection(b) <- projection(p)
i <- intersect(p, b)
plot(p)
plot(b, add=TRUE, col='red')
plot(i, add=TRUE, col='blue', lwd=2)
}

isLonLat                      Is this longitude/latitude data?

Description

Test whether a Raster* or other object has a longitude/latitude coordinate reference system (CRS) by inspecting the PROJ.4 coordinate reference system description. couldBeLonLat also returns TRUE if the CRS is NA but the x coordinates are within -365 and 365 and the y coordinates are within -90.1 and 90.1.

Usage

isLonLat(x)
couldBeLonLat(x, warnings=TRUE)

Arguments

x Raster* object
warnings logical. If TRUE, a warning is given if the CRS is NA or when the CRS is longitude/latitude but the coordinates do not match that

Value

Logical
Examples

```r
r <- raster()
islOnLat(r)
projection(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"
isLonLat(r)
```

### KML

Write a KML or KMZ file

**Description**

Export raster data to a KML file and an accompanying PNG image file. Multi-layer objects can be used to create an animation. The function attempts to combine these into a single (and hence more convenient) KMZ file (a zip file containing the KML and PNG files).

See package plotKML for more advanced functionality

**Usage**

```r
## S4 method for signature 'RasterLayer'
KML(x, filename, col=rev(terrain.colors(255)),
    colNA=NA, maxpixels=100000, blur=1, zip='', overwrite=FALSE, ...)

## S4 method for signature 'RasterStackBrick'
KML(x, filename, time=NULL, col=rev(terrain.colors(255)),
    colNA=NA, maxpixels=100000, blur=1, zip='', overwrite=FALSE, ...)

## S4 method for signature 'Spatial'
KML(x, filename, zip='', overwrite=FALSE, ...)
```

**Arguments**

- `x` Raster* object
- `filename` output filename
- `time` character vector with time labels for multilayer objects. The length of this vector should be `nlayers(x)` to indicate "when" or `nlayers(x)+1` to indicate "begin-end"
- `col` color scheme to be used (see `image`)
- `colNA` The color to use for the background (default is transparent)
- `maxpixels` maximum number of pixels. If `ncell(raster) > maxpixels`, `sampleRegular` is used to reduce the number of pixels
- `blur` Integer (default=1). Higher values help avoid blurring of isolated pixels (at the expense of a png file that is `blur^2` times larger)
- `zip` If there is no zip program on your path (on windows), you can supply the full path to a zip.exe here, in order to make a KMZ file
- `overwrite` logical. If TRUE, overwrite the file if it exists
- ... If `x` is a Raster* object, additional arguments that can be passed to `image`
Value

None. Used for the side-effect files written to disk.

Author(s)

This function was adapted for the raster package by Robert J. Hijmans, with ideas from Tony Fischerbach, and based on functions in the maptools package by Duncan Golicher, David Forrest and Roger Bivand.

Examples

```
## Not run:
# Meuse data from the sp package
data(meuse.grid)
b <- rasterFromXYZ(meuse.grid)
projection(b) <- "+init=epsg:28992"
# transform to longitude/latitude
p <- projectRaster(b, crs="+proj=longlat +datum=WGS84", method='ngb')
KML(p, file='meuse.kml')

## End(Not run)
```

Description

Create a RasterBrick with a Boolean layer for each class (value, or subset of the values) in a RasterLayer. For example, if the cell values of a RasterLayer indicate what vegetation type they are, this function will create a layer (presence/absence; dummy variable) for each of these classes. Classes and cell values are always truncated to integers.

You can supply a second spatially overlapping RasterLayer with larger cells (do not use smaller cells!). In this case the cell values are counts for each class. A similar result might be obtained more efficiently by using layerize with a single RasterLayer followed by aggregate(x, , sum).

Usage

```
## S4 method for signature 'RasterLayer,missing'
layerize(x, classes=NULL, falseNA=FALSE, filename='', ...)

## S4 method for signature 'RasterLayer,RasterLayer'
layerize(x, y, classes=NULL, filename='', ...)
```
Arguments

- **x**: RasterLayer
- **y**: RasterLayer or missing
- **classes**: numeric. The values (classes) for which layers should be made. If NULL all classes are used
- **falseNA**: logical. If TRUE, cells that are not of the class represented by a layer are NA rather than FALSE
- **filename**: character. Output filename (optional)
- **...**: Additional arguments as for `writeRaster`

Value

- RasterBrick

Examples

```r
r <- raster(nrow=36, ncol=72)
r[] <- round(runif(ncell(r))*5)
r[1:5] <- NA
b <- layerize(r)

r2 <- raster(nrow=10, ncol=10)
b2 <- layerize(r, r2)
```

layerStats

**Correlation and (weighted) covariance**

Description

Compute correlation and (weighted) covariance for multi-layer Raster objects. Like `cellStats` this function returns a few values, not a Raster* object (see `Summary_methods` for that).

Usage

```
layerStats(x, stat, w, asSample=TRUE, na.rm=FALSE, ...)
```

Arguments

- **x**: RasterStack or RasterBrick for which to compute a statistic
- **stat**: Character. The statistic to compute: either 'cov' (covariance), 'weighted.cov' (weighted covariance), or 'pearson' (correlation coefficient)
- **w**: RasterLayer with the weights (should have the same extent, resolution and number of layers as x) to compute the weighted covariance
- **asSample**: Logical. If TRUE, the statistic for a sample (denominator is n-1) is computed, rather than for the population (denominator is n)
- **na.rm**: Logical. Should missing values be removed?
- **...**: Additional arguments (none implemented)
Value

List with two items: the correlation or (weighted) covariance matrix, and the (weighted) means.

Author(s)

Jonathan A. Greenberg & Robert Hijmans. Weighted covariance based on code by Mort Canty

References

For the weighted covariance:


See Also

cellStats, covNwt, weightedNmean

Examples

```r
b <- brick(system.file("external/rlogo.grd", package="raster"))
layerStats(b, 'pearson')
layerStats(b, 'cov')

# weigh by column number
w <- init(b, v='col')
layerStats(b, 'weighted.cov', w=w)
```

Description

Local functions for two RasterLayer objects (using a focal neighborhood)

Usage

```r
## S4 method for signature 'RasterLayer,RasterLayer'
localFun(x, y, ngb=5, fun, filename='', ...)
```
Arguments

- **x**: RasterLayer or RasterStack/RasterBrick
- **y**: object of the same class as x, and with the same number of layers
- **ngb**: integer. rectangular neighbourhood size. Either a single integer or a vector of two integers c(rows, cols), such as c(3,3) to have a 3 x 3 focal window
- **fun**: function
- **filename**: character. Output filename (optional)
- **...**: additional arguments as for `writeRaster`

Value

RasterLayer

Note

The first two arguments that `fun` needs to accept are vectors representing the local cells of RasterLayer x and y (each of length `ngb * ngb`). It also must have an ellipsis (…) argument

See Also

corLocal, localFun

Examples

```r
set.seed(0)
b <- stack(system.file("external/rlogo.grd", package="raster"))
x <- flip(b[[2]], 'y') + runif(ncell(b))
y <- b[[1]] + runif(ncell(b))

f <- localFun(x, y, fun=cor)

## Not run:
# local regression:
rfun <- function(x, y, ...) {
  m <- lm(y~x)
  # return R^2
  summary(m)$r.squared
}

ff <- localFun(x, y, fun=rfun)
plot(f, ff)

## End(Not run)```

Description

The following logical (boolean) operators are available for computations with RasterLayer objects:

\&, |, and !

The following functions are available with a Raster\* argument:

is.na, is.nan, is.finite, is.infinite

Value

A Raster object with logical (TRUE/FALSE values)

Note

These are convenient operators/functions that are most useful for relatively small RasterLayers for which all the values can be held in memory. If the values of the output RasterLayer cannot be held in memory, they will be saved to a temporary file. In that case it could be more efficient to use calc instead.

See Also

Math-methods, overlay, calc

Examples

r <- raster(ncols=10, nrows=10)
r[] <- runif(ncell(r)) * 10
r1 <- r < 3 | r > 6
r2 <- !r1
r3 <- r >= 3 & r <= 6
r4 <- r2 == r3
r[r>3] <- NA
r5 <- is.na(r)
r[1:5]
run(r1[1:5])
run(r2[1:5])
run(r3[1:5])
Description

Create a new Raster* object that has the same values as x, except for the cells that are NA (or other maskvalue) in a 'mask'. These cells become NA (or other updatevalue). The mask can be either another Raster* object of the same extent and resolution, or a Spatial* object (e.g. SpatialPolygons) in which case all cells that are not covered by the Spatial object are set to updatevalue. You can use inverse=TRUE to set the cells that are not NA (or other maskvalue) in the mask, or not covered by the Spatial* object, to NA (or other updatevalue).

Usage

```r
## S4 method for signature 'RasterLayer,RasterLayer'
mask(x, mask, filename="", inverse=FALSE,
     maskvalue=NA, updatevalue=NA, updateNA=FALSE, ...)

## S4 method for signature 'RasterStackBrick,RasterLayer'
mask(x, mask, filename="", inverse=FALSE,
     maskvalue=NA, updatevalue=NA, updateNA=FALSE, ...)

## S4 method for signature 'RasterLayer,RasterStackBrick'
mask(x, mask, filename="", inverse=FALSE,
     maskvalue=NA, updatevalue=NA, updateNA=FALSE, ...)

## S4 method for signature 'RasterStackBrick,RasterStackBrick'
mask(x, mask, filename="", inverse=FALSE,
     maskvalue=NA, updatevalue=NA, updateNA=FALSE, ...)

## S4 method for signature 'Raster,Spatial'
mask(x, mask, filename="", inverse=FALSE,
     updatevalue=NA, updateNA=FALSE, ...)
```

Arguments

- **x**  
  Raster* object
- **mask**  
  Raster* object or a Spatial* object
- **filename**  
  character. Optional output filename
- **inverse**  
  logical. If TRUE, areas on mask that are _not_ the maskvalue are masked
- **maskvalue**  
  numeric. The value in mask that indicates the cells of x that should become updatevalue (default = NA)
- **updatevalue**  
  numeric. The value that cells of x should become if they are not covered by mask (and not NA)
- **updateNA**  
  logical. If TRUE, NA values outside the masked area are also updated to the the updatevalue (only relevant if the updatevalue is not NA)
Value matching for `Raster*` objects

Description

`match` returns a `Raster*` object with the position of the matched values. The cell values are the index of the table argument.

%in% returns a logical `Raster*` object indicating if the cells values were matched or not.

Usage

```r
match(x, table, nomatch = NA_integer_, incomparables = NULL)
```

Arguments

- `x` : `Raster*` object
- `table` : vector of the values to be matched against
- `nomatch` : the value to be returned in the case when no match is found. Note that it is coerced to integer
- `incomparables` : a vector of values that cannot be matched. Any value in `x` matching a value in this vector is assigned the `nomatch` value. For historical reasons, `FALSE` is equivalent to `NULL`
Math-methods

Value

Raster* object

See Also

calc, match

Examples

```r
r <- raster(nrow=10, ncol=10)
r[] <- 1:100
m <- match(r, c(5:10, 50:55))
n <- r %in% c(5:10, 50:55)
```

---

Mathematical functions

Description

Generic mathematical functions that can be used with a Raster* object as argument:

- abs
- sign
- sqrt
- ceiling
- floor
- trunc
- cummax
- cummin
- cumprod
- cumsum
- log
- log10
- log2
- log1p
- acos
- acosh
- asin
- asinh
- atan
- atanh
- exp
- expm1
- cos
- cosh
- sin
- sinh
- tan
- tanh

Note

You can use the, somewhat more flexible, function `calc` instead of the Math-methods.

See Also

Arith-methods, calc, overlay, atan2

Examples

```r
r1 <- raster(nrow=10, ncol=10)
r1 <- setValues(r1, runif(ncell(r1)) * 10)
r2 <- sqrt(r1)
s <- stack(r1, r2) - 5
b <- abs(s)
```
**merge**

Merge Raster* objects

**Description**

Merge Raster* objects to form a new Raster object with a larger spatial extent. If objects overlap, the values get priority in the same order as the arguments, but NA values are ignored (except when overlap=FALSE). See subs to merge a Raster* object and a data.frame.

**Usage**

```r
## S4 method for signature 'Raster,Raster'
merge(x, y, ..., tolerance=0.05, filename="", overlap=TRUE, ext=NULL)

## S4 method for signature 'RasterStackBrick,missing'
merge(x, ..., tolerance=0.05, filename="", ext=NULL)

## S4 method for signature 'Extent,ANY'
merge(x, y, ...)
```

**Arguments**

- **x**  
  Raster* or Extent object
- **y**  
  Raster* if x is a Raster* object (or missing). If x is an Extent, y can be an Extent or object from which an Extent can be extracted
- **...**  
  additional Raster or Extent objects (and/or arguments for writing files as in writeRaster)
- **tolerance**  
  numeric. permissible difference in origin (relative to the cell resolution). See all.equal
- **filename**  
  character. Output filename (optional)
- **overlap**  
  logical. If FALSE values of overlapping objects are based on the first layer, even if they are NA
- **ext**  
  Extent object (optional) to limit the output to that extent

**Details**

The Raster objects must have the same origin and resolution. In areas where the Raster objects overlap, the values of the Raster object that is first in the sequence of arguments will be retained. If you would rather use the average of cell values, or do another computation, you can use mosaic instead of merge.

**Value**

RasterLayer or RasterBrick
metadata

### Description

Get or set a metadata to a Raster object

### Usage

```r
metadata(x)
metadata(x) <- value
```

### Arguments

- **x**: Raster* object
- **value**: list with named elements. Each element may be another list of named elements (but these nested lists are not allowed to be lists themselves)

### Value

Raster* object or list

### Note

The metadata can contain single values or vectors of basic data types (character, integer, numeric) and Date. Some other types may also be supported. You cannot use a matrix or data.frame as a meta-data element.

---

**Examples**

```r
r1 <- raster(xmn=-150, ymn=60, ncols=30, nrows=30)
r1[] <- 1:ncell(r1)
r2 <- raster(xmn=-100, xmx=-50, ymx=50, ymn=30)
res(r2) <- c(xres(r1), yres(r1))
r2[] <- 1:ncell(r2)
rm <- merge(r1, r2)

# if you have many RasterLayer objects in a list
# you can use do.call:
x <- list(r1, r2)
# add arguments such as filename
# x$filename <- 'test.tif'
m <- do.call(merge, x)
```
modal

Examples

```r
r <- raster(nc=10, nr=10)
r[] <- 1:cell(r)

m <- list(wave=list(a=1, b=2, c=c('cool', 'important')), that=list(red='44', blue=1:5, days=as.Date(c('2014-1-15', '2014-2-15'))), this='888 miles from here', today=NA)

metadata(r) <- m
x <- writeRaster(r, 'test.grd', overwrite=TRUE)
metax <- metadata(x)

identical(metax, m)

## Not run:
# nested too deep
badmeta1 <- list(wave=list(a=1, b=2, c='x'), that=list(red='44', blue=list(bad=5)))
metadata(r) <- badmeta1

# missing names
badmeta2 <- list(wave=list(1, 2, c='x'), that=list(red='44', blue=14), this='8m')
metadata(r) <- badmeta2

# matrix not allowed
badmeta3 <- list(wave=list(a=1, b=matrix(1:4, ncol=2), c='x'), that=list(red='44'))
metadata(r) <- badmeta3

## End(Not run)
```

Description

Compute the mode for a vector of numbers, or across raster layers. The mode, or modal value, is the most frequent value in a set of values.

Usage

```r
## S4 method for signature 'ANY'
modal(x, ..., ties='random', na.rm=FALSE, freq=FALSE)

## S4 method for signature 'Raster'
modal(x, ..., ties='random', na.rm=FALSE, freq=FALSE)
```

Arguments

- `x`: vector of numbers (typically integers), characters, logicals, or factors, or a Raster* object
mosaic

... additional argument of the same type as x
ties character. Indicates how to treat ties. Either 'random', 'lowest', 'highest', 'first', or 'NA'
na.rm logical. If TRUE, NA values are ignored. If FALSE, NA is returned if x has any NA values
freq return the frequency of the modal value, instead of the modal value

Value

vector or RasterLayer. The vector has length 1 and is of the same type as x, except when x is a factor and additional arguments (values) are supplied, in which case the values are coerced to characters and a character value is returned.

Examples

data <- c(0,1,2,3,3,3,3,4,4,4,4,5,5,6,7,7,8,9,NA)
modal(data, na.rm=TRUE)

mosaic Merge Raster* objects using a function for overlapping areas

Description

Mosaic Raster* objects to form a new object with a larger spatial extent. A function is used to compute cell values in areas where layers overlap (in contrast to the merge function which uses the values of the 'upper' layer). All objects must have the same origin, resolution, and coordinate reference system.

Usage

## S4 method for signature 'Raster,Raster'
mosaic(x, y, ..., fun, tolerance=0.05, filename="")

Arguments

x Raster* object
y Raster* object
... Additional Raster or Extent objects (and/or arguments for writing files as in writeRaster)
fun Function. E.g. mean, min, or max. Must be a function that accepts a 'na.rm' argument
tolerance Numeric. permissible difference in origin (relative to the cell resolution). See all.equal
filename Character. Output filename (optional)
Details
The Raster objects must have the same origin and resolution.

Value
RasterLayer or RasterBrick object.

See Also
merge, extend

Examples
r <- raster(ncol=100, nrow=100)
r1 <- crop(r, extent(-10, 11, -10, 11))
r2 <- crop(r, extent(0, 20, 0, 20))
r3 <- crop(r, extent(9, 30, 9, 30))

r1[] <- 1:ncell(r1)
r2[] <- 1:ncell(r2)
r3[] <- 1:ncell(r3)

m1 <- mosaic(r1, r2, r3, fun=mean)

s1 <- stack(r1, r1+2)
s2 <- stack(r2, r2/2)
s3 <- stack(r3, r3+4)
m2 <- mosaic(s1, s2, s3, fun=min)

# if you have a list of Raster objects, you can use do.call
x <- list(r1, r2, r3)
names(x)[1:2] <- c('x', 'y')
x$fun <- mean
x$na.rm <- TRUE

y <- do.call(mosaic, x)

---

movingFun

Moving functions

Description
Helper function to compute 'moving' functions, such as the 'moving average'

Usage
movingFun(x, n, fun=mean, type='around', circular=FALSE, na.rm=FALSE)
Arguments

x A vector of numbers
n Size of the 'window', i.e. the number of sequential elements to use in the function
fun A function like mean, min, max, sum
type Character. One of 'around', 'to', or 'from'. The choice indicates which values should be used in the computation. The focal element is always used. If type is 'around', the other elements are before and after the focal element. Alternatively, you can select the elements preceding the focal element ('to') or those coming after it 'from'. For example, to compute the movingFun with n=3 for element 5 of a vector; 'around' used elements 4,5,6; 'to' used elements 3,4,5, and 'from' uses elements 5,6,7

circular Logical. If TRUE, the data are considered to have a circular nature (e.g. months of the year), and the last elements in vector x are used in the computation of the moving function of the first element(s) of the vector, and the first elements are used in the computation of the moving function for the last element(s)
na.rm Logical. If TRUE, NA values should be ignored (by fun)

Value

Numeric

Author(s)

Robert J. Hijmans, inspired by Diethelm Wuertz’ rollFun function in the fTrading package

Examples

movingFun(1:12, 3, mean)
movingFun(1:12, 3, mean, 'to')
movingFun(1:12, 3, mean, 'from')
movingFun(1:12, 3, mean, circular=TRUE)

v <- c(0,1,2,3,3,3,4,4,4,5,5,6,7,7,8,9,NA)
movingFun(v, n=5)
movingFun(v, n=5, na.rm=TRUE)

names Names of raster layers

Description

Get or set the names of the layers of a Raster* object
NAvalue

Usage

## S4 method for signature 'Raster'
names(x)

## S4 replacement method for signature 'Raster'
names(x) <- value

## S4 method for signature 'Raster'
labels(object)

Arguments

- **x**: Raster* object
- **object**: Raster* object
- **value**: character (vector)

Value

Character

See Also

nlayers, bands

Examples

```
r <- raster(ncols=5, nrows=5)
r[] <- 1:ncell(r)
s <- stack(r, r, r)
nlayers(s)
names(s)
names(s) <- c('a', 'b', 'c')
names(s)[2] <- 'hello world'
names(s)
s
labels(s)
```

**NAvalue**

*Set the NA value of a RasterLayer*

Description

NAvalue returns the value that is used to write NA values to disk (in 'raster' type files). If you set the NA value of a Raster* object, this value will be interpreted as NA when reading the values from a file. Values already in memory will not be affected.

If the NA value is smaller than zero, all values smaller or equal to that number will be set to NA.
Usage
NAvalue(x) <- value
NAvalue(x)

Arguments
x A Raster object
value the value to be interpreted as NA; set this before reading the values from the file. Integer values are matched exactly; for decimal values files any value <= the value will be interpreted as NA

Value
Returns or set the NA value used for storage on disk.

Examples

r1 <- raster(system.file("external/rlogo.grd", package="raster"))
r2 <- r1
NAvalue(r2)
NAvalue(r2) <- 255
#plot(r1)
#x11()
#plot(r2)

ncell Number or rows, columns, and cells of a Raster\* object

Description
Get the number of rows, columns, or cells of a Raster\* object.

Usage
ncol(x)
nrow(x)
ncell(x)
ncol(x) <- value
nrow(x) <- value

Arguments
x a Raster object
value row or column number (integer > 0)
Value

Integer

See Also

dim, extent, res

Examples

```r
r <- raster()
nCell(r)
nCol(r)
nRow(r)
dim(r)

nRow(r) <- 18
nCol(r) <- 36
# equivalent to
dim(r) <- c(18, 36)
```

<table>
<thead>
<tr>
<th>nlayers</th>
<th>Number of layers</th>
</tr>
</thead>
</table>

Description

Get the number of layers in a Raster* object, typically used with a (multilayer) RasterStack or RasterBrick object

Usage

```r
nLayers(x)
```

Arguments

- `x` Raster* object

Value

integer

See Also

names
Options

Global options for the raster package

Description

Set, inspect, reset, save a number of global options used by the raster package.

Most of these options are used when writing files to disk. They can be ignored by specific functions if the corresponding argument is provided as an argument to these functions.

The default location is returned by `rasterTmpDir`. It is the same as that of the R temp directory but you can change it (for the current session) with `rasterOptions(tmpdir="path")`.

To permanently set any of these options, you can add them to `<your R installation>/etc/Rprofile.site>`. For example, to change the default directory used to save temporary files, add a line like this: `options(rasterTmpDir='c:/temp/')` to that file. All temporary raster files in that folder that are older than 24 hrs are deleted when the raster package is loaded.

Function `tmpDir` returns the location of the temporary files

Usage

`rasterOptions(format, overwrite, datatype, tmpdir, tmptime, progress, timer, chunksize, maxmemory, todisk, setfileext, tolerance, standardnames, deprecatedwarnings, addheader, estimatemem, default=FALSE)`

tmpDir(create=TRUE)

Arguments

- `format` character. The default file format to use. See `writeFormats`
- `overwrite` logical. The default value for overwriting existing files. If `TRUE`, existing files will be overwritten
- `datatype` character. The default data type to use. See `dataType`
- `tmpdir` character. The default location for writing temporary files; See `rasterTmpFile`
- `tmptime` number > 1. The number of hours after which a temporary file will be deleted. As files are deleted when loading the raster package, this option is only useful if you save this option so that it is loaded when starting a new session
progress character. Valid values are "text", "window" and "" (the default in most functions, no progress bar)

timer Logical. If TRUE, the time it took to complete the function is printed

chunksize integer. Maximum number of cells to read/write in a single chunk while processing (chunk by chunk) disk based Raster* objects

maxmemory integer. Maximum number of cells to read into memory. I.e., if a Raster* object has more than this number of cells, canProcessInMemory will return FALSE

todisk logical. For debugging only. Default is FALSE and should normally not be changed. If TRUE, results are always written to disk, even if no filename is supplied (a temporary filename is used)

setfileext logical. Default is TRUE. If TRUE, the file extension will be changed when writing (if known for the file type). E.g. GTiff files will be saved with the .tif extension

tolerance numeric. The tolerance used when comparing the origin and resolution of Raster* objects. Expressed as the fraction of a single cell. This should be a number between 0 and 0.5

standardnames logical. Default is TRUE. Should names be standardized to be syntactically valid names (using make.names)

deprecatedwarnings logical. If TRUE (the default) a warning is generated when a deprecated (obsolete) function is used

addheader character. If not equal to "" (the default) an additional header file is written when a raster format file (grd/gri) is written. Supported formats are as in hdr

default logical. If TRUE, all options are set to their default values

create logical. If TRUE, the temporary files directory is created if it does not exist

estimatemem logical. If TRUE, the amount of RAM available is estimated, and the value of maxmemory is ignored

Value
list of the current options (invisibly). If no arguments are provided the options are printed.

See Also
options, rasterTmpFile

Examples

```r
## Not run:
rasterOptions()
rasterOptions(chunksize=2e+07)

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

Origin returns (or sets) the coordinates of the point of origin of a Raster* object. This is the point closest to (0, 0) that you could get if you moved towards that point in steps of the x and y resolution.

# Usage

```r
origin(x, ...) 
origin(x) <- value
```

# Arguments

- **x**
  - Raster* object
- **value**
  - numeric vector of length 1 or 2
- **...**
  - additional arguments. None implemented

# Value

A vector of two numbers (x and y coordinates), or a changed origin for x.

# See Also

- `extent`

# Examples

```r
r <- raster(xrn=-0.5, xmx = 9.5, ncol=10)
origin(r)
r
origin(r) <- 0
r
```
**Description**

Create a new Raster* object, based on two or more Raster* objects. (You can also use a single object, but perhaps `calc` is what you are looking for in that case).

You should supply a function `fun` to set the way that the RasterLayers are combined. The number of arguments in the function must match the number of Raster objects (or take any number). For example, if you combine two RasterLayers you could use multiply: `fun=function(x,y){return(x*y)}` percentage: `fun=function(x,y){return(100 * x / y)}`. If you combine three layers you could use `fun=function(x,y,z){return((x + y) * z)}`

Note that the function must work for vectors (not only for single numbers). That is, it must return the same number of elements as its input vectors. Alternatively, you can also supply a function such as `sum`, that takes `n` arguments (as '...'), and perhaps also has a `na.rm` argument, like in `sum(..., na.rm)`.

If a single multi-layer object is provided, its layers are treated as individual RasterLayer objects if the argument "unstack=TRUE" is used. If multiple objects are provided, they should have the same number of layers, or it should be possible to recycle them (e.g., 1, 3, and 9 layers, which would return a RasterBrick with 9 layers).

**Usage**

```r
## S4 method for signature 'Raster,Raster'
overlay(x, y, ..., fun, filename='', recycle=TRUE, forcefun=FALSE)

## S4 method for signature 'Raster,missing'
overlay(x, y, ..., fun, filename='', unstack=TRUE, forcefun=FALSE)
```

**Arguments**

- `x` : Raster* object
- `y` : Raster* object, or missing (only useful if `x` has multiple layers)
- `...` : Additional Raster objects (and/or arguments for writing files as in `writeRaster`)
- `fun` : Function to be applied. When using RasterLayer objects, the number of arguments of the function should match the number of Raster objects, or it should take any number of arguments. When using multi-layer objects the function should match the number of layers of the RasterStack/Brick object (unless unstack=FALSE)
- `filename` : Character. Output filename (optional)
- `recycle` : Logical. Should layers from Raster objects with fewer layers be recycled?
- `unstack` : Logical. Should layers be ustacked before computation (i.e. does the `fun` refer to individual layers in a multilayer object)?
- `forcefun` : Boolean. If TRUE, overlay will not attempt to internally use `apply` (it is rarely necessary to use this argument)
Details

Instead of the overlay function you can also use arithmetic functions such as *, /, +, - with Raster objects (see examples). In that case you cannot specify an output filename. Moreover, the overlay function should be more efficient when using large data files that cannot be loaded into memory, as the use of the complex arithmetic functions might lead to the creation of many temporary files.

While you can supply functions such as sum or mean, it would be more direct to use the Raster* objects as arguments to those functions (e.g. sum(r1, r2, r3))

See rasterize and extract for "overlays" involving Raster* objects and polygons, lines, or points.

Value

Raster* object

See Also

calc, Arith-methods

Examples

r <- raster(ncol=10, nrow=10)
r1 <- init(r, fun=runif)
r2 <- init(r, fun=runif)
r3 <- overlay(r1, r2, fun=function(x,y)(return(x+y)))

# long version for multiplication
r4 <- overlay(r1, r2, fun=function(x,y){(x*y)})

# use the individual layers of a RasterStack to get a RasterLayer
s <- stack(r1, r2)
r5 <- overlay(s, fun=function(x,y) x*y)
# equivalent to
r5c <- calc(s, fun=function(x) x[1]*x[2])

# Combine RasterStack and RasterLayer objects (s2 has four layers.
# r1 (one layer) and s (two layers) are recycled)
s2 <- stack(r1, r2, r3, r4)
b <- overlay(r1, s, s2, fun=function(x,y,z){return(x*y*z)})

# use a single RasterLayer (same as calc function)
r6 <- overlay(r1, fun=sqrt)

# multiplication with more than two layers
# (make sure the number of RasterLayers matches the arguments of 'fun')
r7 <- overlay(r1, r2, r3, r4, fun=function(a,b,c,d){return(a*b*c*d)})
# equivalent function, efficient if values can be loaded in memory
r8 <- r1 * r2 + r3 * r4

# Also works with multi-layer objects.
s1 <- stack(r1, r2, r3)
x <- overlay(s1, s1, fun=function(x,y)x+y+5)
# in this case the first layer of the shorter object is recycled.
# i.e., s2 is treated as stack(r1, r3, r1)
s2 <- stack(r1, r3)
y <- overlay(s1, s2, fun=sum)

Description

Pair plots of layers in a RasterStack or RasterBrick. This is a wrapper around graphics function pairs.

Usage

```r
## S4 method for signature 'RasterStackBrick'
pairs(x, hist=TRUE, cor=TRUE, use="pairwise.complete.obs", maxpixels=100000, ...)
```

Arguments

- `x`: RasterBrick or RasterStack
- `hist`: Logical. If TRUE a histogram of the values is shown on the diagonal
- `cor`: Logical. If TRUE the correlation coefficient is shown in the upper panels
- `use`: Argument passed to the `cor` function
- `maxpixels`: Integer. Number of pixels to sample from each layer of large Raster objects
- `...`: Additional arguments (only cex and main)

See Also

`boxplot`, `hist`, `density`

Examples

```r
r <- raster(system.file("external/test.grd", package="raster") )
s <- stack(r, 1/r, sqrt(r))
pairs(s)

## Not run:
# to make individual histograms:
hist(r)
# or scatter plots:
plot(r, 1/r)

## End(Not run)
```
Description

Perspective plot of a RasterLayer. This is an implementation of a generic function in the graphics package.

Usage

```r
## S4 method for signature 'RasterLayer'
persp(x, maxpixels=1e+05, ext=NULL, ...)

## S4 method for signature 'RasterStackBrick'
persp(x, y=1, maxpixels=10000, ext=NULL, ...)
```

Arguments

- `x` : Raster* object
- `y` : integer > 0 & <= `nlayers(x)` to select the layer of `x` if `x` is a RasterLayer or RasterBrick
- `maxpixels` : integer > 0. Maximum number of cells to use for the plot. If `maxpixels` < `ncell(x)`, `sampleRegular` is used before plotting
- `ext` : Extent. Can be used to zoom in a region (see also `zoom` and `crop(x, drawExtent())`)
- `...` : Any argument that can be passed to `persp` (graphics package)

See Also

- `plot3D`, `persp`, `contour`, `plot`

Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))
persp(r)
```
plot

Description

Plot (that is, make a map of) the values of a Raster* object, or make a scatterplot of their values.

Points, lines, and polygons can be drawn on top of a map using plot(..., add=TRUE), or with functions like points, lines, polygons

See the rasterVis package for more advanced (trellis/lattice) plotting of Raster* objects.

Usage

```r
## S4 method for signature 'Raster,ANY'
plot(x, y, maxpixels=500000, col, alpha=NULL,
     colNA=NA, add=FALSE, ext=NULL, useRaster=TRUE, interpolate=FALSE,
     addfun=NULL, nc, nr, maxnl=16, main, npretty=0, ...)

## S4 method for signature 'Raster,Raster'
plot(x, y, maxpixels=100000, cex, xlab, ylab, nc, nr,
     maxnl=16, main, add=FALSE, gridded=FALSE, ncol=25, nrow=25, ...)
```

Arguments

- **x**: Raster* object
- **y**: If x is a RasterStack or RasterBrick: integer, character (layer name(s)), or missing to select which layer(s) to plot. If missing, all RasterLayers in the RasterStack will be plotted (up to a maximum of 16). Or another Raster* object of the same extent and resolution, to produce a scatter plot of the cell values.
- **maxpixels**: integer > 0. Maximum number of cells to use for the plot. If maxpixels < ncell(x), sampleRegular is used before plotting. If gridded=TRUE maxpixels may be ignored to get a larger sample
- **col**: A color palette, i.e. a vector of n contiguous colors generated by functions like rainbow, heat.colors, topo.colors, bpy.colors or one or your own making, perhaps using colorRampPalette. If none is provided, rev(terrain.colors(255)) is used unless x has a 'color table'
- **alpha**: Number between 0 and 1 to set transparency. 0 is entirely transparent, 1 is not transparent (NULL is equivalent to 1)
- **colNA**: The color to use for the background (default is transparent)
- **add**: Logical. Add to current plot?
- **ext**: An extent object to zoom in a region (see also zoom and crop(x, drawExtent())
- **useRaster**: If TRUE, the rasterImage function is used for plotting. Otherwise the image function is used. This can be useful if rasterImage does not work well on your system (see note)
interpolate Logical. Should the image be interpolated (smoothed)? Only used when useRaster = TRUE

addfun Function to add additional items such as points or polygons to the plot (map). Typically containing statements like "points(xy); plot(polygons, add=TRUE)". This is particularly useful to add something to each map when plotting a multi-layer Raster* object.

npretty integer. Number of decimals for pretty labels on the axes

Graphical parameters. Any argument that can be passed to image.plot and to plot, such as axes=FALSE, main='title', ylab='latitude'

xlab Optional. x-axis label)

ylab Optional. y-axis label)

nc Optional. The number of columns to divide the plotting device in (when plotting multiple layers in a RasterLayer or RasterBrick object)

nr Optional. The number of rows to divide the plotting device in (when plotting multiple layers in a RasterLayer or RasterBrick object)

maxnl integer. Maximum number of layers to plot (for a multi-layer object)

main character. Main plot title

cex Symbol size for scatter plots

grounded logical. If TRUE the scatterplot is gridded (counts by cells)

ncol integer. Number of columns for gridding

nrow integer. Number of rows for gridding

Details

Most of the code for the plot function for a single Raster* object was taken from image.plot (fields package).

Raster objects with a color-table (e.g. a graphics file) are plotted according to that color table.

Note

raster uses rasterImage from the graphics package. For unknown reasons this does not work on Windows Server and on a few versions of Windows XP. On that system you may need to use argument useRaster=FALSE to get a plot.

See Also

The rasterVis package has lattice based methods for plotting Raster* objects (like spplot) red-green-blue plots (e.g. false color composites) can be made with plotRGB 

barplot, hist, text, persp, contour, pairs
# RasterLayer
\begin{verbatim}
r <- raster(nrows=10, ncols=10)
r <- setValues(r, 1:ncell(r))
plot(r)

e <- extent(r)
plot(e, add=TRUE, col='red', lwd=4)
e <- e / 2
plot(e, add=TRUE, col='red')
\end{verbatim}

# Scatterplot of 2 RasterLayers
\begin{verbatim}
r2 <- sqrt(r)
plot(r, r2)
plot(r, r2, gridded=TRUE)
\end{verbatim}

# Multi-layer object (RasterStack / Brick)
\begin{verbatim}
s <- stack(r, r2, r/r)
plot(s, 2)
\end{verbatim}

# two objects, different range, one scale:
r[] <- runif(ncell(r))
r2 <- r/2
brks <- seq(0, 1, by=0.1)
b <- length(brks)-1
cols <- rev(terrain.colors(b))
par(mfrow=c(1,2))
plot(r, breaks=brks, col=cols, lab=brks, zlim=c(0,1), main='first')
plot(r2, breaks=brks, col=cols, lab=brks, zlim=c(0,1), main='second')

# breaks and labels
\begin{verbatim}
x <- raster(nc=10, nr=10)
x[] <- runif(ncell(x))
br <- c(0, 0.25, 0.75, 1)
arg <- list(at=c(0.12,0.5,0.87), labels=c("Low","Med.","High"))
plot(x, col=terrain.colors(3), breaks=b)
plot(x, col=terrain.colors(3), breaks=brk, axis.args=arg)
par(mfrow=c(1,1))
\end{verbatim}

# color ramp
\begin{verbatim}
plot(x, col=colorRampPalette(c("red", "white", "blue"))(255))
\end{verbatim}

# adding random points to the map
\begin{verbatim}
xy <- cbind(-180 + runif(10) * 360, -90 + runif(10) * 180)
points(xy, pch=3, cex=5)
\end{verbatim}

# for SpatialPolygons do
# plot(pols, add=TRUE)
# adding the same points to each map of each layer of a RasterStack
fun <- function() {
  points(xy, cex=2)
  points(xy, pch=3, col='red')
}
plot(s, addfun=fun)

---

plotRGB

Red-Green-Blue plot of a multi-layered Raster object

Description

Make a Red-Green-Blue plot based on three layers (in a RasterBrick or RasterStack). Three layers (sometimes referred to as "bands" because they may represent different bandwidths in the electromagnetic spectrum) are combined such that they represent the red, green and blue channel. This function can be used to make 'true (or false) color images' from Landsat and other multi-band satellite images.

Usage

```
## S4 method for signature 'RasterStackBrick'
plotRGB(x, r=1, g=2, b=3, scale, maxpixels=500000, stretch=NULL, 
  ext=NULL, interpolate=FALSE, colNA='white', alpha, bgalpha, addfun=NULL, zlim=NULL, 
  zlimcol=NULL, axes=FALSE, xlab='', ylab='', asp=NULL, add=FALSE, ...)
```

Arguments

- `x` : RasterBrick or RasterStack
- `r` : integer. Index of the Red channel, between 1 and nlayers(x)
- `g` : integer. Index of the Green channel, between 1 and nlayers(x)
- `b` : integer. Index of the Blue channel, between 1 and nlayers(x)
- `scale` : integer. Maximum (possible) value in the three channels. Defaults to 255 or to the maximum value of `x` if that is known and larger than 255
- `maxpixels` : integer > 0. Maximum number of pixels to use
- `stretch` : character. Option to stretch the values to increase the contrast of the image: "lin" or "hist"
- `ext` : An Extent object to zoom in to a region of interest (see drawExtent)
- `interpolate` : logical. If TRUE, interpolate the image when drawing
- `colNA` : color for the background (NA values)
- `alpha` : transparency. Integer between 0 (transparent) and 255 (opaque)
- `bgalpha` : Background transparency. Integer between 0 (transparent) and 255 (opaque)
addfun Function to add additional items such as points or polygons to the plot (map). See plot
zlim numeric vector of length 2. Range of values to plot (optional)
zlimcol If NULL the values outside the range of zlim get the color of the extremes of the range. If zlimcol has any other value, the values outside the zlim range get the color of NA values (see colNA)
axes logical. If TRUE axes are drawn (and arguments such as main="title" will be honored)
xlab character. Label of x-axis
ylab character. Label of y-axis
asp numeric. Aspect (ratio of x and y. If NULL, and appropriate value is computed to match data for the longitude/latitude coordinate reference system, and 1 for planar coordinate reference systems
add logical. If TRUE add values to current plot
... graphical parameters as in plot or rasterImage

Author(s)
Robert J. Hijmans; stretch option based on functions by Josh Gray

See Also
plot

Examples
b <- brick(system.file("external/rlogo.grd", package="raster"))
plotRGB(b)
plotRGB(b, 3, 2, 1)
plotRGB(b, 3, 2, 1, stretch='hist')

pointDistance Distance between points

Description
Calculate the geographic distance between two (sets of) points on the WGS ellipsoid (lonlat=TRUE) or on a plane (lonlat=FALSE). If both sets do not have the same number of points, the distance between each pair of points is given. If both sets have the same number of points, the distance between each point and the corresponding point in the other set is given, except if allpairs=TRUE.

Usage
pointDistance(p1, p2, lonlat, allpairs=FALSE, ...)
pointDistance

Arguments

p1 x and y coordinate of first (set of) point(s), either as c(x, y), matrix(ncol=2), or SpatialPoints*

p2 x and y coordinate of second (set of) second point(s) (like for p1). If this argument is missing, a distance matrix is computed for p1

lonlat logical. If TRUE, coordinates should be in degrees; else they should represent planar (’Euclidean’) space (e.g. units of meters)

allpairs logical. Only relevant if the number of points in x and y is the same. If FALSE the distance between each point in x with the corresponding point in y is returned. If TRUE a full distance matrix is returned

... Additional arguments. None implemented

Value

A single value, or a vector, or matrix of values giving the distance in meters (lonlat=TRUE) or map-units (for instance, meters in the case of UTM) If p2 is missing, a distance matrix is returned

Author(s)

Robert J. Hijmans and Jacob van Etten. The distance for longitude/latitude data uses GeographicLib by C.F.F. Karney

See Also

distanceFromPoints, distance, gridDistance, spdistsN1. The geosphere package has many additional distance functions and other functions that operate on spherical coordinates

Examples

a <- cbind(c(1,5,55,31),c(3,7,20,22))
b <- cbind(c(4,2,8,65),c(50,-90,20,32))

pointDistance(c(0, 0), c(1, 1), lonlat=FALSE)
pointDistance(c(0, 0), c(1, 1), lonlat=TRUE)
pointDistance(c(0, 0), a, lonlat=TRUE)
pointDistance(a, b, lonlat=TRUE)

#Make a distance matrix
dst <- pointDistance(a, lonlat=TRUE)
# coerce to dist object
dst <- as.dist(dst)
**predict**

**Spatial model predictions**

**Description**

Make a Raster object with predictions from a fitted model object (for example, obtained with `lm`, `glm`). The first argument is a Raster object with the independent (predictor) variables. The names in the Raster object should exactly match those expected by the model. This will be the case if the same Raster object was used (via `extract`) to obtain the values to fit the model (see the example). Any type of model (e.g. `glm`, `gam`, `randomForest` for which a predict method has been implemented (or can be implemented) can be used.

This approach (predict a fitted model to raster data) is commonly used in remote sensing (for the classification of satellite images) and in ecology, for species distribution modeling.

**Usage**

```r
## S4 method for signature 'Raster'
predict(object, model, filename="", fun = predict, ext = NULL,
         const = NULL, index = 1, na.rm = TRUE, inf.rm = FALSE, factors = NULL,
         format, datatype, overwrite = FALSE, progress = '', ...)```

**Arguments**

- `object` Raster* object. Typically a multi-layer type (RasterStack or RasterBrick)
- `model` fitted model of any class that has a `predict` method (or for which you can supply a similar method as `fun` argument. E.g. `glm`, `gam`, or `randomForest`
- `filename` character. Optional output filename
- `fun` function. Default value is `predict`, but can be replaced with e.g. `predict.se` (depending on the type of model), or your own custom function.
- `ext` Extent object to limit the prediction to a sub-region of `x`
- `const` data.frame. Can be used to add a constant for which there is no Raster object for model predictions. Particularly useful if the constant is a character-like factor value for which it is currently not possible to make a RasterLayer
- `index` integer. To select the column(s) to use if `predict` returns a matrix with multiple columns
- `na.rm` logical. Remove cells with NA values in the predictors before solving the model (and return a NA value for those cells). This option prevents errors with models that cannot handle NA values. In most other cases this will not affect the output. An exception is when predicting with a boosted regression trees model because these return predicted values even if some (or all!) variables are NA
- `inf.rm` logical. Remove cells with values that are not finite (some models will fail with -Inf/Inf values). This option is ignored when `na.rm=FALSE`
predict

factors list with levels for factor variables. The list elements should be named with names that correspond to names in object such that they can be matched. This argument may be omitted for standard models such as 'glm' as the predict function will extract the levels from the model object, but it is necessary in some other cases (e.g. cforest models from the party package)

format character. Output file type. See writeRaster (optional)

datatype character. Output data type. See dataType (optional)

overwrite logical. If TRUE, "filename" will be overwritten if it exists

progress character. "text", "window", or "" (the default, no progress bar)

... additional arguments to pass to the predict.'model' function

Value

RasterLayer or RasterBrick

Note

For more on the use of the predict function see this resource on species distribution modeling.

See Also

Use interpolate if your model has 'x' and 'y' as implicit independent variables (e.g., in kriging).

Examples

# A simple model to predict the location of the R in the R-logo using 20 presence points # and 50 (random) pseudo-absence points. This type of model is often used to predict # species distributions. See the dismo package for more of that.

# create a RasterStack or RasterBrick with with a set of predictor layers
logo <- brick(system.file("external/rlogo.grd", package="raster"))

names(logo)

## Not run:
## the predictor variables
par(mfrow=c(2,2))

plotRGB(logo, main='logo')

plot(logo, 1, col=rgb(cbind(0.255,0,0), maxColorValue=255))

plot(logo, 2, col=rgb(cbind(0,0.255,0), maxColorValue=255))

plot(logo, 3, col=rgb(cbind(0,0,0.255), maxColorValue=255))

par(mfrow=c(1,1))

## End(Not run)

# known presence and absence points
p <- matrix(c(48, 48, 48, 53, 50, 46, 54, 70, 84, 85, 74, 84, 95, 85, 66, 42, 26, 4, 19, 17, 7, 14, 26, 29, 39, 45, 51, 56, 46, 38, 31, 22, 34, 66, 70, 73, 63, 46, 43, 28), ncol=2)

a <- matrix(c(22, 33, 64, 85, 92, 94, 59, 27, 30, 64, 60, 33, 31, 9,
# extract values for points
xy <- rbind(cbind(1, p), cbind(0, a))
v <- data.frame(cbind(pa=xy[,1], extract(logo, xy[,2:3])))

# build a model, here an example with glm
model <- glm(formula=p~., data=v)

# predict to a raster
r1 <- predict(logo, model, progress='text')
plot(r1)
points(p, bg='blue', pch=21)
points(a, bg='red', pch=21)

# use a modified function to get a RasterBrick with p and se
# from the glm model. The values returned by 'predict' are in a list,
# and this list needs to be transformed to a matrix
predfun <- function(model, data) {
  v <- predict(model, data, se.fit=TRUE)
  cbind(p=as.vector(v$fit), se=as.vector(v$se.fit))
}

# predfun returns two variables, so use index=1:2
r2 <- predict(logo, model, fun=predfun, index=1:2)

## Not run:
## You can use multiple cores to speed up the predict function
## by calling it via the clusterR function (you may need to install the snow package)
beginCluster()
r1c <- clusterR(logo, predict, args=list(model))
r2c <- clusterR(logo, predict, args=list(model=model, fun=predfun, index=1:2))

## End(Not run)

# principal components of a RasterBrick
# here using sampling to simulate an object too large
# too feed all its values to prcomp
sr <- sampleRandom(logo, 100)
pca <- prcomp(sr)

# note the use of the 'index' argument
x <- predict(logo, pca, index=1:3)
plot(x)

## Not run:
## partial least square regression
library(pls)
model <- plsR(formula=p~., data=v)
# this returns an array:
predict(model, v[1:5,])
# write a function to turn that into a matrix
pfun <- function(x, data) {
  y <- predict(x, data)
  d <- dim(y)
  dim(y) <- c(prod(d[1:2]), d[3])
  y
}

pp <- predict(logo, model, fun=pfun, index=1:3)

# Random Forest

library(randomForest)
rfmod <- randomForest(pa ~ ., data=v)

## note the additional argument "type='response'" that is
## passed to predict.randomForest
r3 <- predict(logo, rfmod, type='response', progress='window')

## get a RasterBrick with class membership probabilities
vv <- v
vv$pa <- as.factor(vv$pa)
rfmod2 <- randomForest(pa ~ ., data=vv)
r4 <- predict(logo, rfmod2, type='prob', index=1:2)
spplot(r4)

# cforest (other Random Forest implementation) example with factors argument
v$red <- as.factor(round(v$red/100))
logo$red <- round(logo[[1]]/100)

library(party)
m <- cforest(pa ~ ., control=cforest_unbiased(mtry=3), data=v)
f <- list(levels(v$red))
names(f) <- 'red'
pc <- predict(logo, m, OOB=TRUE, factors=f)

# knn example, using calc instead of predict
library(class)
cl <- factor(c(rep(1, nrow(p)), rep(0, nrow(a))))
train <- extract(logo, rbind(p, a))
k <- calc(logo, function(x) as.integer(as.character(knn(train, x, cl))))

## End(Not run)
Description

These are low level functions that can be used by programmers to develop new functions. If in doubt, it is almost certain that you do not need these functions as these are already embedded in all other functions in the raster package.

canProcessInMemory is typically used within functions. In the raster package this function is used to determine if the amount of memory needed for the function is available. If there is not enough memory available, the function returns FALSE, and the function that called it will write the results to a temporary file.

readStart opens file connection(s) for reading, readStop removes it.

pbCreate creates a progress bar, pbStep sets the progress, and pbClose closes it.

Usage

canProcessInMemory(x, n=4)
pbCreate(nsteps, progress, style=3, label='Progress', ...)
pbStep(pb, step=NULL, label='')
pbClose(pb, timer)
readStart(x, ...)
readStop(x)
getCluster()
returnCluster()

Arguments

x RasterLayer or RasterBrick object (for connections) or RasterStack object (can-ProcessInMemory)
n integer. The number of copies of the Raster* object cell values that a function needs to be able to have in memory
nsteps integer. Number of steps the progress bar will make from start to end (e.g. nrow(raster))
progress character. 'text', 'window', or "
style style for text progress bar. See txtProgressBar
label character. Label for the window type progress bar
... additional arguments (None implemented, except for 'silent=TRUE' for read-Start for files read with gdal, and other arguments passed to gdal.open)
pb progress bar object created with pbCreate
step which step is this ( 1 <= step <= nsteps ). If step is NULL, a single step is taken
timer logical. If TRUE, time to completion will be printed. If missing, the value will be taken from the rasterOptions

Value

canProcessInMemory: logical
closeConnection: RasterLayer or RasterBrick object
getCluster: snow cluster object
Examples

```r
r <- raster(nrow=100, ncol=100)
canProcessInMemory(r, 4)
r <- raster(nrow=100000, ncol=100000)
canProcessInMemory(r, 2)
```

projection

Get or set a coordinate reference system (projection)

Description

Get or set the coordinate reference system (CRS) of a Raster* object.

Usage

```r
## S4 method for signature 'ANY'
crs(x, asText=FALSE, ...)

projection(x, asText=TRUE)

projection(x) <- value
```

Arguments

- `x`: Raster* or Spatial object
- `asText`: logical. If TRUE, the projection is returned as text. Otherwise a CRS object is returned
- `...`: additional arguments (not implemented)
- `value`: CRS object or a character string describing a projection and datum in the PROJ.4 format

Details

projections are done by with the PROJ.4 library exposed by rgdal

Value

Raster*, Spatial*, CRS, or character object

Note

crs replaces earlier function projection. For compatibility with sp you can use proj4string instead of crs.
projectRaster

Description

Project the values of a Raster* object to a new Raster* object with another projection (coordinate reference system, CRS). You can do this by providing the new projection as a single argument in which case the function sets the extent and resolution of the new object. To have more control over the transformation, and, for example, to assure that the new object lines up with other datasets, you can provide a Raster* object with the properties that the input data should be projected to.

projectExtent returns a RasterLayer with a projected extent, but without any values. This RasterLayer can then be adjusted (e.g. by setting its resolution) and used as a template ‘to’ in projectRaster.

Usage

```
projectRaster(from, to, res, crs, method="bilinear",
               alignOnly=FALSE, over=FALSE, filename="", ...)  
```

projectExtent(object, crs)

Arguments

- **from**: Raster* object
- **to**: Raster* object with the parameters to which 'from' should be projected
- **res**: single or (vector of) two numerics. To, optionally, set the output resolution if 'to' is missing
- **crs**: character or object of class 'CRS'. PROJ.4 description of the coordinate reference system. In projectRaster this is used to set the output CRS if 'to' is missing, or if 'to' has no valid CRS
- **method**: method used to compute values for the new RasterLayer. Either 'ngb' (nearest neighbor), which is useful for categorical variables, or 'bilinear' (bilinear interpolation; the default value), which is appropriate for continuous variables.
- **alignOnly**: logical. Use to or other parameters only to align the output (i.e. same origin and resolution), but use the projected extent from from
over logical. If TRUE wrapping around the date-line is turned off. This can be desirable for global data (to avoid mapping the same areas twice) but it is not desirable in other cases

filename character. Output filename

... additional arguments as for writeRaster

object Raster* object

Details

There are two approaches you can follow to project the values of a Raster object.

1) Provide a crs argument, and, optionally, a res argument, but do not provide a to argument.

2) Create a template Raster with the CRS you want to project to. You can use an existing object, or use projectExtent for this or an existing Raster* object. Also set the number of rows and columns (or the resolution), and perhaps adjust the extent. The resolution of the output raster should normally be similar to that of the input raster. Then use that object as from argument to project the input Raster to. This is the preferred method because you have most control. For example you can assure that the resulting Raster object lines up with other Raster objects.

Projection is performed using the PROJ.4 library accessed through the rgdal package.

One of the best places to find PROJ.4 coordinate reference system descriptions is http://www.spatialreference.org.

You can also consult this page: http://geotiff.maptools.org/proj_list to find the parameter options and names for projections.

Also see projInfo('proj'), projInfo('ellps'), and projInfo('datum') for valid PROJ.4 values.

Value

RasterLayer or RasterBrick object.

Note

Vector (points, lines, polygons) can be transformed with spTransform.

projectExtent does not work very well when transforming projected circumpolar data to (e.g.) longitude/latitude. With such data you may need to adjust the returned object. E.g. do ymax(object) <= 90

Author(s)

Robert J. Hijmans and Joe Cheng

See Also

resample, CRS-class, projInfo, spTransform
Examples

```r
# create a new (not projected) RasterLayer with cellnumbers as values
r <- raster(xmn=-110, xmx=-90, ymn=40, ymx=60, ncol=40, nrow=40)
r <- setValues(r, 1:ncell(r))
projection(r)
# proj.4 projection description
newproj <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"

# we need the rgdal package for this
if (require(rgdal)) {
  # simplest approach
  pr1 <- projectRaster(r, crs=newproj)

  # alternatively also set the resolution
  pr2 <- projectRaster(r, crs=newproj, res=20000)

  # inverse projection, back to the properties of 'r'
  inv <- projectRaster(pr2, r)

  # to have more control, provide an existing Raster object, here we create one
  # using projectExtent (no values are transferred)
  pr3 <- projectExtent(r, newproj)
  # Adjust the cell size
  res(pr3) <- 200000
  # now project
  pr3 <- projectRaster(r, pr3)

  # Not run:
  # using a higher resolution
  res(pr1) <- 10000
  pr <- projectRaster(r, pr1, method='bilinear')
  inv <- projectRaster(pr, r, method='bilinear')
  dif <- r - inv
  # small difference
  plot(dif)

  # End(Not run)
}
```

---

**Description**

Properties of the values of the file that a RasterLayer object points to
dataSize returns the number of bytes used for each value (pixel, grid cell) dataSigned is TRUE for data types that include negative numbers.
Usage

`quantile(x, ...)`

Arguments

- `x`: Raster object
- `...`: Additional arguments: `na.rm=TRUE`, `ncells=NULL`, and additional arguments to the stats::quantile function, see `quantile`. `ncells` can be used to set the number of cells to be sampled, for very large raster datasets.

Value

A vector of quantiles

Description

Compute quantiles for the cell values of a RasterLayer. If you want to compute quantiles for each cell across a number of layers, you can use `calc(x, fun=quantile).`
raster

See Also
density, cellStats

Examples

r <- raster(ncol=100, nrow=100)
r[] <- rnorm(ncell(r), 0, 50)
quantile(r)
quantile(r, probs = c(0.25, 0.75), type=7, names = FALSE)

Create a RasterLayer object

Description

Methods to create a RasterLayer object. RasterLayer objects can be created from scratch, a file, an Extent object, a matrix, an 'image' object, or from a Raster*, Spatial*, im (spatstat) asc, kase (adehabitat*), grf (geoR) or kde object.

In many cases, e.g. when a RasterLayer is created from a file, it does (initially) not contain any cell (pixel) values in (RAM) memory, it only has the parameters that describe the RasterLayer. You can access cell-values with getValue, extract and related functions. You can assign new values with setValue and with replacement.

For an overview of the functions in the raster package have a look here: raster-package.

Usage

## S4 method for signature 'character'
raster(x, band=1, ...)

## S4 method for signature 'RasterLayer'
raster(x)

## S4 method for signature 'RasterStack'
raster(x, layer=0)

## S4 method for signature 'RasterBrick'
raster(x, layer=0)

## S4 method for signature 'missing'
raster(nrows=100, ncols=360, xmn=-180, xmx=180, ymn=-90, ymx=90,
crs, ext, resolution, vals=NULL)

## S4 method for signature 'Extent'
raster(x, nrows=10, ncols=10, crs=NA, ...)

## S4 method for signature 'matrix'
raster(x, xmn=0, xmx=1, ymn=0, ymx=1, crs=NA, template=NULL)

## S4 method for signature 'big.matrix'
raster(x, xmn=0, xmx=1, ymn=0, ymx=1, crs=NA, template=NULL)

## S4 method for signature 'Spatial'
raster(x, origin, ...)

## S4 method for signature 'SpatialGrid'
raster(x, layer=1, values=TRUE)

## S4 method for signature 'SpatialPixels'
raster(x, layer=1, values=TRUE)

## S4 method for signature 'sf'
raster(x, origin, ...)

Arguments

- **x**: filename (character), Extent, Raster*, sf, SpatialPixels*, SpatialGrid*, object, 'image', matrix, im, or missing. Supported file types are the 'native' raster package format and those that can be read via rgdal (see readGDAL).
- **band**: integer. The layer to use in a multi-layer file. Additional arguments, see Details.
- **layer**: integer. The layer (variable) to use in a multi-layer file, or the layer to extract from a RasterStack/Brick or SpatialPixelsDataFrame or SpatialGridDataFrame. An empty RasterLayer (no associated values) is returned if layer=0.
- **values**: logical. If TRUE, the cell values of 'x' are copied to the RasterLayer object that is returned.
- **nrows**: integer > 0. Number of rows.
- **ncols**: integer > 0. Number of columns.
- **xmn**: minimum x coordinate (left border).
- **xmx**: maximum x coordinate (right border).
- **ymn**: minimum y coordinate (bottom border).
- **ymx**: maximum y coordinate (top border).
- **ext**: object of class Extent. If present, the arguments xmn, xmx, ymn and ynx are ignored.
- **crs**: character or object of class CRS. PROJ.4 type description of a Coordinate Reference System (map projection). If this argument is missing, and the x coordinates are within -360 .. 360 and the y coordinates are within -90 .. 90, "+proj=longlat +datum=WGS84" is used. Also see under Details if x is a character (filename).
- **resolution**: numeric vector of length 1 or 2 to set the resolution (see res). If this argument is used, arguments ncols and nrows are ignored.
- **vals**: optional. Values for the new RasterLayer. Accepted formats are as for setValues.
origin minimum y coordinate (bottom border)
template Raster* or Extent object used to set the extent (and CRS in case of a Raster* object). If not NULL, arguments xmn, xmx, ymn, ymx and crs (unless template is an Extent object) are ignored

Details

If x is a filename, the following additional variables are recognized:

sub: positive integer. Subdataset number for a file with subdatasets

native: logical. Default is FALSE except when package rgdal is missing. If TRUE, reading and writing of IDRISI, BIL, BSQ, BIP, SAGA, and Arc ASCII files is done with native (raster package) drivers, rather than via rgdal. 'raster' and netcdf format files are always read with native drivers.

RAT: logical. The default is TRUE, in which case a raster attribute table is created for files that have one

offset: integer. To indicate the number of header rows on non-standard ascii files (rarely useful; use with caution)

crs: character. PROJ.4 string to set the CRS. Ignored when the file provides a CRS description that can be interpreted.

If x represents a NetCDF file, the following additional variable is recognized:

varname: character. The variable name, such as 'tasmax' or 'pr'. If not supplied and the file has multiple variables, a guess will be made (and reported)

lvar: integer > 0 (default=3). To select the 'level variable' (3rd dimension variable) to use, if the file has 4 dimensions (e.g. depth instead of time)

level: integer > 0 (default=1). To select the 'level' (4th dimension variable) to use, if the file has 4 dimensions, e.g. to create a RasterBrick of weather over time at a certain height.

To use NetCDF files the ncdf4 package needs to be available. It is assumed that these files follow, or are compatible with, the CF-1 convention (The GMT format may also work). If the ncdf file does not have a standard extension (which is used to recognize the file format), you can use argument ncdf=TRUE to indicate the format.

If x is a Spatial or an Extent object, additional arguments are for the method with signature 'missing'

Value

RasterLayer

See Also

stack, brick
Examples

# Create a RasterLayer object from a file
# N.B.: For your own files, omit the 'system.file' and 'package="raster"' bits
# these are just to get the path to files installed with the package

f <- system.file("external/test.grd", package="raster")
f
r <- raster(f)

logo <- raster(system.file("external/rlogo.grd", package="raster"))

# from scratch
r1 <- raster(nrows=108, ncols=21, xmin=0, xmax=10)

# from an Extent object
e <- extent(r)
r2 <- raster(e)

# from another Raster* object
r3 <- raster(r)
s <- stack(r, r, r)
r4 <- raster(s)
r5 <- raster(s, 3)

## Not run:
# from NSIDC sea ice concentration file
baseurl <- "ftp://sidads.colorado.edu/pub/DATASETS/"
# southern hemisphere
f1 <- paste(baseurl,
  "nsidc0051_gsfc_nasateam_seaice/final-gsfc/south/daily/2013/nt_20130114_f17_v01_s.bin",
  sep="")
# or northern hemisphere
f2 <- paste(baseurl,
  "nsidc0051_gsfc_nasateam_seaice/final-gsfc/north/daily/2013/nt_20130105_f17_v01_n.bin",
  sep="")

if (!file.exists(basename(f1))) download.file(f1, basename(f1), mode = "wb")
ice1 <- raster(basename(f1))

if (!file.exists(basename(f2))) download.file(f2, basename(f2), mode = "wb")
ice2 <- raster(basename(f2))

## End(Not run)
Description

A raster is a database organized as a rectangular grid that is sub-divided into rectangular cells of equal area (in terms of the units of the coordinate reference system). The 'raster' package defines a number of "S4 classes" to manipulate such data.

The main user level classes are RasterLayer, RasterStack and RasterBrick. They all inherit from BasicRaster and can contain values for the raster cells.

An object of the RasterLayer class refers to a single layer (variable) of raster data. The object can point to a file on disk that holds the values of the raster cells, or hold these values in memory. Or it can not have any associated values at all.

A RasterStack represents a collection of RasterLayer objects with the same extent and resolution. Organizing RasterLayer objects in a RasterStack can be practical when dealing with multiple layers; for example to summarize their values (see calc) or in spatial modeling (see predict).

An object of class RasterBrick can also contain multiple layers of raster data, but they are more tightly related. An object of class RasterBrick can refer to only a single (multi-layer) data file, whereas each layer in a RasterStack can refer to another file (or another band in a multi-band file). This has implications for processing speed and flexibility. A RasterBrick should process quicker than a RasterStack (irrespective if values are on disk or in memory). However, a RasterStack is more flexible as a single object can refer to layers that have values stored on disk as well as in memory. If a layer that does not refer to values on disk (they only exists in memory) is added to a RasterBrick, it needs to load all its values into memory (and this may not be possible because of memory size limitations).

Objects can be created from file or from each other with the following functions: raster, brick and stack.

Raster* objects can also be created from SpatialPixels* and SpatialGrid* objects from the sp package using as, or simply with the function raster, brick, or stack. Vice versa, Raster* objects can be coerced into a sp type object with as(), e.g. as(x, 'SpatialGridDataFrame').

Common generic methods implemented for these classes include:

summary, show, dim, and plot, ...

[ is implemented for RasterLayer.

The classes described above inherit from the BasicRaster class which inherits from BasicRaster. The BasicRaster class describes the main properties of a raster such as the number of columns and rows, and it contains an object of the link[exterior]{Extent-class} to describe its spatial extent (coordinates). It also holds the 'coordinate reference system' in a slot of class CRS-class defined in the sp package. A BasicRaster cannot contain any raster cell values and is therefore seldomly used.

The Raster* class inherits from BasicRaster. It is a virtual class; which means that you cannot create an object of this class. It is used only to define methods for all the classes that inherit from it (RasterLayer, RasterStack and RasterBrick). Another virtual class is the RasterStackBrick class. It is formed by a class union of RasterStack and RasterBrick. You cannot make objects of it, but methods defined for objects of this class as arguments will accept objects of the RasterLayer and RasterStack as that argument.

Classes RasterLayer and RasterBrick have a slot with an object of class RasterFile that describes the properties of the file they point to (if they do). RasterLayer has a slot with an object of class SingleLayerData, and the RasterBrick class has a slot with an object of class MultipleLayerData. These 'datalayer' classes can contain (some of) the values of the raster cells.
These classes are not further described here because users should not need to directly access these slots. The 'setter' functions such as setValues should be used instead. Using such 'setter' functions is much safer because a change in one slot should often affect the values in other slots.

**Objects from the Class**

Objects can be created by calls of the form `new("RasterLayer", ...), or with the helper functions such as `raster`.

**Slots**

Slots for RasterLayer and RasterBrick objects

- title: Character
- file: Object of class ".RasterFile"
- data: Object of class ".SingleLayerData" or ".MultipleLayerData"
- history: To record processing history, not yet in use
- legend: Object of class `.RasterLegend`, Default legend. Should store preferences for plotting. Not yet implemented except that it stores the color table of images, if available
- extent: Object of `Extent-class`
- ncols: Integer
- nrows: Integer
- crs: Object of class "CRS", i.e. the coordinate reference system. In Spatial* objects this slot is called 'proj4string'

**Examples**

```r
showClass("RasterLayer")
```

---

**rasterFromCells**

*Subset a raster by cell numbers*

**Description**

This function returns a new raster based on an existing raster and cell numbers for that raster. The new raster is cropped to the cell numbers provided, and, if `values=TRUE` has values that are the cell numbers of the original raster.

**Usage**

```r
rasterFromCells(x, cells, values=TRUE)
```
Arguments

- **x**: Raster* object (or a SpatialPixels* or SpatialGrid* object)
- **cells**: vector of cell numbers
- **values**: Logical. If TRUE, the new RasterLayer has cell values that correspond to the cell numbers of x

Details

Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the Raster* object.

Value

RasterLayer

See Also

rowFromCell

Examples

```r
r <- raster(ncols=100, nrows=100)
cells <- c(3:5, 210)
r <- rasterFromCells(r, cells)
cbind(1:ncell(r), getValues(r))
```

---

`rasterFromXYZ`  
Create a Raster* object from x, y, z values

Description

Create a Raster* object from x, y and z values. x and y represent spatial coordinates and must be on a regular grid. If the resolution is not supplied, it is assumed to be the minimum distance between x and y coordinates, but a resolution of up to 10 times smaller is evaluated if a regular grid can otherwise not be created. z values can be single or multiple columns (variables) If the exact properties of the RasterLayer are known beforehand, it may be preferable to simply create a new RasterLayer with the raster function instead, compute cell numbers and assign the values with these (see example below).

Usage

```r
rasterFromXYZ(xyz, res=c(NA,NA), crs=NA, digits=5)
```
Arguments

- `xyz` matrix or data.frame with at least three columns: x and y coordinates, and values (z). There may be several 'z' variables (columns)
- `res` numeric. The x and y cell resolution (optional)
- `crs` CRS object or a character string describing a projection and datum in PROJ.4 format
- `digits` numeric, indicating the requested precision for detecting whether points are on a regular grid (a low number of digits is a low precision)

Value

RasterLayer or RasterBrick

See Also

See `rasterize` for points that are not on a regular grid

Examples

```r
r <- raster(nrow=10, ncol=10, xmn=0, xmx=10, ymn=0, ymx=10, crs=NA)
r[] <- runif(ncell(r))
r[r<0.5] <- NA
xyz <- rasterToPoints(r)

r2 <- rasterFromXYZ(xyz)

# equivalent to:
r3 <- raster(nrow=10, ncol=10, xmn=0, xmx=10, ymn=0, ymx=10)
cells <- cellFromXY(r3, xyz[,1:2])
r3[cells] <- xyz[,3]
```

rasterize  

*Rasterize points, lines, or polygons*

Description

Transfer values associated with 'object' type spatial data (points, lines, polygons) to raster cells. For polygons, values are transferred if the polygon covers the center of a raster cell. For lines, values are transferred to all cells that are touched by a line. You can combine this behaviour by rasterizing polygons as lines first and then as polygons.

If x represents points, each point is assigned to a grid cell. Points that fall on a border between cells are placed in the cell to the right and/or in the cell below. The value of a grid cell is determined by the values associated with the points and function `fun`. 
Usage

```r
## S4 method for signature 'matrix,Raster'
rasterize(x, y, field, fun='last', background=NA,
  mask=FALSE, update=FALSE, updateValue='all', filename="", na.rm=TRUE, ...)
## S4 method for signature 'SpatialPoints,Raster'
rasterize(x, y, field, fun='last', background=NA,
  mask=FALSE, update=FALSE, updateValue='all', filename="", na.rm=TRUE, ...)
## S4 method for signature 'SpatialLines,Raster'
rasterize(x, y, field, fun='last', background=NA,
  mask=FALSE, update=FALSE, updateValue='all', filename="", ...)
## S4 method for signature 'SpatialPolygons,Raster'
rasterize(x, y, field, fun='last', background=NA,
  mask=FALSE, update=FALSE, updateValue='all', filename="",
  getCover=FALSE, silent=TRUE, ...)
```

Arguments

- **x** points (a SpatialPoints* object, or a two-column matrix (or data.frame)), SpatialLines*, SpatialPolygons*, or an Extent object
- **y** Raster* object
- **field** numeric or character. The value(s) to be transferred. This can be a single number, or a vector of numbers that has the same length as the number of spatial features (points, lines, polygons). If `x` is a Spatial*DataFrame, this can be the column name of the variable to be transferred. If missing, the attribute index is used (i.e. numbers from 1 to the number of features). You can also provide a vector with the same length as the number of spatial features, or a matrix where the number of rows matches the number of spatial features
- **fun** function or character. To determine what values to assign to cells that are covered by multiple spatial features. You can use functions such as `min`, `max`, or `mean`, or one of the following character values: 'first', 'last', 'count'. The default value is 'last'. In the case of SpatialLines*, 'length' is also allowed (currently for planar coordinate systems only).
  - If `x` represents points, `fun` must accept a `na.rm` argument, either explicitly or through 'dots'. This means that `fun=length` fails, but `fun=function(x,...) length(x)` works, although it ignores the `na.rm` argument. To use the `na.rm` argument you can use a function like this: `fun=function(x, na.rm)if (na.rm) length(na.omit(x)) else (length(x),` or use a function that removes NA values in all cases, like this function to compute the number of unique values per grid cell "richness": `fun=function(x, ...) {length(unique(na.omit(x))`)
  - If you want to count the number of points in each grid cell, you can use `fun='count'` or `fun=function(x,...){length(x)}`.
  - You can also pass multiple functions using a statement like `fun=function(x, ...) c(length(x),mean(x))`, in which case the returned object is a RasterBrick (multiple layers).
- **background** numeric. Value to put in the cells that are not covered by any of the features of `x`. Default is NA
mask logical. If TRUE the values of the input Raster object are 'masked' by the spatial features of x. That is, cells that spatially overlap with the spatial features retain their values, the other cells become NA. Default is FALSE. This option cannot be used when update=TRUE.

update logical. If TRUE, the values of the Raster* object are updated for the cells that overlap the spatial features of x. Default is FALSE. Cannot be used when mask=TRUE.

updateValue numeric (normally an integer), or character. Only relevant when update=TRUE. Select, by their values, the cells to be updated with the values of the spatial features. Valid character values are 'all', 'NA', and '!NA'. Default is 'all'.

filename character. Output filename (optional)

na.rm If TRUE, NA values are removed if fun honors the na.rm argument

gCover logical. If TRUE, the fraction of each grid cell that is covered by the polygons is returned (and the values of field, fun, mask, and update are ignored. The fraction covered is estimated by dividing each cell into 100 subcells and determining presence/absence of the polygon in the center of each subcell.

silent Logical. If TRUE, feedback on the polygon count is suppressed. Default is FALSE.

Additional arguments for file writing as for writeRaster

Value

RasterLayer or RasterBrick

See Also

extract

Examples

# rasterize points
r <- raster(ncols=36, nrows=18)
n <- 1000
set.seed(123)
x <- runif(n) * 360 - 180
y <- runif(n) * 180 - 90
xy <- cbind(x, y)
# get the (last) indices
r0 <- rasterize(xy, r)
# presence/absence (NA) (is there a point or not?)
r1 <- rasterize(xy, r, field=1)
# how many points?
r2 <- rasterize(xy, r, fun=function(x,...)length(x))
vals <- runif(n)
# sum of the values associated with the points
r3 <- rasterize(xy, r, vals, fun=sum)
```r
# with a SpatialPointsDataFrame
vals <- 1:n
p <- data.frame(xy, name=vals)
coordinates(p) <- ~x+y
r <- rasterize(p, r, 'name', fun=min)
#r2 <- rasterize(p, r, 'name', fun=max)
#plot(r, r2, cex=0.5)

# rasterizes lines

cds1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60))
cds2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
cds3 <- rbind(c(-125,0), c(0,0), c(40,5), c(15,-45))

lines <- spLines(cds1, cds2, cds3)
r <- raster(ncols=90, nrows=45)
r <- rasterize(lines, r)

## Not run:
plot(r)
plot(lines, add=TRUE)

r <- rasterize(lines, r, fun='count')
plot(r)

r[] <- 1:ncell(r)
r <- rasterize(lines, r, mask=TRUE)
plot(r)

r[] <- 1
r[lines] <- 10
plot(r)

## End(Not run)

# rasterize polygons

p1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60), c(-180,-20))
hole <- rbind(c(-150,-20), c(-100,-10), c(-110,20), c(-150,-20))
p1 <- list(p1, hole)
p2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55), c(-10,0))
p3 <- rbind(c(-125,0), c(0,0), c(40,5), c(15,-45), c(-125,0))
pols <- spPolygons(p1, p2, p3)
r <- raster(ncol=90, nrow=45)
r <- rasterize(pols, r, fun=sum)
```
## Not run:

```r
plot(r)
plot(pols, add=T)

# add a polygon
p5 <- rbind(c(-180, 10), c(0, 90), c(40, 90), c(145, -10),
            c(-25, -15), c(-180, 0), c(-180, 10))
addpoly <- SpatialPolygons(list(Polygons(list Polygon(p5), 1)))
addpoly <- as(addpoly, "SpatialPolygonsDataFrame")
addpoly@data[1, 1] <- 10
r2 <- rasterize(addpoly, r, field=1, update=TRUE, updateValue="NA")
plot(r2)
plot(pols, border="blue", lwd=2, add=TRUE)
plot(addpoly, add=TRUE, border="red", lwd=2)

# get the percentage cover of polygons in a cell
r3 <- raster(ncol=36, nrow=18)
r3 <- rasterize(pols, r3, getCover=TRUE)
```

## End(Not run)

### Description

Functions in the raster package create temporary files if the values of an output RasterLayer cannot be stored in memory (RAM). This can happen when no filename is provided to a function and in functions where you cannot provide a filename (e.g. when using 'raster algebra').

Temporary files are automatically removed at the start of each session. During a session you can use showTmpFiles to see what is there and removeTmpFiles to delete all the temporary files. `rastertmpfile` returns a temporary filename. These can be useful when developing your own functions. These filenames consist of `prefix_date_time_pid_rn` where `pid` is the process id returned by `Sys.getpid` and `rn` is a 5 digit random number. This should make tempfiles unique if created at different times and also when created in parallel processes (different pid) that use `set.seed` and call `rastertmpfile` at the same time. It is possible, however, to create overlapping names (see the examples), which is undesirable and can be avoided by setting the prefix argument.

### Usage

```r
rastertmpfile(prefix='r_tmp_')
showTmpFiles()
removeTmpFiles(h=24)
```

### Arguments

- **prefix**: Character. Prefix to the filename (which will be followed by 10 random numbers)
h Numeric. The minimum age of the files in number of hours (younger files are not deleted)

Details

The default path where the temporary files are stored is returned (can be changed with `rasterOptions`).

Value

- `rasterTmpFile` returns a valid file name
- `showTmpFiles` returns the names (.grd only) of the files in the temp directory
- `removeTmpFiles` returns nothing

See Also

- `rasterOptions`, `tempfile`

Examples

```r
## Not run:
rasterTmpFile('mytemp_')
showTmpFiles()
removeTmpFiles(h=24)

## End(Not run)
```

---

### rasterToContour

**Raster to contour lines conversion**

Description

RasterLayer to contour lines. This is a wrapper around `contourLines`.

Usage

```r
rasterToContour(x, maxpixels=100000, ...)
```

Arguments

- **x** a RasterLayer object
- **maxpixels** Maximum number of raster cells to use; this function fails when too many cells are used
- **...** Any argument that can be passed to `contourLines`

Details

Most of the code was taken from maptools::ContourLines2SLDF, by Roger Bivand & Edzer Pebesma.
**Value**

SpatialLinesDataFrame

**Examples**

```r
f <- system.file("external/test.grd", package="raster")
r <- raster(f)
x <- rasterToContour(r)
class(x)
plot(r)
plot(x, add=TRUE)
```

---

**rasterToPoints**  
*Raster to points conversion*

**Description**

Raster to point conversion. Cells with NA are not converted. A function can be used to select a subset of the raster cells (by their values).

**Usage**

```r
rasterToPoints(x, fun=NULL, spatial=FALSE, ...)
```

**Arguments**

- `x` A Raster* object
- `fun` Function to select a subset of raster values
- `spatial` Logical. If TRUE, the function returns a SpatialPointsDataFrame object
- `...` Additional arguments. Currently only `progress` to specify a progress bar. "text", "window", or "" (the default, no progress bar)

**Details**

`fun` should be a simple function returning a logical value.

E.g.: `fun=function(x){x==1}` or `fun=function(x){x>3}`

**Value**

A matrix with three columns: x, y, and v (value), or a SpatialPointsDataFrame object
rasterToPolygons

rasterToPolygons

Examples

r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r)) * 10
r[r>8] <- NA
p <- rasterToPoints(r)
p <- rasterToPoints(r, fun=function(x){x>6})
#plot(r)
#points(p)

rasterToPolygons

Raster to polygons conversion

Description

Raster to polygons conversion. Cells with NA are not converted. A function can be used to select a subset of the raster cells (by their values).

Usage

rasterToPolygons(x, fun=NULL, n=4, na.rm=TRUE, digits=12, dissolve=FALSE)

Arguments

x
fun
n
na.rm
digits
dissolve

Raster* object
function to select a subset of raster values (only allowed if x has a single layer)
integer. The number of nodes for each polygon. Only 4, 8, and 16 are allowed
If TRUE, cells with NA values in all layers are ignored
number of digits to round the coordinates to
logical. If TRUE, polygons with the same attribute value will be dissolved into multi-polygon regions. This option requires the rgeos package

Details

fun should be a simple function returning a logical value.
E.g.: fun=function(x){x==1} or fun=function(x){x>3 & x<6}

Value

SpatialPolygonsDataFrame
Examples

```r
r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r)) * 10
r[r>8] <- NA
pol <- rasterToPolygons(r, fun=function(x){x>6})

#plot(r > 6)
#plot(pol, add=TRUE, col='red')
```

Description

These classes are for internal use only

readAll

Read values from disk

Usage

```r
readAll(object)
```

Arguments

- `object`: a Raster* object

See Also

- `getValues`
- `getValuesBlock`
- `extract`

Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))
r <- readAll(r)
```
Description

Reclassify values of a Raster* object. The function (re)classifies groups of values to other values. For example, all values between 1 and 10 become 1, and all values between 11 and 15 become 2 (see functions subs and cut for alternative approaches).

Reclassification is done with matrix rcl, in the row order of the reclassify table. Thus, if there are overlapping ranges, the first time a number is within a range determines the reclassification value.

Usage

```r
## S4 method for signature 'Raster'
reclassify(x, rcl, filename='', include.lowest=FALSE, right=TRUE, ...)
```

Arguments

- **x**: Raster* object
- **rcl**: matrix for reclassification. This matrix must have 3 columns. The first two columns are "from" "to" of the input values, and the third column "becomes" has the new value for that range. (You can also supply a vector that can be coerced into a n*3 matrix (with byrow=TRUE)). You can also provide a two column matrix ("is", "becomes") which can be useful for integer values. In that case, the right argument is automatically set to NA
- **filename**: character. Output filename (optional)
- **include.lowest**: logical, indicating if a value equal to the lowest value in rcl (or highest value in the second column, for right = FALSE) should be included. The default is FALSE
- **right**: logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa. The default is TRUE. A special case is to use right=NA. In this case both the left and right intervals are open
- ...: additional arguments as for writeRaster

Value

Raster* object

See Also

subs, clamp, cut, calc
**Examples**

```r
r <- raster(ncols=36, nrows=18)
r[] <- runif(ncell(r))
# reclassify the values into three groups
# all values >= 0 and <= 0.25 become 1, etc.
m <- c(0, 0.25, 1, 0.25, 0.5, 2, 0.5, 1, 3)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rcl <- reclassify(r, rclmat)

# equivalent to
rc <- reclassify(r, c(-Inf,0.25,1, 0.25,0.5,2, 0.5,Inf,3))
```

---

**rectify**

**rectify a Raster object**

---

**Description**

rectify changes a rotated Raster* object into a non-rotated (rectangular) object. This is wrapper function around `resample`.

**Usage**

```r
rectify(x, ext, res, method='ngb', filename='', ...)
```

**Arguments**

- `x` Raster* object to be rectified
- `ext` Optional. Extent object or object from which an Extent object can be extracted
- `res` Optional. Single or two numbers to set the resolution
- `method` Method used to compute values for the new RasterLayer, should be "bilinear" for bilinear interpolation, or "ngb" for nearest neighbor
- `filename` Character. Output filename
- `...` Additional arguments as for `writeRaster`

**Value**

RasterLayer or RasterBrick object
replacement

Replace cell values or layers of a Raster* object

Description

You can set values of a Raster* object, when i is a vector of cell numbers, a Raster*, Extent, or Spatial* object.

These are shorthand methods that work best for relatively small Raster* objects. In other cases you can use functions such as calc and rasterize.

Methods

\[ x[i] \leftarrow \text{value} \]
\[ x[i,j] \leftarrow \text{value} \]

Arguments:
- \( x \): a Raster* object
- \( i \): cell number(s), row number(s), Extent, Spatial* object
- \( j \): columns number(s) (only available if \( i \) is (are) a row number(s))
- \( \text{value} \): new cell value(s)

See Also

calc, rasterize

Examples

\[
\begin{align*}
  r & \leftarrow \text{raster(ncol=10, nrow=5)} \\
  r[] & \leftarrow 1:n\text{cell}(r) * 2 \\
  r[1,] & \leftarrow 1 \\
  r[,1] & \leftarrow 2 \\
  r[1,1] & \leftarrow 3 \\
  s & \leftarrow \text{stack}(r, \sqrt{r}) \\
  s[s<5] & \leftarrow \text{NA}
\end{align*}
\]

resample

Resample a Raster object

Description

Resample transfers values between non matching Raster* objects (in terms of origin and resolution). Use projectRaster if the target has a different coordinate reference system (projection).

Before using resample, you may want to consider using these other functions instead: aggregate, disaggregate, crop, extend, merge.
Usage

```r
## S4 method for signature 'Raster,Raster'
resample(x, y, method="bilinear", filename="", ...)```

Arguments

- `x`: Raster* object to be resampled
- `y`: Raster* object with parameters that `x` should be resampled to
- `method`: method used to compute values for the new RasterLayer, should be "bilinear" for bilinear interpolation, or "ngb" for using the nearest neighbor
- `filename`: character. Output filename (optional)
- `...`: Additional arguments as for `writeRaster`

Value

RasterLayer or RasterBrick object

Author(s)

Robert J. Hijmans and Joe Cheng

See Also

aggregate, disaggregate, crop, extend, merge, projectRaster

Examples

```r
r <- raster(nrow=3, ncol=3)
r[] <- 1:ncell(r)
s <- raster(nrow=10, ncol=10)
s <- resample(r, s, method="bilinear")
#par(mfrow=c(1,2))
#plot(r)
#plot(s)
```

---

### Description

Get (or set) the x and/or y resolution of a Raster* object

Usage

```r
xres(x)
yres(x)
res(x)
res(x) <- value```
Arguments

- **x**: Raster* object
- **value**: Resolution (single number or vector of two numbers)

**Value**

A single numeric value or two numeric values.

**See Also**

`extent`, `ncell`

**Examples**

```r
r <- raster(ncol=18, nrow=18)
xres(r)
yres(r)
res(r)

res(r) <- 1/120
# set yres differently
res(r) <- c(1/120, 1/60)
```

---

**RGB**

*Create a Red-Green-Blue Raster object*

**Description**

Make a Red-Green-Blue object that can be used to create images.

**Usage**

```r
## S4 method for signature 'RasterLayer'
RGB(x, filename='', col=rainbow(25), breaks=NULL, alpha=FALSE,
    colNA='white', zlim=NULL, zlimcol=NULL, ext=NULL, ...)
```

**Arguments**

- **x**: RasterBrick or RasterStack
- **filename**: character. Output filename (optional)
- **col**: A color palette, that is a vector of n contiguous colors generated by functions like `rainbow`, `heat.colors`, `topo.colors`, `bpy.colors` or one of your own making, perhaps using `colorRampPalette`. If none is provided, `rev(terrain.colors(255))` is used unless `x` has a `color table`
breaks numeric. A set of finite numeric breakpoints for the colours: must have one more breakpoint than colour and be in increasing order
alpha If TRUE a fourth layer to set the background transparency is added
colNA color for the background (NA values)
zlim vector of length 2. Range of values to plot
zlimcol If NULL the values outside the range of zlim get the color of the extremes of the range. If zlimcol has any other value, the values outside the zlim range get the color of NA values (see colNA)
ext An Extent object to zoom in to a region of interest (see drawExtent)
... additional arguments as for writeRaster

See Also
plotRGB

Examples
r <- raster(system.file("external/test.grd", package="raster"))
x <- RGB(r)
plot(x, col=gray(0:9/10))
plotRGB(x)

rotate Rotate

Description
Rotate a Raster* object that has x coordinates (longitude) from 0 to 360, to standard coordinates between -180 and 180 degrees. Longitude between 0 and 360 is frequently used in global climate models.

Usage
## S4 method for signature 'Raster'
rotate(x, filename='', ...)

Arguments
x Raster* object
filename character. Output filename (optional)
... additional arguments as for writeRaster

Value
RasterLayer or a RasterBrick object
See Also

flip

Examples

r <- raster(nrow=18, ncol=36)
m <- matrix(1:ncell(r), nrow=18)
r[] <- as.vector(t(m))
extent(r) <- extent(0, 360, -90, 90)
rr <- rotate(r)

---

rotated

Do the raster cells have a rotation?

Description

Do the raster cells have a rotation?

Usage

rotated(x)

Arguments

x A Raster* object

Value

Logical value

See Also

rectify

Examples

r <- raster()
rotated(r)
round

Description

These functions take a single RasterLayer argument \( x \) and change its values to integers.

- **ceiling**: returns a RasterLayer with the smallest integers not less than the corresponding values of \( x \).
- **floor**: returns a RasterLayer with the largest integers not greater than the corresponding values of \( x \).
- **trunc**: returns a RasterLayer with the integers formed by truncating the values in \( x \) toward 0.
- **round**: returns a RasterLayer with values rounded to the specified number of digits (decimal places; default 0).

Details

see ?base::round

Value

a RasterLayer object

Methods

- **ceiling**\( (x) \)
- **floor**\( (x) \)
- **trunc**\( (x, \ldots) \)
- **round**\( (x, \text{digits}=0) \)

- a RasterLayer object

- **digits**: integer indicating the precision to be used

- ... additional arguments

See Also

- **round**

Examples

```r
r <- raster(ncol=10, nrow=10)
r[] <- runif(ncell(r)) * 10
s <- round(r)
```
rowFromCell  

**Row or column number from a cell number**

**Description**

These functions get the row and/or column number from a cell number of a Raster* object.

**Usage**

```
colFromCell(object, cell)
rowFromCell(object, cell)
rowColFromCell(object, cell)
```

**Arguments**

- `object`: Raster* object (or a SpatialPixels* or SpatialGrid* object)
- `cell`: cell number(s)

**Details**

The `colFromCell` and similar functions accept a single value, or a vector or list of these values. Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the Raster* object.

**Value**

row of column number(s)

**See Also**

- `cellFrom`

**Examples**

```
r <- raster(ncols=10, nrows=10)
colFromCell(r, c(5,15))
rowFromCell(r, c(5,15))
rowColFromCell(r, c(5,15))
```
rowSums and colSums for Raster objects

Description

Sum values of Raster objects by row or column.

Usage

```r
## S4 method for signature 'Raster'
rowSums(x, na.rm=FALSE, dims=1L,...)
## S4 method for signature 'Raster'
colSums(x, na.rm=FALSE, dims=1L,...)
```

Arguments

- `x` : Raster* object
- `na.rm` : logical. If TRUE, NA values are ignored
- `dims` : this argument is ignored
- `...` : additional arguments (none implemented)

Value

vector (if `x` is a RasterLayer) or matrix

See Also

See `cellStats` for summing all cells values

Examples

```r
r <- raster(ncols=2, nrows=5)
values(r) <- 1:10
as.matrix(r)
rowSums(r)
colSums(r)
```
SampleInt  

Sample integer values

Description

Take a random sample from a range of integer values between 1 and n. Its purpose is similar to that of `sample`, but that function fails when n is very large.

Usage

```r
sampleInt(n, size, replace=FALSE)
```

Arguments

- **n**: Positive number (integer); the number of items to choose from
- **size**: Non-negative integer; the number of items to choose
- **replace**: Logical. Should sampling be with replacement?

Value

vector of integer numbers

Examples

```r
sampleInt(1e12, 10)
```

# this may fail:
# sample.int(1e12, 10)
# sample.int(1e9, 10)

sampleRandom  

Random sample

Description

Take a random sample from the cell values of a Raster* object (without replacement).

Usage

```r
## S4 method for signature 'Raster'
sampleRandom(x, size, na.rm=TRUE, ext=NULL,
             cells=FALSE, rowcol=FALSE, xy=FALSE, sp=FALSE, asRaster=FALSE, ...)
```
Arguments

- **x**: Raster* object
- **size**: positive integer giving the number of items to choose
- **na.rm**: logical. If TRUE (the default), NA values are removed from random sample
- **ext**: Extent object. To limit regular sampling to the area within the extent
- **cells**: logical. If TRUE, sampled cell numbers are also returned
- **rowcol**: logical. If TRUE, sampled row and column numbers are also returned
- **xy**: logical. If TRUE, coordinates of sampled cells are also returned
- **sp**: logical. If TRUE, a SpatialPointsDataFrame is returned
- **asRaster**: logical. If TRUE, a Raster* object is returned with random cells with values, all other cells with NA
- **...**: Additional arguments as in `writeRaster`. Only relevant when `asRaster=TRUE`

Details

With argument `na.rm=TRUE`, the returned sample may be smaller than requested

Value

A vector, matrix (if `cells=TRUE` or `x` is a multi-layered object), or a SpatialPointsDataFrame (if `sp=TRUE`)

See Also

`sampleRegular`, `sampleStratified`

Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))
sampleRandom(r, size=10)
s <- stack(r, r)
sampleRandom(s, size=5, cells=TRUE, sp=TRUE)
```

Description

Take a systematic sample from a Raster* object.

Usage

```r
## S4 method for signature 'Raster'
sampleRegular(x, size, ext=NULL, cells=FALSE, xy=FALSE, asRaster=FALSE, sp=FALSE, useGDAL=FALSE, ...)
```
Arguments

- **x**: Raster object
- **size**: positive integer giving the number of items to choose.
- **ext**: Extent. To limit regular sampling to the area within that box
- **cells**: logical. Also return sampled cell numbers (if asRaster=FALSE)
- **xy**: logical. If TRUE, coordinates of sampled cells are also returned
- **asRaster**: logical. If TRUE, a RasterLayer or RasterBrick is returned, rather than the sampled values
- **sp**: logical. If TRUE, a SpatialPointsDataFrame is returned
- **useGDAL**: logical. If TRUE, GDAL is used to sample in some cases. This is quicker, but can result in values for a different set of cells than when useGDAL=FALSE. Only for rasters that are accessed via rgdal, and are not rotated. When TRUE arguments cells, xy, and sp are ignored (i.e., FALSE

... additional arguments. None implemented

Value

A vector (single layer object), matrix (multi-layered object; or if cells=TRUE, or xy=TRUE), Raster* object (if asRaster=TRUE), or SpatialPointsDataFrame (if sp=TRUE)

See Also

sampleRandom, sampleStratified

Examples

```R
r <- raster(system.file("external/test.grd", package="raster"))
v <- sampleRegular(r, size=100)
x <- sampleRegular(r, size=100, asRaster=TRUE)
```

Description

Take a stratified random sample from the cell values of a Raster* object (without replacement). An attempt is made to sample size cells from each stratum. The values in the RasterLayer x are rounded to integers; with each value representing a stratum.

Usage

```R
## S4 method for signature 'RasterLayer'
sampleStratified(x, size, exp=10, na.rm=TRUE, xy=FALSE, ext=NULL, sp=FALSE, ...)
```
Arguments

- **x**  
  Raster* object, with values (rounded to integers) representing strata

- **size**  
  Positive integer giving the number of items to choose

- **exp**  
  Numeric >= 1. 'Expansion factor' that is multiplied with size to get an initial sample. Can be increased when you get an insufficient number of samples for small strata

- **na.rm**  
  Logical. If TRUE (the default), NA values are removed from random sample

- **xy**  
  Logical. Return coordinates of cells rather than cell numbers

- **ext**  
  Extent object. To limit regular sampling to the area within the extent

- **sp**  
  Logical. If TRUE, a SpatialPointsDataFrame is returned

- **...**  
  Additional arguments. None implemented

Details

The function may not work well when the size (number of cells) of some strata is relatively small.

Value

Matrix of cell numbers (and optionally coordinates) by stratum

See Also

- `sampleRandom`, `sampleRegular`

Examples

```r
r <- raster(ncol=10, nrow=10)
names(r) <- 'stratum'
r[] <- round((runif(ncell(r))+0.5)*3)
sampleStratified(r, size=3)
```

Description

Center and/or scale raster data

Usage

```r
## S4 method for signature 'Raster'
scale(x, center=TRUE, scale=TRUE)
```
scalebar

Arguments

x Raster* object
center logical or numeric. If TRUE, centering is done by subtracting the layer means (omitting NAs), and if FALSE, no centering is done. If center is a numeric vector with length equal to the nlayers(x), then each layer of x has the corresponding value from center subtracted from it.
scale logical or numeric. If TRUE, scaling is done by dividing the (centered) layers of x by their standard deviations if center is TRUE, and the root mean square otherwise. If scale is FALSE, no scaling is done. If scale is a numeric vector with length equal to nlayers(x), each layer of x is divided by the corresponding value. Scaling is done after centering.

Value

Raster* object

See Also

scale

Examples

b <- brick(system.file("external/rlogo.grd", package="raster"))
bs <- scale(b)

Description

Add a scalebar to a plot

Usage

scalebar(d, xy = NULL, type = "line", divs = 2, below = "", lonlat = NULL, label, adj=c(0.5, -0.5), lwd = 2, ...)

Arguments

d distance covered by scalebar
xy x and y coordinate to place the plot. Can be NULL. Use xy=click() to make this interactive
type "line" or "bar"
divs Number of divisions for a bar type. 2 or 4
below Text to go below scalebar (e.g., "kilometers")
lonlat Logical or NULL. If logical, TRUE indicates if the plot is using longitude/latitude coordinates. If NULL this is guessed from the plot’s coordinates

adj adjustment for text placement

label Vector of three numbers to label the scale bar (beginning, midpoint, end)

lwd line width for the "line" type scalebar

... arguments to be passed to other methods

Value

None. Use for side effect of a scalebar added to a plot

Author(s)

Robert J. Hijmans; partly based on a function by Josh Gray

See Also

plot

Examples

f <- system.file("external/test.grd", package="raster")
r <- raster(f)
plot(r)
scalebar(1000)
scalebar(1000, xy=c(178000, 333500), type='bar', divs=4)

select Geometric subsetting

Description

Geometrically subset Raster* or Spatial* objects by drawing on a plot (map).

Usage

## S4 method for signature 'Raster'
select(x, use='rec', ...)

## S4 method for signature 'Spatial'
select(x, use='rec', draw=TRUE, col='cyan', size=2, ...)
Arguments

x         Raster*, SpatialPoints*, SpatialLines*, or SpatialPolygons*
use       character: 'rec' or 'pol'. To use a rectangle or a polygon for selecting
draw      logical. Add the selected features to the plot?
col       color to use to draw the selected features (when draw=TRUE)
size      integer > 0. Size to draw the selected features with (when draw=TRUE))
...       additional arguments. None implemented

Value

Raster* or Spatial* object

See Also

click, crop

Examples

## Not run:

# select a subset of a RasterLayer
r <- raster(nrow=10, ncol=10)
r[] <- 1:ncell(r)
plot(r)
s <- select(r) # now click on the map twice

# plot the selection on a new canvas:
x11()
plot(s)

# select a subset of a SpatialPolygons object
p1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60), c(-180,-20))
hole <- rbind(c(-150,-20), c(-100,-10), c(-110,20), c(-150,-20))
p2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55), c(-10,0))
p3 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45), c(-125,0))
pols <- SpatialPolygons( list( Polygons(list(Polygon(p1), Polygon(hole)), 1),
                                Polygons(list(Polygon(p2)), 2), Polygons(list(Polygon(p3)), 3)))
pols@polygons[[1]]@Polygons[[2]]@hole <- TRUE

plot(pols, col=rainbow(3))
ps <- select(pols) # now click on the map twice
ps

## End(Not run)
setExtent

Set the extent of a RasterLayer

Description

setExtent sets the extent of a Raster* object. Either by providing a new Extent object or by setting the extreme coordinates one by one.

Usage

```r
setExtent(x, ext, keepres=FALSE, snap=FALSE)
extent(x) <- value
```

Arguments

- `x`: A Raster* object
- `ext`: An object of class Extent (which you can create with `extent`, or an object that has an extent (e.g. a Raster* or Spatial* object))
- `keepres`: logical. If TRUE, the resolution of the cells will stay the same after adjusting the bounding box (by adjusting the number of rows and columns). If FALSE, the number of rows and columns will stay the same, and the resolution will be adjusted.
- `snap`: logical. If TRUE, the extent is adjusted so that the cells of the input and output RasterLayer are aligned
- `value`: An object of class Extent (which you can create with `extent`)

Value

a Raster* object

See Also

`extent`, `Extent-class`

Examples

```r
r <- raster()
bb <- extent(-10, 10, -20, 20)
extent(r) <- bb
r <- setExtent(r, bb, keepres=TRUE)
```
setMinMax

---

**setMinMax**

*Compute min and max values*

---

**Description**

The minimum and maximum value of a RasterLayer are computed (from a file on disk if necessary) and stored in the returned Raster* object.

**Usage**

```r
setMinMax(x)
```

**Arguments**

- `x`: A Raster* object

**Value**

A Raster* object

**See Also**

`getValues`

**Examples**

```r
r <- raster(system.file("external/test.grd", package="raster"))
r
r <- setMinMax(r)
r
```

---

**setValues**

*Set values of a Raster object*

---

**Description**

Assign (new) values to a Raster* object.
Usage

```r
## S4 method for signature 'RasterLayer'
setValues(x, values, ...)

## S4 method for signature 'RasterBrick'
setValues(x, values, layer=-1, ...)

## S4 method for signature 'RasterStack'
setValues(x, values, layer=-1, ...)

## S4 method for signature 'RasterLayerSparse'
setValues(x, values, index=NULL, ...)

values(x) <- value
```

Arguments

- `x`: A `Raster*`
- `values`: Cell values to associate with the `Raster*` object. There should be values for all cells
- `value`: Cell values to associate with the `Raster*` object. There should be values for all cells
- `layer`: Layer number (only relevant for `RasterBrick` and `RasterStack` objects). If missing, the values of all layers is set
- `index`: Cell numbers corresponding to the values
- `...`: Additional arguments (none implemented)

Value

```
a Raster* object
```

Note

While you can access the `values` slot of the objects directly, you would do that at your own peril because when setting values, multiple slots need to be changed; which is what `setValues` takes care of.

See Also

`replacement`

Examples

```r
r <- raster(ncol=10, nrow=10)
vals <- 1:ncell(r)
r <- setValues(r, vals)
# equivalent to
r[] <- vals
```
shapefile

Read or write a shapefile

Description

Reading and writing of "ESRI shapefile" format spatial data. Only the three vector types (points, lines, and polygons) can be stored in shapefiles. These are simple wrapper functions around readOGR and writeOGR (rgdal package). A shapefile should consist of at least four files: .shp (the geometry), .dbf (the attributes), .shx (the index that links the two, and .prj (the coordinate reference system). If the .prj file is missing, a warning is given. If any other file is missing an error occurs (although one could in principle recover the .shx from the .shp file). Additional files are ignored.

Usage

## S4 method for signature 'character'
shapefile(x, stringsAsFactors=FALSE, verbose=FALSE, warnPRJ=TRUE, ...)

## S4 method for signature 'Spatial'
shapefile(x, filename='', overwrite=FALSE, ...)

Arguments

x character (a file name, when reading a shapefile) or Spatial* object (when writing a shapefile)
filename character. Filename to write a shapefile
overwrite logical. Overwrite existing shapefile?
verbose logical. If TRUE, information about the file is printed
warnPRJ logical. If TRUE, a warning is given if there is no .prj file
stringsAsFactors logical. If TRUE, strings are converted to factors
... Additional arguments passed to rgdal functions readOGR or writeOGR

Value

Spatial*DataFrame (reading). Nothing is returned when writing a shapefile.

Examples

if (require(rgdal)) {

filename <- system.file("external/lux.shp", package="raster")
filename
p <- shapefile(filename)

## Not run:
Description

Shift the location of a Raster* of vector type Spatial* object in the x and/or y direction.

Usage

## S4 method for signature 'Raster'
shift(object, x=0, y=0, filename='', ...)

## S4 method for signature 'SpatialPolygons'
shift(object, x=0, y=0, ...)

## S4 method for signature 'SpatialLines'
shift(object, x=0, y=0, ...)

## S4 method for signature 'SpatialPoints'
shift(object, x=0, y=0, ...)

Arguments

- **object** Raster* or Spatial* object
- **x** numeric. The shift in horizontal direction
- **y** numeric. The shift in vertical direction
- **filename** character file name (optional)
- **...** if object is a Raster* object: additional arguments as for `writeRaster`

Value

Same object type as x

See Also

flip, rotate, and the elide function in the maptools package

Examples

```r
r <- raster()
r <- shift(r, x=1, y=-1)
```
Description

This is a deprecated function. Use terrain instead.

Usage

slopeAspect(dem, filename='', out=c('slope', 'aspect'), unit='radians',
neighbors=8, flatAspect, ...)

Arguments

dem  RasterLayer object with elevation values in map units, or in meters when the crs is longitude/latitude
filename  Character. Filename. optional
out  Character vector containing one or more of these options: 'slope', 'aspect'
unit  Character. 'degrees' or 'radians'
neighbors  Integer. Indicating how many neighboring cells to use to compute slope for any cell. Either 8 (queen case) or 4 (rook case), see Details
flatAspect  Numeric or NA. What value to use for aspect when slope is zero (and hence the aspect is undefined)? The default value is 90 degrees (or 0.5*pi radians)
...  Standard additional arguments for writing RasterLayer files

See Also

terrain

sp  Create SpatialLines* or SpatialPolygons*

Description

Helper functions to simplify the creation of SpatialLines* or SpatialPolygons* objects from coordinates.

Usage

spLines(x, ..., attr=NULL, crs=NA)
spPolygons(x, ..., attr=NULL, crs=NA)
Arguments

- **x**: matrix of list with matrices. Each matrix must have two columns with x and y coordinates (or longitude and latitude, in that order). Multi-line or multi-polygon objects can be formed by combining matrices in a list
- **...**: additional matrices and/or lists with matrices
- **attr**: data.frame with the attributes to create a *DataFrame object. The number of rows must match the number of lines/polygons
- **crs**: the coordinate reference system (PROJ4 notation)

Value

- SpatialLines* or SpatialPolygons*

Examples

```r
x1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60))
x2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
x3 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))
x4 <- rbind(c(41,-41.5), c(51,-35), c(62,-41), c(51,-50))
a <- spLines(x1, x2, x3)
b <- spLines(x1, list(x2, x3), attr=data.frame(id=1:2), crs='+proj=longlat +datum=WGS84')
hole <- rbind(c(-150,-20), c(-100,-10), c(-110,20))
d <- spPolygons(list(x1,hole), x2, list(x3, x4))
att <- data.frame(ID=1:3, name=c('a', 'b', 'c'))
e <- spPolygons(list(x1,hole), x2, list(x3, x4), attr=att, crs='+proj=longlat +datum=WGS84')
```

spplot

Use spplot to plot a Raster* object

Description

A wrapper function around spplot (sp package). With spplot it is easy to map several layers with a single legend for all maps. spplot is itself a wrapper around the levelplot function in the lattice package, and see the help for these functions for additional options.

One of the advantages of these wrapper functions is the additional maxpixels argument to sample large Raster objects for faster drawing.
Methods

spplotHobj, ..., maxpixels=50000, as.table=TRUE)

obj A Raster* object
...
Any argument that can be passed to \texttt{spplot} and \texttt{levelplot}
maxpixels Integer. Number of pixels to sample from each layer of large Raster objects

See Also

\texttt{plot, plotRGB}

The rasterVis package has more advanced plotting methods for Raster objects

Examples

r <- raster(system.file("external/test.grd", package="raster"))
s <- stack(r, r*2)
names(s) <- c(‘meuse’, ‘meuse x 2’)

spplot(s)

ts <- data.frame(sampleRandom(r, 10, xy=TRUE))
coordinates(ts) <- ~ x + y

spplot(s, scales = list(draw = TRUE),
xlab = "easting", ylab = "northing",
col.regions = rainbow(99, start=.1),
names.attr=c(‘original’, ‘times two’),
sp.layout = list("sp.points", pts, pch=20, cex=2, col='black'),
par.settings = list(fontsize = list(text = 12)), at = seq(0, 4000, 500))

stack

\textit{Create a RasterStack object}

Description

A RasterStack is a collection of RasterLayer objects with the same spatial extent and resolution. A RasterStack can be created from RasterLayer objects, or from raster files, or both. It can also be created from a SpatialPixelsDataFrame or a SpatialGridDataFrame object.

Usage

\# S4 method for signature 'character'
stack(x, ..., bands=NULL, varname="", native=FALSE, RAT=TRUE, quick=FALSE)

\# S4 method for signature 'Raster'
stack(x, ..., layers=NULL)

## S4 method for signature 'missing'
stack(x)

## S4 method for signature 'list'
stack(x, bands=NULL, native=FALSE, RAT=TRUE, ...)

Arguments

- `x`: filename (character), Raster* object, missing (to create an empty RasterStack), SpatialGrid*, SpatialPixels*, or list (of filenames and/or Raster* objects). If x is a list, additional arguments ... are ignored
- `bands`: integer. which bands (layers) of the file should be used (default is all layers)
- `layers`: integer (or character with layer names) indicating which layers of a RasterBrick should be used (default is all layers)
- `native`: logical. If TRUE native drivers are used instead of gdal drivers (where available, such as for BIL and Arc-ASCII files)
- `RAT`: logical. If TRUE a raster attribute table is created for files that have one
- `quick`: logical. If TRUE the extent and resolution of the objects are not compared. This speeds up the creation of the RasteStack but should be use with great caution. Only use this option when you are absolutely sure that all the data in all the files are aligned, and you need to create RasterStack for many (>100) files
- `varname`: character. To select the variable of interest in a NetCDF file (see raster
- `...`: additional filenames or Raster* objects

Value

RasterStack

See Also

- addLayer, dropLayer, raster, brick

Examples

```r
# file with one layer
fn <- system.file("external/test.grd", package="raster")
s <- stack(fn, fn)
r <- raster(fn)
s <- stack(r, fn)
nlayers(s)

# file with three layers
slogo <- stack(system.file("external/rlogo.grd", package="raster"))
nlayers(slogo)
slogo
```
**Description**

Apply a function on subsets of a RasterStack or RasterBrick. The layers to be combined are indicated with the vector `indices`. The function used should return a single value, and the number of layers in the output Raster* equals the number of unique values in `indices`. For example, if you have a RasterStack with 6 layers, you can use `indices=c(1,1,1,2,2,2)` and `fun=sum`. This will return a RasterBrick with two layers. The first layer is the sum of the first three layers in the input RasterStack, and the second layer is the sum of the last three layers in the input RasterStack. Indices are recycled such that `indices=c(1,2)` would also return a RasterBrick with two layers (one based on the odd layers (1,3,5), the other based on the even layers (2,4,6)).

See `calc` if you want to use a more efficient function that returns multiple layers based on _all_ layers in the Raster* object.

**Usage**

```
stackApply(x, indices, fun, filename='', na.rm=TRUE, ...)
```

**Arguments**

- `x`: Raster* object
- `indices`: integer. Vector of length `nLayers(x)` (shorter vectors are recycled) containing all integer values between 1 and the number of layers of the output Raster*
- `fun`: function that returns a single value, e.g. `mean` or `min`, and that takes a `na.rm` argument (or can pass through arguments via `...`)
- `na.rm`: logical. If `TRUE`, NA cells are removed from calculations
- `filename`: character. Optional output filename
- `...`: additional arguments as for `writeRaster`

**Value**

A new Raster* object, and in some cases the side effect of a new file on disk.

**See Also**

`calc`, `stackSelect`

**Examples**

```
r <- raster(ncol=10, nrow=10)
r[] <- 1:ncell(r)
s <- brick(r,r,r,r,r)
s <- s * 1:6
bl <- stackApply(s, indices=c(1,1,1,2,2,2), fun=sum)
```
Description

A RasterStack is a collection of RasterLayers with the same spatial extent and resolution. They can be created from RasterLayer objects, or from file names. These two functions allow you to save the references to raster files and recreate a rasterStack object later. They only work if the RasterStack points to layers that have their values on disk. The values are not saved, only the references to the files.

Usage

stackOpen(stackfile)
stackSave(x, filename)

Arguments

stackfile Filename for the RasterStack (to save it on disk)
x RasterStack object
filename File name

Details

When a RasterStack is saved to a file, only pointers (filenames) to raster datasets are saved, not the data. If the name or location of a raster file changes, the RasterStack becomes invalid.

Value

RasterStack object

See Also

writeRaster, stack, addLayer

Examples

define a RasterStack
b1
b2 <- stackApply(s, indices=c(1,2,3,1,2), fun=sum)
b2

stackSave

Save or open a RasterStack file

Description

A RasterStack is a collection of RasterLayers with the same spatial extent and resolution. They can be created from RasterLayer objects, or from file names. These two functions allow you to save the references to raster files and recreate a rasterStack object later. They only work if the RasterStack points to layers that have their values on disk. The values are not saved, only the references to the files.

Usage

stackOpen(stackfile)
stackSave(x, filename)

Arguments

stackfile Filename for the RasterStack (to save it on disk)
x RasterStack object
filename File name

Details

When a RasterStack is saved to a file, only pointers (filenames) to raster datasets are saved, not the data. If the name or location of a raster file changes, the RasterStack becomes invalid.

Value

RasterStack object

See Also

writeRaster, stack, addLayer

Examples

file <- system.file("external/test.grd", package="raster")
s <- stack(c(file, file))
s <- stackSave(s, "mystack")
# note that filename adds an extension .stk to a stackfile
## Not run:
s2 <- stackOpen("mystack.stk")
s2

## End(Not run)
**Description**

Use a Raster* object to select cell values from different layers in a multi-layer Raster* object. The object to select values \( y \) should have cell values between 1 and \( \text{nLayers}(x) \). The values of \( y \) are rounded.

See `extract` for extraction of values by cell, point, or otherwise.

**Usage**

```
## S4 method for signature 'RasterStack,Raster'
stackSelect(x, y, recycle=FALSE, type='index', filename='', ...)  
```

**Arguments**

- **x** : RasterStack or RasterBrick object
- **y** : Raster* object
- **recycle** : Logical. Recursively select values (default = FALSE. Only relevant if \( y \) has multiple layers. E.g. if \( x \) has 12 layers, and \( y \) has 4 layers, the indices of the \( y \) layers are used three times.
- **type** : Character. Only relevant when `recycle=TRUE`. Can be 'index' or 'truefalse'. If it is 'index', the cell values of \( y \) should represent layer numbers. If it is 'truefalse' layer numbers are indicated by 0 (not used, NA returned) and 1 (used)
- **filename** : Character. Output filename (optional)
- **...** : Additional arguments as for `writeRaster`

**Value**

Raster* object

**See Also**

`stackApply`, `extract`

**Examples**

```r
r <- raster(ncol=10, nrow=10)
r[] <- 1
s <- stack(r, r+2, r+5)
r[] <- round((runif(ncell(r)))*3)
x <- stackSelect(s, r)
```
Description

Linear strech of values in a Raster object

Usage

stretch(x, minv=0, maxv=255, minq=0, maxq=1, filename='')

Arguments

- **x**: Raster* object
- **minv**: numeric >= 0 and smaller than maxv. lower bound of streched value
- **maxv**: numeric <= 255 and larger than maxv. upper bound of streched value
- **minq**: numeric >= 0 and smaller than maxq. lower quitile bound of original value
- **maxq**: numeric <= 1 and larger than minq. upper quitile bound of original value
- **filename**: character. Filename for the output Raster object (optional)
- **...**: additional arguments as for writeRaster

Value

Raster* object

See Also

stretch argument in plotRGB

Examples

```r
r <- raster(nc=10, nr=10)
r[] <- 1:100 * 10
stretch(r)
s <- stack(r, r*2)
stretch(s)
```
subset

Subset layers in a Raster* object

Description

Extract a set of layers from a RasterStack or RasterBrick object.

Usage

```r
## S4 method for signature 'Raster'
subset(x, subset, drop=TRUE, filename='', ...)  
## S4 method for signature 'RasterStack'
subset(x, subset, drop=TRUE, filename='', ...)  
```

Arguments

- `x`: RasterBrick or RasterStack object
- `subset`: integer or character. Should indicate the layers (represented as integer or by their name)
- `drop`: If TRUE, a selection of a single layer will be returned as a RasterLayer
- `filename`: character. Output filename (optional)
- `...`: additional arguments as for `writeRaster`

Value

Raster* object

See Also

dropLayer

Examples

```r
s <- stack(system.file("external/rlogo.grd", package="raster"))
set <- subset(s, 2:3)

# Note that this is equivalent to
sel2 <- s[[2:3]]

# and in this particular case:
set3 <- droplayer(s, 1)

nlayers(s)
nlayers(set)
```
substitute

Substitute values in a Raster* object

Description

Substitute (replace) values in a Raster* object with values in a data.frame. The data.frame should have a column to identify the key (ID) to match with the cell values of the Raster* object, and one or more columns with replacement values. By default these are the first and second column but you can specify other columns with arguments by and which. It is possible to match one table to multiple layers, or to use multiple layers as a single key, but not both.

Usage

```r
## S4 method for signature 'Raster,data.frame'
subs(x, y, by=1, which=2, subsWithNA=TRUE, filename='', ...)  
```

Arguments

- `x`: Raster* object
- `y`: data.frame
- `by`: column number(s) or name(s) identifying the key (ID) to match rows in data.frame `y` to values of the Raster object
- `which`: column number or name that has the new (replacement) values
- `subsWithNA`: logical. If TRUE values that are not matched become NA. If FALSE, they retain their original value (which could also be NA). This latter option is handy when you want to replace only one or a few values. It cannot be used when `x` has multiple layers
- `filename`: character. Optional output filename
- `...`: additional arguments as for `writeRaster`

Details

You could obtain the same result with `reclassify`, but `subs` is more efficient for simple replacement. Use `reclassify` if you want to replace ranges of values with new values.

You can also replace values using a fitted model. E.g. fit a model to `glm` or `loess` and then call `predict`

Value

Raster object
Summary

See Also
classify, clamp, cut

Examples

```r
r <- raster(ncol=10, nrow=10)
r[] <- round(runif(ncell(r)) * 10)
df <- data.frame(id=2:8, v=c(10,10,11,11,12:14))
x <- subs(r, df)
x2 <- subs(r, df, subsWithNA=FALSE)

df$v2 <- df$v * 10
x3 <- subs(r, df, which=2:3)

s <- stack(r, r*3)
names(s) <- c('first', 'second')
x4 <- subs(s, df)
x5 <- subs(s, df, which=2:3)
```

Description

Summarize a Raster* object. A sample is used for very large files.

Usage

```r
## S4 method for signature 'RasterLayer'
summary(object, maxsamp=100000, ...)
```

Arguments

- **object**: Raster* object
- **maxsamp**: positive integer. Sample size used for large datasets
- **...**: additional arguments. None implemented

Value

matrix with (an estimate of) the median, minimum and maximum values, the first and third quartiles, and the number of cells with NA values

See Also
cellStats, link[raster]{quantile}
Description

The following summary methods are available for Raster* objects:

- mean, max, min, range, prod, sum, any, all

All methods take na.rm as an additional logical argument. Default is na.rm=FALSE. If TRUE, NA values are removed from calculations. These methods compute a summary statistic based on cell values of RasterLayers and the result of these methods is always a single RasterLayer (except for range, which returns a RasterBrick with two layers). See calc for functions not included here (e.g. median) or any other custom functions.

You can mix RasterLayer, RasterStack and RasterBrick objects with single numeric or logical values. However, because generic functions are used, the method applied is chosen based on the first argument: 'x'. This means that if r is a RasterLayer object, mean(r, 5) will work, but mean(5, r) will not work.

To summarize all cells within a single RasterLayer, see cellStats and maxValue and minValue.

Value

a RasterLayer

See Also

calc

Examples

```r
r1 <- raster(nrow=10, ncol=10)
r1 <- setValues(r1, runif(ncell(r1)))
r2 <- setValues(r1, runif(ncell(r1)))
r3 <- setValues(r1, runif(ncell(r1)))
r <- max(r1, r2, r3)
r <- range(r1, r2, r3, 1.2)

s <- stack(r1, r2, r3)
r <- mean(s, 2)
```
symdif

Symmetrical difference

Description
Symmetrical difference of SpatialPolygons* objects

Usage
```r
## S4 method for signature 'SpatialPolygons, SpatialPolygons'
symdif(x, y, ...)
```

Arguments
- `x`: SpatialPolygons* object
- `y`: SpatialPolygons* object
- `...`: Additional SpatialPolygons* object(s)

Value
SpatialPolygons*

Author(s)
Robert J. Hijmans

See Also
- `erase`

Examples
```r
# SpatialPolygons
if (require(rgdal) & require(rgdal)) {
p <- shapefile(system.file("external/lux.shp", package="raster"))
b <- as(extent(6, 6.4, 49.75, 50), 'SpatialPolygons')
projection(b) <- projection(p)
sd <- symdif(p, b)
plot(sd, col='red')
}
```
terrain

Terrain characteristics

Description

Compute slope, aspect and other terrain characteristics from a raster with elevation data. The elevation data should be in map units (typically meter) for projected (planar) raster data. They should be in meters when the coordinate reference system (CRS) is longitude/latitude.

Usage

terrain(x, opt='slope', unit='radians', neighbors=8, filename='', ...)

Arguments

- **x**  
  RasterLayer object with elevation values. Values should have the same unit as the map units, or in meters when the CRS is longitude/latitude

- **opt**  
  Character vector containing one or more of these options: slope, aspect, TPI, TRI, roughness, flowdir (see Details)

- **unit**  
  Character. 'degrees', 'radians' or 'tangent'. Only relevant for slope and aspect. If 'tangent' is selected that is used for slope, but for aspect 'degrees' is used (as 'tangent' has no meaning for aspect)

- **neighbors**  
  Integer. Indicating how many neighboring cells to use to compute slope for any cell. Either 8 (queen case) or 4 (rook case). Only used for slope and aspect, see Details

- **filename**  
  Character. Output filename (optional)

- **...**  
  Standard additional arguments for writing Raster* objects to file

Details

When neighbors=4, slope and aspect are computed according to Fleming and Hoffer (1979) and Ritter (1987). When neighbors=8, slope and aspect are computed according to Horn (1981). The Horn algorithm may be best for rough surfaces, and the Fleming and Hoffer algorithm may be better for smoother surfaces (Jones, 1997; Burrough and McDonnell, 1998). If slope = 0, aspect is set to 0.5*pi radians (or 90 degrees if unit='degrees'). When computing slope or aspect, the CRS (projection) of the RasterLayer x must be known (may not be NA), to be able to safely differentiate between planar and longitude/latitude data.

flowdir returns the 'flow direction' (of water), i.e. the direction of the greatest drop in elevation (or the smallest rise if all neighbors are higher). They are encoded as powers of 2 (0 to 7). The cell to the right of the focal cell 'x' is 1, the one below that is 2, and so on:

```
  32  64  128
 16  x   1
  8  4   2
```
If two cells have the same drop in elevation, a random cell is picked. That is not ideal as it may prevent the creation of connected flow networks. ArcGIS implements the approach of Greenlee (1987) and I might adopt that in the future.

The terrain indices are according to Wilson et al. (2007), as in gdaldem. TRI (Terrain Ruggedness Index) is the mean of the absolute differences between the value of a cell and the value of its 8 surrounding cells. TPI (Topographic Position Index) is the difference between the value of a cell and the mean value of its 8 surrounding cells. Roughness is the difference between the maximum and the minimum value of a cell and its 8 surrounding cells.

Such measures can also be computed with the `focal` function:

```r
f <- matrix(1, nrow=3, ncol=3)
TRI <- focal(x, w=f, fun=function(x, ...) sum(abs(x[-5]-x[5]))/8, pad=TRUE, padValue=NA)
TPI <- focal(x, w=f, fun=function(x, ...) x[5] - mean(x[-5]), pad=TRUE, padValue=NA)
rough <- focal(x, w=f, fun=function(x, ...) max(x) - min(x), pad=TRUE, padValue=NA, na.rm=TRUE)
```

---

**References**


---

**See Also**

`hillshade`

**Examples**

```r
# Not run:
elevation <- getData('alt', country='CHE')
x <- terrain(elevation, opt=c('slope', 'aspect'), unit='degrees')
plot(x)

# TPI for different neighborhood size:
tpiw <- function(x, w=5) {

```
text  

Add labels to a map

Description

Plots labels, that is a textual (rather than color) representation of values, on top an existing plot (map).

Usage

```r
# S4 method for signature 'RasterLayer'
text(x, labels, digits=0, fun=NULL, halo=FALSE, ...)
```

```r
# S4 method for signature 'RasterStackBrick'
text(x, labels, digits=0, fun=NULL, halo=FALSE, ...)
```

```r
# S4 method for signature 'SpatialPolygons'
text(x, labels, halo=FALSE, ...)
```

```r
# S4 method for signature 'SpatialPoints'
text(x, labels, halo=FALSE, ...)
```

Arguments

- `x`  
  Raster*, SpatialPoints* or SpatialPolygons* object

- `labels`  
  character. Optional. Vector of labels with \texttt{length(x)} or a variable name from \texttt{names(x)}

- `digits`  
  integer. how many digits should be used?

- `fun`  
  function to subset the values plotted (as in \texttt{rasterToPoints})

- `halo`  
  logical. If \texttt{TRUE} a 'halo' is printed around the text. If \texttt{TRUE}, additional arguments \texttt{hc=’white’} and \texttt{hw=0.1} can be modified to set the colour and width of the halo

- `...`  
  additional arguments to pass to graphics function \texttt{text}

See Also

- \texttt{text}
- \texttt{plot}
Examples

```r
r <- raster(nrows=4, ncols=4)
r <- setValues(r, 1:ncell(r))
plot(r)
text(r)

plot(r)
text(r, halo=TRUE, hc='blue', col='white', hw=0.2)

plot(r, col=bpy.colors(5))
text(r, fun=function(x)(x<5 | x>12), col=c('red', 'white'), vfont=c("sans serif", "bold"), cex=2)
```

Description
Transpose a Raster* object

Usage
`t(x)`

Arguments
`x` a Raster* object

Value
RasterLayer or RasterBrick

See Also
transpose: `flip`, `rotate`

Examples

```r
r <- raster(nrow=18, ncol=36)
r[] <- 1:ncell(r)
rt <- t(r)
```
Description

Trim (shrink) a Raster* object by removing outer rows and columns that all have the same value (e.g. NA).
Or remove the whitespace before or after a string of characters (or a matrix, or the character values in a data.frame).

Usage

```r
## S4 method for signature 'Raster'
trim(x, padding=0, values=NA, filename='', ...)
## S4 method for signature 'character'
trim(x, internal=FALSE, ...)
```

Arguments

- `x`: Raster* object or a character string
- `values`: numeric. Value(s) based on which a Raster* should be trimmed
- `padding`: integer. Number of outer rows/columns to keep
- `filename`: character. Optional output filename
- `internal`: logical. If TRUE, sequential internal spaces are replaced by a single space
- `...`: If `x` is a Raster* object: additional arguments as for `writeRaster`

Value

A RasterLayer or RasterBrick object (if `x` is a Raster* object) or a character string (if `x` is a character string).

Author(s)

Robert J. Hijmans and Jacob van Etten

Examples

```r
r <- raster(ncol=18, nrow=18)
r[39:49] <- 1
r[113:155] <- 2
r[200] <- 6
s <- trim(r)

trim(" hi folks ! ")
```
Description

Extent objects: Objects are combined into their union. See `crop` and `extend` to union a Raster object with an Extent object.

Two SpatialPolygons* objects. Overlapping polygons (between layers, not within layers) are intersected, other spatial objects are appended. Tabular attributes are joined. See `bind` if you want to combine polygons without intersection.

Single SpatialPolygons* object. Overlapping polygons are intersected. Original attributes are lost. New attributes allow for determining how many, and which, polygons overlapped.

Union for SpatialLines and SpatialPoints simply combines the two data sets; without any geometric intersections. This is equivalent to `bind`.

Usage

```r
## S4 method for signature 'Extent,Extent'
union(x, y)

## S4 method for signature 'SpatialPolygons,SpatialPolygons'
union(x, y)

## S4 method for signature 'SpatialPolygons,missing'
union(x, y)

## S4 method for signature 'SpatialLines,SpatialLines'
union(x, y)

## S4 method for signature 'SpatialPoints,SpatialPoints'
union(x, y)
```

Arguments

- `x`: Extent or SpatialPolygons* object
- `y`: Same as `x` or missing

Value

Extent or SpatialPolygons object

See Also

`intersect`, `extent`, `setExtent`

`merge` for merging a data.frame with attributes of Spatial objects and `+,SpatialPolygons,SpatialPolygons-method` for an algebraic notation
Examples

```r
e1 <- extent(-10, 10, -20, 20)
e2 <- extent(0, 20, -40, 5)
union(e1, e2)

#SpatialPolygons
if (require(rgdal) & require(rgeos)) {
p <- shapefile(system.file("external/lux.shp", package="raster"))
p0 <- aggregate(p)
b <- as(extent(6, 6.4, 49.75, 50), 'SpatialPolygons')
projection(b) <- projection(p)
u <- union(p0, b)
plot(u, col=2:4)
}
```

---

**unique**

**Unique values**

Description

This function returns the unique values in a RasterLayer, or the unique combinations of values in a multi-layer raster object.

Usage

```r
## S4 method for signature 'RasterLayer,missing'
unique(x, incomparables=FALSE, ...)

## S4 method for signature 'RasterStackBrick,missing'
unique(x, incomparables=FALSE, ...)
```

Arguments

- `x` : Raster object
- `incomparables` : ignored. Must be missing
- `...` : additional arguments. One implemented: progress, as in `writeRaster`

Value

vector or matrix

See Also

`unique`
**unstack**

**Examples**

```r
r <- raster(ncol=10, nrow=10)
r[] <- round(runif(ncell(r))*10)
unique(r)
unique(stack(r, round(r/2)))
```

**Description**

Create a list of RasterLayer objects from a RasterStack or RasterBrick

**Usage**

```r
unstack(x, ...)
```

**Arguments**

- `x`: a RasterStack object
- `...`: not used. further arguments passed to or from other methods

**Value**

A list of RasterLayer objects

**See Also**

`stack`

**Examples**

```r
file <- system.file("external/test.grd", package="raster")
s <- stack(file, file)
list1 <- unstack(s)
b <- brick(s)
list2 <- unstack(b)
```
update

Update raster cells of files (on disk)

Description

Update cell values of a file (i.e., cell values on disk) associated with a RasterLayer or RasterBrick. User beware: this function _will_ make changes to your file (first make a copy if you are not sure what you are doing).

Writing starts at a cell number cell. You can write a vector of values (in cell order), or a matrix. You can also provide a vector of cell numbers (of the same length as vector v) to update individual cells.

See `writeFormats` for supported formats.

Usage

```r
## S4 method for signature 'RasterLayer'
update(object, v, cell, ...)
## S4 method for signature 'RasterBrick'
update(object, v, cell, band, ...)
```

Arguments

- `object` RasterLayer or RasterBrick that is associated with a file
- `v` vector or matrix with new values
- `cell` cell from where to start writing. Or a vector of cell numbers if v is a vector of the same length.
- `band` band (layer) to update (for RasterBrick objects).
- `...` additional arguments. None implemented

Value

RasterLayer or RasterBrick

Examples

```r
# setting up an example RasterLayer with file
r <- raster(nrow=5, ncol=10)
r[] = 0
r <- writeRaster(r, 'test', overwrite=TRUE, datatype='INT2S')
as.matrix(r)

# update with a vector starting a cell
r <- update(r, v=rep(1, 5), cell=6)
# 99.99 gets rounded because this is an integer file
r <- update(r, v=9.99, cell=50)
as.matrix(r)
```
validCell

## validCell

**Validity of a cell, column or row number**

### Description

Simple helper functions to determine if a row, column or cell number is valid for a certain Raster* object.

### Usage

```r
validCell(object, cell)
validCol(object, colnr)
validRow(object, rownr)
```

### Arguments

- `object`: Raster* object (or a SpatialPixels* or SpatialGrid* object)
- `cell`: cell number(s)
- `colnr`: column number; or vector of column numbers
- `rownr`: row number; or vector of row numbers

### Value

logical value

### Examples

```r
# using a new default raster (1 degree global)
r <- raster()
validCell(r, c(-1, 0, 1))
validRow(r, c(-1, 1, 100, 10000))
```
validNames Create valid names

Description
Create a set of valid names (trimmed, no duplicates, not starting with a number).

Usage
validNames(x, prefix='layer')

Arguments
- x character
- prefix character string used if x is empty

Value
character

See Also
make.names

Examples
validNames(c('a', 'a', '', '1', NA, 'b', 'a'))

weighted.mean Weighted mean of rasters

Description
Computes the weighted mean for each cell of a number or raster layers. The weights can be spatially variable or not.

Usage
## S4 method for signature 'RasterStackBrick,vector'
weighted.mean(x, w, na.rm=FALSE, filename='', ...)

## S4 method for signature 'RasterStackBrick,RasterStackBrick'
weighted.mean(x, w, na.rm=FALSE, filename='', ...)
### which

Returns a RasterLayer with TRUE or FALSE setting cells that are NA to FALSE (unless na.rm=FALSE). If the RasterLayer has numbers, all values that are 0 become FALSE and all other values become TRUE. The function can also return the cell numbers that are TRUE.

#### Usage

```
## S4 method for signature 'RasterLayer'
Which(x, cells=FALSE, na.rm=TRUE, ...)
```
which.min

Arguments

- **x**  
  RasterLayer
- **cells**  
  logical. If TRUE, cell numbers are returned, otherwise a RasterLayer is returned
- **na.rm**  
  logical. If TRUE, NA values are treated as FALSE, otherwise they remain NA (only when cells=FALSE)
- ...  
  Additional arguments (none implemented)

Value

RasterLayer

See Also

which.max, which.min

Examples

```r
r <- raster(ncol=10, nrow=10)
set.seed(0)
r[] <- runif(ncell(r))
r[r < 0.2] <- 0
r[r > 0.8] <- 1
r[r > 0 & r < 1] <- 0.5

Which(r, cells=TRUE)
Which(r > 0.5, cells=TRUE)

s1 <- r > 0.5
s2 <- Which(r > 0.5)
s1[1:15]
s2[1:15]

# this expression
x1 <- Which(r, na.rm=FALSE)
# is the inverse of
x2 <- r==0
```

<table>
<thead>
<tr>
<th>which.min</th>
<th>Where is the min or max value?</th>
</tr>
</thead>
</table>

Description

Which cells have the minimum / maximum value (for a RasterLayer), or which layer has the minimum/maximum value (for a RasterStack or RasterBrick)?

which.min and which.max return the index of the first layer that has the min or max value for a cell. This can be problematic if there are ties.

In you want the index of all the layers that have the min or max value, use whiches.min or whiches.max (only for objects with less than 10 layers).
which.min

Usage

which.min(x)
which.max(x)
whiches.min(x, ...)
whiches.max(x, ...)

Arguments

x Raster* object
...
additional arguments (none implemented)

Value

(which.*): vector of cell numbers (if x is a RasterLayer). If x is a RasterStack or RasterBrick, a RasterLayer giving the number of the first layer with the minimum or maximum value for a cell.

(whiches.*) An integer in which each digit represents a layer. For example, 35 means "layers 3 and 5".

Note

There is a limit to accurate integer number representation. Therefore, do not use whiches.* with more than 15 layers.

See Also

Which

Examples

b <- brick(system.file("external/rlogo.grd", package="raster"))
r <- which.min(b)
i <- which.min(b[[3]])
xy <- xyFromCell(b, i)
plot(b[[3]])
points(xy)
x <- whiches.min(b)
freq(x)
**Description**

List supported file types for writing RasterLayer values to disk.

When a function writes a file to disk, the file format is determined by the 'format=' argument if supplied, or else by the file extension (if the extension is known). If other cases the default format is used. The 'factory-fresh' default format is 'raster', but this can be changed using `rasterOptions`.

**Usage**

```r
writeFormats()
```

**Details**

`writeFormats` returns a matrix of the file formats (the "drivers") that are supported.

Supported formats include:

<table>
<thead>
<tr>
<th>File type</th>
<th>Long name</th>
<th>default extension</th>
<th>Multiband support</th>
</tr>
</thead>
<tbody>
<tr>
<td>raster</td>
<td>'Native' raster package format</td>
<td>.grd</td>
<td>Yes</td>
</tr>
<tr>
<td>ascii</td>
<td>ESRI Ascii</td>
<td>.asc</td>
<td>No</td>
</tr>
<tr>
<td>SAGA</td>
<td>SAGA GIS</td>
<td>.sdat</td>
<td>No</td>
</tr>
<tr>
<td>IDRISI</td>
<td>IDRISI</td>
<td>.rst</td>
<td>No</td>
</tr>
<tr>
<td>CDF</td>
<td>netCDF (requires ncdf4)</td>
<td>.nc</td>
<td>Yes</td>
</tr>
<tr>
<td>GTiff</td>
<td>GeoTiff (requires rgdal)</td>
<td>.tif</td>
<td>Yes</td>
</tr>
<tr>
<td>ENVI</td>
<td>ENVI .hdr Labelled</td>
<td>.envi</td>
<td>Yes</td>
</tr>
<tr>
<td>EHdr</td>
<td>ESRI .hdr Labelled</td>
<td>.bil</td>
<td>Yes</td>
</tr>
<tr>
<td>HFA</td>
<td>Erdas Imagine Images (.img)</td>
<td>.img</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**See Also**

`GDALDriver-class`

**Examples**

```r
writeFormats()
```
writeRaster

Description

Write an entire Raster* object to a file, using one of the many supported formats. See writeValues for writing in chunks (e.g. by row).

When writing a file to disk, the file format is determined by the 'format=' argument if supplied, or else by the file extension (if the extension is known). If other cases the default format is used. The default format is 'raster', but this setting can be changed (see rasterOptions).

Usage

## S4 method for signature 'RasterLayer,character'
writeRaster(x, filename, format, ...)

## S4 method for signature 'RasterStackBrick,character'
writeRaster(x, filename, format, bylayer, suffix='numbers', ...)

Arguments

- x: Raster* object
- filename: Output filename
- format: Character. Output file type. See writeFormats. If this argument is not provided, it is attempted to infer it from the filename extension. If that fails, the default format is used. The default format is 'raster', but this can be changed using rasterOptions
- ...: Additional arguments:
  - datatype: Character. Output data type (e.g. 'INT2S' or 'FLT4S'). See dataType. If no datatype is specified, 'FLT4S' is used, unless this default value was changed with rasterOptions
  - overwrite: Logical. If TRUE, "filename" will be overwritten if it exists
  - progress: Character. Set a value to show a progress bar. Valid values are "text" and "window".
  - naflag: Numeric. To overwrite the default value used to represent NA in a file
  - bandorder: Character. 'BIL', 'BIP', or 'BSQ'. For 'native' file formats only. For some other formats you can use the 'options' argument (see below)
  - options: Character. File format specific GDAL options. E.g., when writing a geotiff file you can use: options=c("COMPRESS=NONE", "TFW=YES")
    You can use options=c("PROFILE=BASLINE") to create a plain tif with no GeoTIFF tags. This can be useful when writing files to be read by applications intolerant of unrecognised tags. (see http://www.gdal.org/frmt_gtiff.html)
  - NetCDF files have the following additional, optional, arguments: varname, varunit, longname, xname, yname, zname, zunit
  - prj: Logical. If TRUE, the crs is written to a .prj file. This can be useful when writing to an ascii file or another file type that does not store the crs
  - bylayer: if TRUE, write a separate file for each layer. You can provide a vector of filenames that matches the number of layers. Or you can provide a single filename that will get a unique suffix (see below)
suffix 'numbers' or 'names' to determine the suffix that each file gets when bylayer=TRUE; either a number between 1 and nlayers(x) or names(x)

Details

See writeFormats for supported file types ("formats", "drivers").

The rgdal package is needed, except for these file formats: 'raster', 'BIL', 'BIP', 'BSQ', 'SAGA', 'ascii', 'IDRISI', and 'CDF'. Some of these formats can be used with or without rgdal (idrisi, SAGA, ascii). You need the 'ncdf4' library for the 'CDF' format.

In multi-layer files (i.e. files saved from RasterStack or RasterBrick objects), in the native 'raster' format, the band-order can be set to BIL ('Bands Interleaved by Line'), BIP ('Bands Interleaved by Pixels') or BSQ ('Bands SeQuential'). Note that bandorder is not the same as filetype here.

Supported file types include:

<table>
<thead>
<tr>
<th>File type</th>
<th>Long name</th>
<th>default extension</th>
<th>Multiband support</th>
</tr>
</thead>
<tbody>
<tr>
<td>raster</td>
<td>'Native' raster package format</td>
<td>.grd</td>
<td>Yes</td>
</tr>
<tr>
<td>ascii</td>
<td>ESRI Ascii</td>
<td>.asc</td>
<td>No</td>
</tr>
<tr>
<td>SAGA</td>
<td>SAGA GIS</td>
<td>.sdat</td>
<td>No</td>
</tr>
<tr>
<td>IDRISI</td>
<td>IDRISI</td>
<td>.rst</td>
<td>No</td>
</tr>
<tr>
<td>CDF</td>
<td>netCDF (requires ncdf4)</td>
<td>.nc</td>
<td>Yes</td>
</tr>
<tr>
<td>GTiff</td>
<td>GeoTiff (requires rgdal)</td>
<td>.tif</td>
<td>Yes</td>
</tr>
<tr>
<td>ENVI</td>
<td>ENVI .hdr Labelled</td>
<td>.envi</td>
<td>Yes</td>
</tr>
<tr>
<td>Ehdr</td>
<td>ESRI .hdr Labelled</td>
<td>.bil</td>
<td>Yes</td>
</tr>
<tr>
<td>HFA</td>
<td>Erdas Imagine Images (.img)</td>
<td>.img</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Value

This function is used for the side-effect of writing values to a file.

See Also

writeFormats, writeValues

Examples

```r
r <- raster(system.file("external/test.grd", package="raster"))

# take a small part
r <- crop(r, extent(179880, 180880, 329880, 330840) )

# write to an integer binary file
rf <- writeRaster(r, filename="allint.grd", datatype='INT4S', overwrite=TRUE)

# make a brick and save multi-layer file
b <- brick(r, sqrt(r))
bf <- writeRaster(b, filename="multi.grd", bandorder='BIL', overwrite=TRUE)
```
# write to a new geotiff file (depends on rgdal)
if (require(rgdal)) {
  rf <- writeRaster(r, filename="test.tif", format="GTiff", overwrite=TRUE)
  bf <- writeRaster(b, filename="multi.tif", options="INTERLEAVE=BAND", overwrite=TRUE)
}

# write to netcdf
if (require(ncdf)) {
  rnc <- writeRaster(r, filename='netCDF.nc', format="CDF", overwrite=TRUE)
}

writeValues

Write values to a file

## Description

Functions for writing blocks (>= 1 row(s)) of values to files. Writing has to start at the first cell of a row (identified with argument start) and the values written must represent 1 or more entire rows. Begin by opening a file with writeStart, then write values to it in chunks. When writing is done close the file with writeStop.

If you want to write all values of a Raster* object at once, you can also use writeRaster which is easier to use but more limited. The functions described here allow writing values to file using chunks of different sizes (e.g. 1 or 10 rows). Function blockSize can be used to suggest a chunk size to use.

## Usage

```
writeStart(x, filename, ...)
writeValues(x, v, start)
writeStop(x)
```

## Arguments

- **x**: Raster* object
- **filename**: character. Output file name
- **...**: additional arguments as for writeRaster
- **v**: vector (RasterLayer) or matrix (RasterBrick) of values
- **start**: Integer. Row number (counting starts at 1) from where to start writing v

## Value

RasterLayer or RasterBrick

## See Also

writeRaster, blockSize, update
Examples

```r

r <- raster(system.file("external/test.grd", package="raster"))
# write to a new binary file in chunks
s <- raster(r)
#
tr <- blockSize(r)
tr
s <- writeStart(s, filename='test.grd', overwrite=TRUE)
for (i in 1:tr$n) {
  v <- getValuesBlock(r, row=tr$row[i], nrows=tr$nrows[i])
  s <- writeValues(s, v, tr$row[i])
}
s <- writeStop(s)

if(require(rgdal)){
  s2 <- writeStart(s, filename='test2.tif', format='GTiff', overwrite=TRUE)
  # writing last row first
  for (i in tr$n:1) {
    v <- getValuesBlock(r, row=tr$row[i], nrows=tr$nrows[i])
    s2 <- writeValues(s2, v, tr$row[i])
  }
  # row number 5 once more
  v <- getValuesBlock(r, row=5, nrows=1)
  writeValues(s2, v, 5)
  s2 <- writeStop(s2)
}

## write values of a RasterStack to a RasterBrick
s <- stack(system.file("external/rlogo.grd", package="raster"))
# create empty brick
b <- brick(s, values=FALSE)
b <- writeStart(b, filename="test.grd", format="raster", overwrite=TRUE)
tr <- blockSize(b)
for (i in 1:tr$n) {
  v <- getValuesBlock(s, row=tr$row[i], nrows=tr$nrows[i])
  b <- writeValues(b, v, tr$row[i])
}
b <- writeStop(b)
# note that the above is equivalent to
# b <- writeRaster(s, filename="test.grd", format="raster", overwrite=TRUE)
```

---

xyFromCell  
Coordinates from a row, column or cell number

**Description**

These functions get coordinates of the center of raster cells for a row, column, or cell number of a Raster* object.
xyFromCell

Usage

```r
## S4 method for signature 'Raster,numeric'
xyFromCol(object, col)
## S4 method for signature 'Raster,numeric'
xyFromRow(object, row)
## S4 method for signature 'Raster,numeric'
xyFromCell(object, cell)
## S4 method for signature 'Raster,numeric'
xyFromCell(object, cell)
## S4 method for signature 'BasicRaster,ANY'
xyFromCell(object, cell, spatial=FALSE, ...)
```

Arguments

- `object`: Raster* object (or a SpatialPixels* or SpatialGrid* object)
- `cell`: cell number(s)
- `col`: column number; or vector of column numbers
- `row`: row number; or vector of row numbers
- `spatial`: If `spatial=TRUE`, `xyFromCell` returns a SpatialPoints object instead of a matrix
- `...`: additional arguments. None implemented

Details

Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the Raster* object.

Value

- `xFromCol, yFromCol, xFromCell, yFromCell`: vector of x or y coordinates
- `xyFromCell`: matrix(x,y) with coordinate pairs

See Also

- `cellFromXY`

Examples

```r
# using a new default raster (1 degree global)
r <- raster()
xFromCol(r, c(1, 120, 180))
yFromRow(r, 90)
xyFromCell(r, 10000)
xyFromCell(r, c(0, 1, 32581, ncell(r), ncell(r)+1))

# using a file from disk
r <- raster(system.file("external/test.grd", package="raster"))
r
cellFromXY(r, c(180000, 330000))
```
# xy for corners of a raster:
xyFromCell(r, c(1, ncol(r), ncell(r)-ncol(r)+1, ncell(r)))

---

## z-values

**Get or set z-values**

### Description

Initial functions for a somewhat more formal approach to get or set z values (e.g. time) associated with layers of Raster* objects. In development.

### Usage

```r
setZ(x, z, name='time')
getZ(x)
```

### Arguments

- **x**: Raster* object
- **z**: vector of z values of any type (e.g. of class 'Date')
- **name**: character label

### Value

- **setZ**: Raster* object
- **getZ**: vector

### Examples

```r
r <- raster(ncol=10, nrow=10)
s <- stack(lapply(1:3, function(x) setValues(r, runif(ncell(r)))))
s <- setZ(s, as.Date('2000-1-1') + 0:2)
s
getZ(s)
```
Description

Experimental function to apply a function over a (time) series of layers of a Raster object

Usage

zApply(x, by, fun=mean, name='', ...)

Arguments

x : Raster* object
by : aggregation indices or function
fun : function to compute aggregated values
name : character label of the new time series
... : additional arguments

Value

Raster* object

Author(s)

Oscar Perpinan Lamigueiro & Robert J. Hijmans

Examples

# 12 values of irradiation, 1 for each month
GØdm=c(2.766,3.491,4.494,5.912,6.989,7.742,7.919,7.027,5.369,3.562,2.814,2.179) * 1000;
# RasterBrick with 12 layers based on GØdm + noise
r <- raster(nc=10, nr=10)
s <- brick(lapply(1:12, function(x) setValues(r, GØdm[x]+100*rnorm(ncell(r)))))

# time
tm <- seq(as.Date('2010-01-15'), as.Date('2010-12-15'), 'month')
s <- setZ(s, tm, 'months')

# library(zoo)
# x <- zApply(s, by=as.yearqtr, fun=mean, name='quarters')
Description

Compute zonal statistics, that is summarized values of a Raster* object for each "zone" defined by a RasterLayer.

If stat is a true function, zonal will fail (gracefully) for very large Raster objects, but it will in most cases work for functions that can be defined as by a character argument ('mean', 'sd', 'min', 'max', or 'sum'). In addition you can use 'count' to count the number of cells in each zone (only useful with na.rm=TRUE, otherwise freq(z) would be more direct.

If a function is used, it should accept a na.rm argument (or at least a ... argument)

Usage

```r
## S4 method for signature 'RasterLayer,RasterLayer'
zonal(x, z, fun='mean', digits=0, na.rm=TRUE, ...)

## S4 method for signature 'RasterStackBrick,RasterLayer'
zonal(x, z, fun='mean', digits=0, na.rm=TRUE, ...)
```

Arguments

- `x` Raster* object
- `z` RasterLayer with codes representing zones
- `fun` function to be applied to summarize the values by zone. Either as character: 'mean', 'sd', 'min', 'max', 'sum'; or, for relatively small Raster* objects, a proper function
- `digits` integer. Number of digits to maintain in 'zones'. By default averaged to an integer (zero digits)
- `na.rm` logical. If TRUE, NA values in x are ignored
- `...` additional arguments. One implemented: progress, as in `writeRaster`

Value

A matrix with a value for each zone (unique value in zones)

See Also

See `cellStats` for 'global' statistics (i.e., all of x is considered a single zone), and `extract` for summarizing values for polygons
Examples

```r
r <- raster(ncols=10, nrows=10)
r[] <- runif(ncell(r)) * 1:ncell(r)
z <- r
z[] <- rep(1:5, each=20)
# for big files, use a character value rather than a function
zonal(r, z, 'sum')

# for smaller files you can also provide a function
## Not run:
zonal(r, z, mean)
zonal(r, z, min)

## End(Not run)

# multiple layers
zonal(stack(r, r*10), z, 'sum')
```

Description

Zoom in on a map (plot) by providing a new extent, by default this is done by clicking twice on the map.

Usage

```r
zoom(x, ...)
## S4 method for signature 'Raster'
zoom(x, ext=drawExtent(), maxpixels=100000, layer=1, new=TRUE, useRaster=TRUE, ...)

## S4 method for signature 'Spatial'
zoom(x, ext=drawExtent(), new=TRUE, ...)

## S4 method for signature 'missing'
zoom(x, ext=drawExtent(), new=TRUE, ...)
```

Arguments

- `x`: Raster* or Spatial* (vector type) object
- `ext`: Extent object, or other object from which an extent can be extracted
- `maxpixels`: Maximum number of pixels used for the map
- `layer`: Positive integer to select the layer to be used if `x` is a mutilayer Raster object
- `new`: Logical. If `TRUE`, the zoomed in map will appear on a new device (window)
- `useRaster`: Logical. If `TRUE`, a bitmap raster is used to plot the image instead of polygons
- `...`: additional parameters for `plot`
Value

Extent object (invisibly)

See Also

drawExtent, plot
Index

!,-, Raster-method (Logic-methods), 121
!=-, BasicRaster, BasicRaster-method
   (Compare-methods), 54
*Topic classes
   Extent-class, 79
   Raster-class, 160
   Rcpp-class, 172
   readAll, 172
*Topic file
   extension, 76
   inifile, 109
*Topic math
   Arith-methods, 23
   atan2, 29
   Compare-methods, 54
   cv, 64
   Logic-methods, 121
   Math-methods, 124
   modal, 127
*Topic methods
   aggregate, 16
   animate, 20
   area, 22
   Arith-methods, 23
   as.data.frame, 24
   as.list, 26
   as.logical, 27
   as.matrix, 27
   as.raster, 29
   atan2, 29
   barplot, 32
   bind, 33
   blockSize, 34
   boundaries, 35
   brick, 37
   calc, 40
   clearValues, 48
   Compare-methods, 54
   contour, 57
   corLocal, 58
   cover, 59
crosstab, 62
erase, 74
extract, 80
Extract by index, 83
factors, 87
filledContour, 89
Gain and offset, 96
geom, 97
getValues, 99
corValuesBlock, 100
corValuesFocal, 101
head, 105
hist, 107
image, 108
interpolate, 110
intersect, 113
localFun, 119
Logic-methods, 121
mask, 122
match, 123
Math-methods, 124
merge, 125
mosaic, 128
overlay, 136
persp, 140
plot, 141
plotRGB, 144
predict, 147
quantile, 156
raster, 157
rasterFromXYZ, 163
rasterize, 164
replacement, 175
RGB, 177
setMinMax, 191
setValues, 191
spplot, 196
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack</td>
<td>197</td>
</tr>
<tr>
<td>stackApply</td>
<td>199</td>
</tr>
<tr>
<td>stackSelect</td>
<td>201</td>
</tr>
<tr>
<td>substitute</td>
<td>204</td>
</tr>
<tr>
<td>Summary</td>
<td>205</td>
</tr>
<tr>
<td>Summary-methods</td>
<td>206</td>
</tr>
<tr>
<td>syndif</td>
<td>207</td>
</tr>
<tr>
<td>text</td>
<td>210</td>
</tr>
<tr>
<td>union</td>
<td>213</td>
</tr>
<tr>
<td>unstack</td>
<td>215</td>
</tr>
<tr>
<td>update</td>
<td>216</td>
</tr>
<tr>
<td>writeRaster</td>
<td>222</td>
</tr>
<tr>
<td>writeValues</td>
<td>225</td>
</tr>
</tbody>
</table>

*Topic package*

- raster-package, 6

*Topic spatial*

- addLayer, 14
- adjacent, 15
- aggregate, 16
- alignExtent, 19
- animate, 20
- approxNA, 21
- area, 22
- Arith-methods, 23
- as.data.frame, 24
- as.list, 26
- as.logical, 27
- as.matrix, 27
- as.raster, 29
- autocorrelation, 30
- bands, 31
- barplot, 32
- bind, 33
- blockSize, 34
- boundaries, 35
- boxplot, 36
- brick, 37
- buffer, 39
- calc, 40
- cellFrom, 43
- cellsFromExtent, 45
- cellStats, 46
- clamp, 47
- clearValues, 48
- click, 49
- clump, 50
- cluster, 51
- colortable, 53

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>compareCRS</td>
<td>55</td>
</tr>
<tr>
<td>compareRaster</td>
<td>56</td>
</tr>
<tr>
<td>contour</td>
<td>57</td>
</tr>
<tr>
<td>corLocal</td>
<td>58</td>
</tr>
<tr>
<td>cover</td>
<td>59</td>
</tr>
<tr>
<td>crop</td>
<td>60</td>
</tr>
<tr>
<td>crosstab</td>
<td>62</td>
</tr>
<tr>
<td>cut</td>
<td>63</td>
</tr>
<tr>
<td>datasource</td>
<td>64</td>
</tr>
<tr>
<td>dataType</td>
<td>65</td>
</tr>
<tr>
<td>density</td>
<td>67</td>
</tr>
<tr>
<td>dim</td>
<td>67</td>
</tr>
<tr>
<td>direction</td>
<td>68</td>
</tr>
<tr>
<td>disaggregate</td>
<td>69</td>
</tr>
<tr>
<td>distance</td>
<td>70</td>
</tr>
<tr>
<td>distanceFromPoints</td>
<td>71</td>
</tr>
<tr>
<td>draw</td>
<td>73</td>
</tr>
<tr>
<td>drawExtent</td>
<td>73</td>
</tr>
<tr>
<td>erase</td>
<td>74</td>
</tr>
<tr>
<td>extend</td>
<td>75</td>
</tr>
<tr>
<td>extent</td>
<td>77</td>
</tr>
<tr>
<td>Extent math</td>
<td>78</td>
</tr>
<tr>
<td>Extent-class</td>
<td>79</td>
</tr>
<tr>
<td>extract</td>
<td>80</td>
</tr>
<tr>
<td>Extract by index</td>
<td>83</td>
</tr>
<tr>
<td>Extreme coordinates</td>
<td>85</td>
</tr>
<tr>
<td>extremeValues</td>
<td>86</td>
</tr>
<tr>
<td>factors</td>
<td>87</td>
</tr>
<tr>
<td>filename</td>
<td>89</td>
</tr>
<tr>
<td>filledContour</td>
<td>89</td>
</tr>
<tr>
<td>flip</td>
<td>90</td>
</tr>
<tr>
<td>flowPath</td>
<td>91</td>
</tr>
<tr>
<td>focal</td>
<td>92</td>
</tr>
<tr>
<td>focalWeight</td>
<td>94</td>
</tr>
<tr>
<td>freq</td>
<td>95</td>
</tr>
<tr>
<td>Gain and offset</td>
<td>96</td>
</tr>
<tr>
<td>geom</td>
<td>97</td>
</tr>
<tr>
<td>getData</td>
<td>98</td>
</tr>
<tr>
<td>getValues</td>
<td>99</td>
</tr>
<tr>
<td>getValuesBlock</td>
<td>100</td>
</tr>
<tr>
<td>getValuesFocal</td>
<td>101</td>
</tr>
<tr>
<td>gridDistance</td>
<td>102</td>
</tr>
<tr>
<td>hdr</td>
<td>104</td>
</tr>
<tr>
<td>head</td>
<td>105</td>
</tr>
<tr>
<td>hillShade</td>
<td>105</td>
</tr>
<tr>
<td>hist</td>
<td>107</td>
</tr>
<tr>
<td>image</td>
<td>108</td>
</tr>
<tr>
<td>initialize</td>
<td>109</td>
</tr>
</tbody>
</table>
zonal, 230
zoom, 231

* Topic *univar*
  cellstats, 46
cv, 64
freq, 95
modal, 127

*, SpatialPolygons, SpatialPolygons-method (Arith-methods), 23
+, SpatialLines, SpatialLines-method (Arith-methods), 23
+, SpatialPoints, SpatialPoints-method (Arith-methods), 23
+, SpatialPolygons, SpatialPolygons-method (Arith-methods), 23
-, SpatialPolygons, SpatialPolygons-method (Arith-methods), 23
==, BasicRaster, BasicRaster-method (Compare-methods), 54
[, Extent, missing, missing-method (Extract by index), 83
[, Extent, numeric, missing-method (Extract by index), 83
[, Raster, Extent, missing-method (Extract by index), 83
[, Raster, RasterLayer, missing-method (Extract by index), 83
[, Raster, Spatial, missing-method (Extract by index), 83
[, Raster, logical, missing-method (Extract by index), 83
[, Raster, matrix, missing-method (Extract by index), 83
[, Raster, missing, missing-method (Extract by index), 83
[, Raster, missing, numeric-method (Extract by index), 83
[, Raster, numeric, missing-method (Extract by index), 83
[, Raster, numeric, numeric-method (Extract by index), 83
[, RasterLayer, RasterLayer, missing-method (replacement), 175
$<-, Raster, matrix, missing-method (replacement), 175
$<-, Raster, missing, numeric-method (replacement), 175
$<-, Raster, numeric, missing-method (replacement), 175
$<-, RasterStackBrick, Raster, missing-method (replacement), 175
$<-, RasterStackBrick, missing, missing-method (replacement), 175

addLayer, 7, 14, 198, 200
addLayer, Raster-method (addLayer), 14
adjacent, 8, 15, 44
aggregate, 7, 12, 16, 70, 117, 175, 176
aggregate, Raster-method (aggregate), 16
aggregate, SpatialLines-method (aggregate), 16
aggregate, SpatialPolygons-method (aggregate), 16
alignExtent, 12, 19, 61
all.equal, 56, 125, 128
all.equal, Raster, Raster-method (compareRaster), 56
animate, 20
animate, RasterStackBrick-method (animate), 20
approx, 21
approxNA, 21
approxNA, RasterStackBrick-method
(approxNA), 21
area, 8, 22
area, RasterLayer-method (area), 22
area, RasterStackBrick-method (area), 22
area, SpatialPolygons-method (area), 22
Arith-methods, 7, 23, 42
as.array, 9
as.array (as.matrix), 27
as.array, RasterLayer-method
(as.data.frame), 24
as.data.frame, 12, 24
as.data.frame, Raster-method
(as.data.frame), 24
as.data.frame, SpatialLines-method
(as.data.frame), 24
as.data.frame, SpatialPolygons-method
(as.data.frame), 24
as.factor (Factors), 87
as.factor, RasterLayer-method (factors), 87
as.integer, 27
as.integer, Raster-method (as.logical), 27
as.list, 26
as.list, Raster-method (as.list), 26
as.logical, 27, 27
as.logical, Raster-method (as.logical), 27
as.matrix, 9, 27
as.matrix, Extent-method (as.matrix), 27
as.matrix, RasterLayer-method
(as.matrix), 27
as.matrix, RasterStackBrick-method
(as.matrix), 27
as.raster, 29, 29
as.raster, RasterLayer-method
(as.raster), 29
as.vector (as.matrix), 27
as.vector, Extent-method (as.matrix), 27
as.vector, Raster-method (as.matrix), 27
asFactor (factors), 87
asFactor, RasterLayer-method (factors), 87
atan2, 29, 124
atan2, Raster, Raster-method (atan2), 29
autocorrelation, 30
band, 11
bandnr (bands), 31
bandnr, RasterLayer-method (bands), 31
bands, 31, 131
boxplot, 10, 32, 33, 142
boxplot, RasterLayer-method (boxplot), 32
BasicRaster-class (Raster-class), 160
bbox, Extent-method (extent), 77
bbox, Raster-method (extent), 77
beginCluster (cluster), 51
bind, 12, 24, 33, 213
bind, data.frame, data.frame-method
(bind), 33
bind, data.frame, missing-method (bind), 33
bind, list, missing-method (bind), 33
bind, matrix, matrix-method (bind), 33
bind, matrix, missing-method (bind), 33
bind, SpatialLines, SpatialLines-method
(bind), 33
bind, SpatialPoints, SpatialPoints-method
(bind), 33
bind, SpatialPolygons, SpatialPolygons-method
(bind), 33
blockSize, 12, 34, 225
boundaries, 8, 35, 40, 69, 71
boundaries, RasterLayer-method
(boundaries), 35
boxplot, 10, 33, 36, 37, 107, 139
boxplot, RasterLayer-method (boxplot), 36
boxplot, RasterStackBrick-method
(boxplot), 36
bpy.colors, 141, 177
brick, 7, 37, 159, 161, 198
brick, array-method (brick), 37
brick, big.matrix-method (brick), 37
brick, character-method (brick), 37
brick, Extent-method (brick), 37
brick, grf-method (brick), 37
brick, kasc-method (brick), 37
brick, list-method (brick), 37
brick, missing-method (brick), 37
brick, RasterBrick-method (brick), 37
brick, RasterLayer-method (brick), 37
brick, RasterStack-method (brick), 37
brick, SpatialGrid-method (brick), 37
calc, 8, 24, 40, 51, 63, 121, 124, 137, 138, 156, 161, 173, 175, 199, 206
calc, Raster, function-method (calc), 40
canProcessInMemory, 13, 135
canProcessInMemory (Programming), 151
codes (getData), 98
celling, Extent-method (Extent math), 78
celling, RasterLayer-method (round), 180
cellFromCol (cellFrom), 43
cellFromColCombine (cellFrom), 43
cellFromColLine (cellFrom), 43
cellFromPolygon (cellFrom), 43
cellFromRow (cellFrom), 43
cellFromRowCol, 11
cellFromRowColCell (cellFrom), 43
cellFromRowColCell, numeric, numeric-method (cellFrom), 43
cellFromRowColCombine (cellFrom), 43
cellFromXY, 11, 46, 227
cellFromXYCell (cellFrom), 43
cellFromXY, BasicRaster-method (cellFrom), 43
cellsFromExtent, 11, 44, 45
cellStats, 9, 46, 118, 119, 157, 182, 205, 206, 230
cellStats, RasterLayer-method (cellStats), 46
cellStats, RasterStackBrick-method (cellStats), 46
clamp, 47, 173, 205
clamp, numeric-method (clamp), 47
clearValues, 48
click, 10, 12, 49, 189
click, missing-method (click), 49
click, Raster-method (click), 49
click, SpatialGrid-method (click), 49
click, SpatialLines-method (click), 49
click, SpatialPixels-method (click), 49
click, SpatialPoints-method (click), 49
click, SpatialPolygons-method (click), 49
clump, 8, 36, 50
cluster, 51
clusterR (cluster), 51
colFromCell (rowFromCell), 181
colFromCell, Raster, numeric-method (rowFromCell), 181
colFromX, 11
colFromX (cellFrom), 43
colFromX, Raster, numeric-method (cellFrom), 43
colorRampPalette, 141, 177
colortable, 53
colortable <- (colortable), 53
colSums (rowSums), 182
colSums, Raster-method (rowSums), 182
Compare-methods, 8, 54
compareCRS, 55
corraster, 11, 56
contour, 10, 57, 57, 108, 140, 142
contour, RasterLayer-method (contour), 57
countLines, 169
coordinates, 11, 97
cor, 58, 139
cor.test, 58
corLocal, 58, 120
 corLocal, RasterLayer, RasterLayer-method (corLocal), 58
corLocal, RasterStackBrick, RasterStackBrick-method (corLocal), 58
couldBeLonLat (isLonLat), 115
cov.wt, 119
cover, 8, 12, 59
cover, RasterLayer, RasterLayer-method (cover), 59
cover, RasterStackBrick, Raster-method (cover), 59
cover, SpatialPolygons, SpatialPolygons-method (cover), 59
crop, 7, 12, 60, 75, 76, 84, 113, 115, 123, 140, 141, 175, 176, 189, 213
crop, Raster-method (crop), 60
crop, Spatial-method (crop), 60
crosstab, 9, 62, 95
crosstab, Raster, Raster-method (crosstab), 62
crosstab, RasterStackBrick, missing-method (crosstab), 62
CRS, 152
crs, 55, 72
INDEX

crs (projection), 152
crs, ANY-method (projection), 152
crs <- (projection), 152
cut, 8, 32, 63, 63, 173, 205
cut, Raster-method (cut), 63
cv, 13, 64
cv, ANY-method (cv), 64
cv, Raster-method (cv), 64
dataSigned (properties), 155
dataSize (properties), 155
dataSource, 64
dataType, 61, 65, 134, 148, 223
dataType <- (dataType), 65
density, 10, 67, 139, 157
density, Raster-method (density), 67
deratify (factors), 87
dim, 67, 133
dim, BasicRaster-method (dim), 67
dim, RasterStackBrick-method (dim), 67
dim <-, BasicRaster-method (dim), 67
dim <-, RasterBrick-method (dim), 67
dim <-, RasterLayer-method (dim), 67
dimensions, 85
direction, 8, 68
direction, RasterLayer-method (direction), 68
disaggregate, 7, 12, 18, 69, 175, 176
disaggregate, Raster-method (disaggregate), 69
distance, 8, 40, 69, 70, 72, 103, 146
distance, RasterLayer, missing-method (distance), 70
distance, RasterLayer, RasterLayer-method (distance), 70
distance, Spatial, Spatial-method (distance), 70
distanceFromPoints, 8, 71, 71, 146
draw, 73
drawExtent, 10, 12, 19, 50, 61, 73, 78, 140, 141, 144, 178, 232
drawLine, 10
drawLine (draw), 73
drawPoly, 10
drawPoly (draw), 73
dropLayer, 7, 198, 203
dropLayer (addLayer), 14
dropLayer, RasterBrick-method (addLayer), 14
dropLayer, RasterStack-method (addLayer), 14
endCluster (cluster), 51
erase, 12, 24, 74, 207
erase, SpatialLines, SpatialPolygons-method (erase), 74
erase, SpatialPolygons, SpatialPolygons-method (erase), 74
extend, 7, 60, 61, 75, 129, 175, 176, 213
extend, Extent-method (extend), 75
extend, Raster-method (extend), 75
extension, 13, 76
extension <- (extension), 76
Extent, 73, 80, 144, 178
Extent (Extent-class), 79
extent, 11, 12, 19, 46, 61, 68, 77, 78, 79, 85, 115, 133, 136, 177, 190, 213
Extent math, 78
extent, BasicRaster-method (extent), 77
extent, Extent-method (extent), 77
extent, GridTopology-method (extent), 77
extent, list-method (extent), 77
extent, matrix-method (extent), 77
extent, numeric-method (extent), 77
extent, sf-method (extent), 77
extent, Spatial-method (extent), 77
Extent-class, 79
extent <- (setExtent), 190
extentFromCells (cellsFromExtent), 45
extract, 9, 12, 51, 80, 84, 138, 157, 166, 172, 201, 230
Extract by index, 83
extract, Raster, data.frame-method (extract), 80
extract, Raster, Extent-method (extract), 80
extract, Raster, matrix-method (extract), 80
extract, Raster, sf-method (extract), 80
extract, Raster, SpatialLines-method (extract), 80
extract, Raster, SpatialPoints-method (extract), 80
extract, Raster, SpatialPolygons-method (extract), 80
extract, Raster, vector-method (extract), 80
extract, SpatialPolygons, data.frame-method (extract), 80
extract, SpatialPolygons, matrix-method (extract), 80
extract, SpatialPolygons, SpatialPoints-method (extract), 80
Extreme coordinates, 85
extremeValues, 86
factors, 87
factorValues (factors), 87
filename, 11, 89, 156
filled.contour, 89, 90
filledContour, 10, 57, 89
flip, 7, 18, 90, 179, 194, 211
flip, RasterLayer-method (flip), 90
flip, RasterStack-brick-method (flip), 90
floor, Extent-method (Extent math), 78
floor, RasterLayer-method (round), 180
flowPath, 91
focal, 8, 21, 22, 30, 36, 92, 94, 95, 102, 209
focal, RasterLayer-method (focal), 92
focalWeight, 93, 94
fourCellsFromXY (cellFrom), 43
freq, 9, 47, 62, 95
freq, RasterLayer-method (freq), 95
freq, RasterStack-brick-method (freq), 95
fromDisk, 13
fromDisk (datasource), 64

gain (Gain and offset), 96
Gain and offset, 96
gain<- (Gain and offset), 96
Geary (autocorrelation), 30
GearyLocal (autocorrelation), 30
geom, 97
geom, SpatialLines-method (geom), 97
geom, SpatialPoints-method (geom), 97
geom, SpatialPolygons-method (geom), 97
geometry, 97
getCluster, 52
getCluster (Programming), 151
data, 13, 98
getValues, 9, 25, 28, 82, 84, 99, 101, 102,
157, 172, 191
getValues, RasterBrick, missing, missing-method (getValues), 99
getValues, RasterBrick, numeric, missing-method (getValues), 99
getValues, RasterLayer, missing, missing-method (getValues), 99
getValues, RasterLayer, numeric, missing-method (getValues), 99
getValues, RasterLayerSparse, missing, missing-method (getValues), 99
getValues, RasterLayerSparse, numeric, missing-method (getValues), 99
getValues, RasterLayerSparse, numeric, numeric-method (getValues), 99
getValues, RasterStack, missing, missing-method (getValues), 99
getValues, RasterStack, numeric, missing-method (getValues), 99
getValues, RasterStack, numeric, numeric-method (getValues), 99
getValuesBlock, 9, 25, 28, 100, 100, 105, 172
getValuesBlock, RasterBrick-method (getValuesBlock), 100
getValuesBlock, RasterLayer-method (getValuesBlock), 100
getValuesBlock, RasterLayerSparse-method (getValuesBlock), 100
getValuesBlock, RasterStack-method (getValuesBlock), 100
getValuesFocal, 9, 82, 100, 101
getValuesFocal, Raster, missing, missing, numeric-method (getValuesFocal), 101
getValuesFocal, Raster, numeric, numeric, numeric-method (getValuesFocal), 101
getValuesFocal, Raster-method (getValuesFocal), 101
getz, 21
getz (z-values), 228
gIntersection, 114
gridDistance, 8, 40, 69, 71, 72, 102, 146
gridDistance, RasterLayer-method (gridDistance), 102
hasValues (datasource), 64
hdr, 13, 104, 135
head, 105
head, RasterLayer-method (head), 105
head, RasterStack-brick-method (head), 105
head, Spatial-method (head), 105
heat.colors, 141, 177
hillshade, 105, 209
hist, 10, 33, 37, 107, 107, 139, 142
hist, Raster-method (hist), 107
image, RasterLayer-method (image), 108
image, RasterStackBrick-method (image), 108
image.plot, 142
inifile, 109
init, 8
init (initialize), 109
initialize, 109
inMemory, 13
inMemory (datasource), 64
interpolate, 9, 110, 148
interpolate, Raster-method (interpolate), 110
intersect, 12, 24, 45, 74, 113, 213
intersect, Extent, ANY-method (intersect), 113
intersect, Raster, ANY-method (intersect), 113
intersect, SpatialLines, SpatialLines-method (intersect), 113
intersect, SpatialLines, SpatialPolygons-method (intersect), 113
intersect, SpatialPoints, ANY-method (intersect), 113
intersect, SpatialPolygons, ANY-method (intersect), 113
intersect, SpatialPolygons, SpatialLines-method (intersect), 113
intersect, SpatialPolygons, SpatialPoints-method (intersect), 113
intersect, SpatialPolygons, SpatialPolygons-method (intersect), 113
is.factor (factors), 87
is.factor, Raster-method (factors), 87
is.factor, RasterStack-method (factors), 87
is.finite, Raster-method (Logic-methods), 121
is.infinite, Raster-method (Logic-methods), 121
is.na, CRS-method (projection), 152
is.na, Raster-method (Logic-methods), 121
is.nan, Raster-method (Logic-methods), 121
isLonLat, 11, 115
isLonLat, ANY-method (isLonLat), 115
isLonLat, BasicRaster-method (isLonLat), 115
isLonLat, character-method (isLonLat), 115
isLonLat, CRS-method (isLonLat), 115
isLonLat, Spatial-method (isLonLat), 115
KML, 12, 116
KML, RasterLayer-method (KML), 116
KML, RasterStackBrick-method (KML), 116
KML, Spatial-method (KML), 116
labels, Raster-method (names), 130
layerize, 117
layerize, RasterLayer, missing-method (layerize), 117
layerize, RasterLayer, RasterLayer-method (layerize), 117
layerStats, 46, 118
length, BasicRaster-method (ncell), 132
levelplot, 196, 197
levels (factors), 87
levels, Raster-method (factors), 87
levels, RasterStack-method (factors), 87
levels<-. (factors), 87
levels<-, Raster-method (factors), 87
lines, RasterLayer-method (plot), 141
lines, SpatialPolygons-method (spplot), 196
localFun, 8, 119, 120
localFun, RasterLayer, RasterLayer-method (localFun), 119
Logic, Raster, Raster-method (Logic-methods), 121
Logic-methods, 7, 121
make.names, 135, 218
mask, 8, 61, 74, 122
mask, Raster, sf-method (mask), 122
mask, Raster, Spatial-method (mask), 122
mask, RasterLayer, RasterLayer-method (mask), 122
INDEX

mask, RasterLayer, RasterStackBrick-method (mask), 122
mask, RasterStackBrick, RasterLayer-method (mask), 122
mask, RasterStackBrick, RasterStackBrick-method (mask), 122
match, 123, 124
match, Raster-method (match), 123
maxValue, 10, 47, 206
maxValue, RasterBrick-method (extremeValues), 86
maxValue, RasterLayer-method (extremeValues), 86
maxValue, RasterStack-method (extremeValues), 86
mean, Raster-method (Summary-methods), 206
merge, 7, 12, 34, 61, 76, 125, 128, 129, 175, 176, 213
merge, Extent, ANY-method (merge), 125
merge, Raster, Raster-method (merge), 125
merge, RasterStackBrick, missing-method (merge), 125
metadata, 126
metadata<- (metadata), 126
minValue, 9, 47, 206
minValue, RasterBrick-method (extremeValues), 86
minValue, RasterLayer-method (extremeValues), 86
minValue, RasterStack-method (extremeValues), 86
modal, 13, 127
modal, ANY-method (modal), 127
modal, Raster-method (modal), 127
Moran, 8
Moran (autocorrelation), 30
MoranLocal (autocorrelation), 30
mosaic, 7, 125, 128
mosaic, Raster, Raster-method (mosaic), 128
movingFun, 129
names, 11, 107, 130, 133, 135, 147
names, Raster-method (names), 130
names, RasterStack-method (names), 130
names<- (names), 130
names<-, Raster-method (names), 130
NAvalue, 11, 131
NAvalue<- (NAvalue), 131
nbands (bands), 31
ncell, 11, 68, 132, 177
ncell, ANY-method (ncell), 132
ncell, BasicRaster-method (ncell), 132
ncol, 10
ncol (ncell), 132
ncol, BasicRaster-method (ncell), 132
ncol<- (ncell), 132
nlayers, 11, 31, 32, 131, 133
nlayers, BasicRaster-method (nlayers), 133
nlayers, Raster-method (nlayers), 133
nlayers, RasterStack-method (nlayers), 133
nrows, 10
nrow, 10
nrow (ncell), 132
nrow, BasicRaster-method (ncell), 132
nrow<- (ncell), 132
offs (Gain and offset), 96
offs<- (Gain and offset), 96
Options, 134
options, 24, 135
origin, 11, 136
origin, BasicRaster-method (origin), 136
origin<- (origin), 136
origin<-, BasicRaster-method (origin), 136
over, 12
overlay, 8, 24, 41, 42, 51, 121, 124, 136
overlay, Raster, missing-method (overlay), 136
overlay, Raster, Raster-method (overlay), 136
pairs, 10, 37, 107, 139, 139, 142
pairs, Raster-method (pairs), 139
pbClose, 13
pbClose (Programming), 151
pbCreate, 13
INDEX

pbCreate (Programming), 151
pbStep, 13
pbStep (Programming), 151
persp, 10, 57, 90, 140, 140, 142
persp, RasterLayer-method (persp), 140
persp, RasterStackBrick-method (persp), 140
plot, Extent, missing-method (plot), 141
plot, Raster, ANY-method (plot), 141
plot, Raster, Raster-method (plot), 141
plot3D, 140
plotRGB, 10, 20, 54, 142, 144, 178, 197, 202
plotRGB, RasterStackBrick-method (plotRGB), 144
pointDistance, 13, 40, 71, 72, 145
predict, 8, 111, 147, 161, 204
predict, Raster-method (predict), 147
predict.gstat, 111
pretty, 142
print, Raster-method (Raster-class), 160
print, Spatial-method (Raster-class), 160
Programming, 150
proj4string (projection), 152
proj4string<- (projection), 152
projectExtent (projectRaster), 153
projection, 11, 55, 152, 208
projection<- (projection), 152
projectRaster, 7, 10, 51, 153, 153, 175, 176
projInfo, 153, 154
properties, 155
quantile, 47, 156, 156
quantile, Raster-method (quantile), 156
rainbow, 32, 141, 177
raster, 7, 9, 39, 157, 161, 198
raster, asc-method (raster), 157
raster, BasicRaster-method (raster), 157
raster, big.matrix-method (raster), 157
raster, character-method (raster), 157
raster, Extent-method (raster), 157
raster, grf-method (raster), 157
raster, GridTopology-method (raster), 157
raster, im-method (raster), 157
raster, kasc-method (raster), 157
raster, kde-method (raster), 157
raster, list-method (raster), 157
raster, matrix-method (raster), 157
raster, missing-method (raster), 157
raster, RasterBrick-method (raster), 157
raster, RasterLayer-method (raster), 157
raster, RasterLayerSparse-method (raster), 157
raster, RasterStack-method (raster), 157
raster, sf-method (raster), 157
raster, Spatial-method (raster), 157
raster, SpatialGrid-method (raster), 157
raster, SpatialPixels-method (raster), 157
Raster-class, 6, 160
raster-package, 6
RasterBrick-class (Raster-class), 160
rasterFromCells, 9, 162
rasterFromXYZ, 9, 163
rasterImage, 29, 108, 142, 145
rasterize, 9, 84, 123, 138, 164, 164, 175
rasterize, data.frame, Raster-method (rasterize), 164
rasterize, Extent, Raster-method (rasterize), 164
rasterize, matrix, Raster-method (rasterize), 164
rasterize, sf, Raster-method (rasterize), 164
rasterize, SpatialLines, Raster-method (rasterize), 164
rasterize, SpatialPoints, Raster-method (rasterize), 164
rasterize, SpatialPolygons, Raster-method (rasterize), 164
RasterLayer-class (Raster-class), 160
RasterLayerSparse-class (Raster-class), 160
rasterOptions, 13, 56, 169, 222, 223
rasterOptions (Options), 134
RasterStack-class (Raster-class), 160
RasterStackBrick-class (Raster-class), 160
rasterTmpFile, 13, 134, 135, 168
rasterToContour, 9, 57, 169
rasterToPoints, 9, 170, 210
rasterToPolygons, 9, 171
ratify (factors), 87
Rcpp-class, 172
Rcpp_SpExtent-class (Rcpp-class), 172
setValues, RasterLayer-method
   (setValues), 191
setValues, RasterLayerSparse-method
   (setValues), 191
setValues, RasterStack-method
   (setValues), 191
setZ (z-values), 228
shapefile, 193
shapefile, character-method (shapefile), 193
shapefile, Spatial-method (shapefile), 193
shift, 7, 194
shift, Raster-method (shift), 194
shift, SpatialLines-method (shift), 194
shift, SpatialPoints-method (shift), 194
shift, SpatialPolygons-method (shift), 194
show, BasicRaster-method (Raster-class), 160
show, Extent-method (Extent-class), 79
show, RasterBrick-method (Raster-class), 160
show, RasterLayer-method (Raster-class), 160
show, RasterStack-method (Raster-class), 160
showTmpFiles, 13
showTmpFiles (rasterTmpFile), 168
Slope and aspect, 195
slopeAspect (Slope and aspect), 195
sp, 195
SpatialLines, 80
SpatialPoints, 80
SpatialPolygons, 80
SpatialVector-class (Raster-class), 160
spDistSN1, 146
SpExtent (Rcpp-class), 172
spLines (sp), 195
spplot, 10, 20, 142, 196, 196, 197
spplot, Raster-method (spplot), 196
spplot, SpatialPoints-method (spplot), 196
SpPoly (Rcpp-class), 172
SpPoly-class (Rcpp-class), 172
SpPolygons (Rcpp-class), 172
spPolygons (sp), 195
spPolygons-class (Rcpp-class), 172
SpPolyPart (Rcpp-class), 172
SpPolyPart-class (Rcpp-class), 172
spTransform, 154
stack, 7, 37, 159, 161, 197, 200, 215
stack, character-method (stack), 197
stack, kasc-method (stack), 197
stack, list-method (stack), 197
stack, missing-method (stack), 197
stack, Raster-method (stack), 197
stack, SpatialGridDataFrame-method (stack), 197
stack, SpatialPixelsDataFrame-method (stack), 197
stackApply, 8, 199, 201
stackOpen (stackSave), 200
stackSave, 200
stackSelect, 8, 199, 201
stackSelect, RasterStackBrick, Raster-method (stackSelect), 201
stretch, 202
subs, 8, 63, 125, 173
subs (substitute), 204
subs, Raster, data.frame-method (substitute), 204
subset, 7, 14, 203
subset, Raster-method (subset), 203
subset, RasterStack-method (subset), 203
substitute, 204
Summary, 205
summary, RasterLayer-method (Summary), 205
summary, RasterStackBrick-method (Summary), 205
Summary-methods, 8, 206
symdif, 12, 207
symdiff, SpatialPolygons, SpatialPolygons-method (symdiff), 207
Sys.getpid, 168
t, 7, 90
t (transpose), 211
t, RasterLayer-method (transpose), 211
t, RasterStackBrick-method (transpose), 211
table, 95
tail (head), 105
tail, RasterLayer-method (head), 105
tail, RasterStackBrick-method (head), 105
247

writeRaster, RasterLayer, character-method (writeRaster), 222
writeRaster, RasterStackBrick, character-method (writeRaster), 222
writeStart, 12
writeStart (writeValues), 225
writeStart, RasterBrick-method (writeValues), 225
writeStart, RasterLayer, character-method (writeValues), 225
writeStop, 12
writeStop (writeValues), 225
writeStop, RasterBrick-method (writeValues), 225
writeValues, 12, 23, 24, 223, 224, 225
writeValues, RasterBrick, matrix-method (writeValues), 225
writeValues, RasterLayer, vector-method (writeValues), 225

xFromCell, 11
xFromCell (xyFromCell), 226
xFromCell, Raster, numeric-method (xyFromCell), 226
xFromCol, 11
xFromCol (xyFromCell), 226
xFromCol, Raster, numeric-method (xyFromCell), 226
xmax, 11
xmax (Extreme coordinates), 85
xmax, BasicRaster-method (Extreme coordinates), 85
xmax, Extent-method (Extreme coordinates), 85
xmax, Spatial-method (Extreme coordinates), 85
xmax< (Extreme coordinates), 85
xmin, 11
xmin (Extreme coordinates), 85
xmin, BasicRaster-method (Extreme coordinates), 85
xmin, Extent-method (Extreme coordinates), 85
xmin< (Extreme coordinates), 85
yFromCell, 11
yFromCell (xyFromCell), 226
yFromCell, Raster, numeric-method (xyFromCell), 226
yFromRow, 11
yFromRow (xyFromCell), 226
yFromRow, Raster, numeric-method (xyFromCell), 226
ymax, 11
ymax (Extreme coordinates), 85
ymax, BasicRaster-method (Extreme coordinates), 85
ymax, Extent-method (Extreme coordinates), 85
ymax, Spatial-method (Extreme coordinates), 85
ymax< (Extreme coordinates), 85
ymin, 11
ymin (Extreme coordinates), 85
ymin, BasicRaster-method (Extreme coordinates), 85
ymin, Extent-method (Extreme coordinates), 85
ymin, Spatial-method (Extreme coordinates), 85
ymin< (Extreme coordinates), 85
yres, 11
yres (resolution), 176
yres, BasicRaster-method (resolution), 176

z-values, 228
zApply, 229
zonal, 9, 62, 95, 230
zonal, RasterLayer, RasterLayer-method (zonal), 230
zonal, RasterStackBrick, RasterLayer-method (zonal), 230
zoom, 10, 140, 141, 231
zoom, missing-method (zoom), 231
zoom, Raster-method (zoom), 231
zoom, Spatial-method (zoom), 231