Package 'palr'

August 5, 2026

Type Package

Title Colour Palettes for Data

LazyData yes

Version 0.5.0

Description Colour palettes for data, based on some well known public data sets. Includes helper functions to map absolute values to known palettes, and capture the work of image colour mapping as raster data sets.

Depends R (>= 3.6.0)

Imports grDevices

Suggests knitr, rmarkdown, raster, testthat (>= 3.0.0), covr, stars

VignetteBuilder knitr

Config/testthat/edition 3

License GPL-3

URL https://australianantarcticdivision.github.io/palr/, https://github.com/AustralianAntarcticDivision/palr

BugReports https://github.com/AustralianAntarcticDivision/palr/issues

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Michael D. Sumner [aut, cre, cph], Abigael Proctor [ctb] (Named the package), Tomas Remenyi [ctb] (Provided colours for element_pal), R Core Team and contributors worldwide [ctb] (source code of image.default)

Maintainer Michael D. Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2026-08-05 05:30:02 UTC

Contents

bathy_deep_pal . . . 2
chl_pal . . . 3
col2hex . . . 4
dirty_pal . . . 4
d_pal . . . 5
ice_pal . . . 6
image_hex . . . 7
image_pal . . . 8
mk_timePal . . . 9
oisst . . . 10
palr . . . 11
sst_pal . . . 11
stretch_linear . . . 12

Index **14**

`bathy_deep_pal` *Bathymetry*

Description

Deep bathymetry colours.

Usage

```
bathy_deep_pal(x, palette = FALSE, alpha = 1, ...)

bathyDeepPal(x, palette = FALSE, alpha = 1, ...)
```

Arguments

`x`	a vector of data values or a single num (n)
`palette`	logical, if `TRUE` return a list with matching colours and values
`alpha`	value in 0,1 to specify opacity
`...`	currently ignored

Details

Colour ramp suitable for deep waters (-5500) to sea level. The palette functions operate in 3 modes: 1) n colours - Pal(6) - returns 6 colours from the palette 2) data - Pal(c(-4000, -2000, -100)) - return colours for 3 depths 3) palette - Pal(palette = TRUE) - return the full palette and breaks Derived from maps created in Matlab by Emmanuel Laurenceau.

Value

colours, palette, or function, see Details

Examples

```
plot(1:15, pch = 19, cex = 4, col  = bathy_deep_pal(15))
```

---

chl_pal *Ocean colour colours for chlorophyll-a.*

Description

Ocean colour palette for chlorophyll-a.

Usage

```
chl_pal(x, palette = FALSE, alpha = 1)

chlPal(x, palette = FALSE, alpha = 1, ...)
```

Arguments

x	a vector of data values or a single number
palette	logical, if TRUE return a list with matching colours and values
alpha	value in 0,1 to specify opacity
...	currently unused

Details

Flexible control of the chlorophyll-a palette. If x is a single number, the function returns that many colours evenly spaced from the palette. If x is a vector of multiple values the palette is queried for colours matching those values, and these are returned. If x is missing and palette is FALSE then a function is returned that will generate n evenly spaced colours from the palette, as per colorRampPalette.

Value

colours, palette, or function, see Details

References

Derived from a file once found at 'http://oceancolor.gsfc.nasa.gov/DOCS/palette_chl_etc.txt'

Examples

```
## just get a small number of evenly space colours
plot(1:10, col = chl_pal(10))
## store the full palette and work with values and colours
pal <- chl_pal(palette = TRUE)
## the standard full palette
image(chl, breaks = pal$breaks, col = pal$cols[-1])
```

col2hex *Colour to hex conversion.*

Description

Create colours from colour names in one easy step.

Usage

```
col2hex(x, alpha = 1)
```

Arguments

x	vector of colour names or hex strings
alpha	optional transparency value in [0,1], can be per colour in x

Value

character string of hex colours

Examples

```
col2hex(c("aliceblue", "firebrick"), alpha = c(1, .5))
col2hex(c("#FFFFFF", "#123456FF"), alpha = 0.1)
```

dirty_pal *Bathymetry palette, absolute for global use*

Description

From AAD underway facility (DiRT).

Usage

```
dirty_pal(x, palette = FALSE, alpha = 1)
```

Arguments

x	a vector of data values or a single num (n)
palette	logical, if TRUE return a list with matching colours and values
alpha	0 to 1 for transparency

Details

The palette functions operate in 3 modes: 1) n colours - Pal(6) - returns 6 colours from the palette 2) data - Pal(c(-4000, -2000, 100)) - return colours for 3 elevations 3) palette - Pal(palette = TRUE) - return the full palette and breaks

The palette spans -8000 to 1000 metres, values outside that range clamp to the end colours.

Value

colours, palette, or function, see Details

Examples

```
dirty_pal(seq(-8000, 0, length.out = 10))
```

---

d_pal *Colours for data values*

Description

Scales input data to the palette, so that colour is mapped linearly to the range of values.

Usage

```
d_pal(x, pal = hcl.colors(84))

data_pal(x, pal = hcl.colors(84))
```

Arguments

x	data vector, maybe be numeric or character
pal	palette, may be colours or a function

Details

Default palette 'pal' is the 'viridis' colours of [grDevices::hcl.colors()], and may be literal colour values or a function.

[data_pal()] is an alias of [d_pal()].

Value

character string of hex colours

Examples

```
plot(1:100, col = d_pal(1:100))
plot(1:100, col = d_pal(1:100, chl_pal))
```

ice_pal *Sea ice colours*

Description

Colours for sea ice.

Usage

```
ice_pal(x, palette = FALSE, alpha = 1, ..., amsre = FALSE)

icePal(x, palette = FALSE, alpha = 1, ...)
```

Arguments

x	a vector of data values or a single num (n)
palette	logical, if TRUE return a list with matching colours and values
alpha	value in 0,1 to specify opacity
...	currently ignored
amsre	use old AMSRE colours ('FALSE' by default)

Details

The palette functions operate in 3 modes: 1) n colours - Pal(6) - returns 6 colours from the palette 2) data - Pal(c(10, 50, 100)) - return colours for 3 ice concentrations 3) palette - Pal(palette = TRUE) - return the full palette and breaks

Value

colours, palette, or function, see Details

References

amsre colours derived from 'http://www.iup.uni-bremen.de/seaice/amsr/'., nsidc colours extracted in data-raw/.

Examples

```
if (requireNamespace("raster")) {
nsidcfile <- system.file("extdata", "nt_20140320_f17_v01_s.bin",
    package = "palr", mustWork = TRUE)
r <- raster::raster(nsidcfile)
icp <- ice_pal(palette = TRUE)
## The AMSR colours
raster::plot(r, col = icp$col, zlim = range(icp$breaks),
main = sprintf("NSIDC ice %s", format(raster::getZ(r))))
}
```

image_hex *Convert image data to a matrix of hex colours*

Description

'image_hex()' turns numeric, integer, raw, logical, or character image data into a character matrix of hex colours, ready for [graphics::rasterImage()], writing to PNG/GeoTIFF, or use as a texture. This is the colour engine shared with the 'ximage' package.

Usage

```
image_hex(
  x,
  col = NULL,
  breaks = NULL,
  zlim = NULL,
  alpha = NULL,
  na.col = "transparent"
)
```

Arguments

x	matrix or array of image data (see Details)
col	colours to map to, a vector or a function
breaks	numeric breakpoints for the colours
zlim	absolute data range for the palette (single-band only)
alpha	opacity multiplier in [0, 1]
na.col	colour for missing values (default "transparent")

Details

Supported inputs are a matrix (palette mapping via 'col', 'breaks', 'zlim'), a 3D array with 1 band (treated as a matrix), 2 bands (grey + alpha), 3 bands (RGB), or 4 bands (RGBA), a raw matrix or array (converted to integer), and a character matrix of colours (passed through, with NA replaced by 'na.col').

Behaviour guarantees:

Non-finite values (NA, NaN, Inf) always map to 'na.col'. Colour scaling for multi-band data is autodetected per image: data within [0, 1] is used as-is, within [0, 255] is divided by 255, anything else is rescaled by the finite range of the colour bands jointly (so hue relationships are preserved); an alpha band is scaled independently. Constant (zero-range) data maps to the middle of the palette. 'zlim' anchors the palette to an absolute range and values outside it map to 'na.col'; 'zlim' is ignored with a warning for multi-band input. 'alpha' is a constant (or recycled) opacity multiplier in [0, 1] applied on top of any existing alpha.

Use the stretch functions ([stretch_linear()] and friends) to normalize high dynamic range data before colour mapping.

Value

character matrix of hex colours

Examples

```
m <- image_hex(volcano)
dim(m)
## rgb array in 0, 255
arr <- array(c(volcano, volcano / 2, 255 - volcano), c(dim(volcano), 3L))
m <- image_hex(arr)
## plot it
plot(NA, xlim = c(0, 1), ylim = c(0, 1), asp = 1)
rasterImage(image_hex(volcano, col = grDevices::hcl.colors(64)), 0, 0, 1, 1)
```

---

image_pal *Map data values to colours*

Description

If no ‘col‘ is provided, the default image palette is used. The density can be controlled with ‘n‘ and the mapping with the optional ‘breaks‘. If ‘breaks‘ is included as well as ‘n‘, ‘n‘ is ignored.

Usage

```
image_pal(x, col, ..., breaks = NULL, n = NULL, zlim = NULL)

image_raster(x, col, ..., breaks = NULL, n = NULL, zlim = NULL)

image_stars(x, col, ..., breaks = NULL, n = NULL, zlim = NULL)
```

Arguments

x	numeric values, raster object (single layer only) or stars object (single variable, 2D array only)
col	function to generate colours, or a vector of hex colours
...	ignored
breaks	optionally used to specify colour mapping
n	optionally used to specify density of colours from ‘col‘ (ignored if ‘breaks‘ is set)
zlim	numeric range to clamp values to an absolute scale (ignored if ‘breaks‘ is set)

Details

The function ‘image_pal()‘ only returns hex character string colours. The function ‘image_raster()‘ will map a raster of numeric values to an RGB 3-layer (channel) raster brick, and ‘image_stars()‘ similarly for a 3-dimensional stars object.

Please note that the expansion to 3-channels is a fairly wasteful thing to do, the overall data is expanded from a single layer to three but this faciliates a specific task of creating textures for 3D mapping, and this is the only way to do it currently. It's also useful in other situations, for controlling exactly the kind of plots we can achieve and for exporting to image formats such as 'GeoTIFF' or 'PNG'.

Value

for ‘image_pal()‘ a vector of hex colours, for ‘image_raster‘ and ‘image_stars‘ a raster or stars object with 3 channel RGB (range 0,255)

Examples

```
vals <- sort(rnorm(100))
cols <- image_pal(vals, zlim = c(-2.4, .5))
plot(vals, col = cols); abline(h = .5)
points(vals, pch  = ".") ## zlim excluded some of the range
```

---

mk_timePal *Time-indexed colour.*

Description

Create a time-indexed colour map, useful for maintaining an absolute scale across time series as a function of date-time.

Usage

```
mk_timePal(x, col)
```

Arguments

x	date-times
col	colours, can be a function or an actual set of colours

Value

function of date-time

Examples

```
dts <- seq(as.Date("1749-01-01"), by = "1 month", length.out = length(sunspots))
d <- data.frame(date = dts, sunspots = as.vector(t(sunspots)))
tpal <- mk_timePal(d$date, col = sst_pal(50))
par(mfrow  = c(2, 1))
plot(sunspots ~ date, col = tpal(date), data = d)
## colours maintained by absolute date
plot(sunspots ~ date, col = tpal(date), data = d[1500:1800, ], cex = 2)
## we can now insert new points and maintain this colour ramp
d2 <- data.frame(date = seq(min(d$date), max(d$date), by = "5 days"))
d2$sunspots <- approxfun(d$date, d$sunspots)(d2$date)
points(sunspots ~ date, col = tpal(date), data = d2, pch = 19, cex = 0.5)
```

---

oisst *Sea surface temperature (SST) and ocean color chlorophyll-a (CHLA).*

Description

SST example raster data set, at 0.25 degree resolution for global coverage in "longitude180/latitude".

Details

Small sample of 'VIIRS' daily chlorophyll-a 2026-06-15 from near Tasmania.

Created using script in data-raw/ using ‘raadtools‘ package.

References

Reynolds, et al.(2007) Daily High-resolution Blended Analyses. Available from 'NOAA' search for 'OISST'.

Climatology is based on 1971-2000 OI.v2 SST, Satellite data: Navy NOAA19 METOP AVHRR, Ice data: #' NCEP ice Source: NOAA/National Climatic Data Center.

Examples

```
dim(oisst)
class(oisst)
image(oisst, useRaster = TRUE)
pal <- chl_pal(palette = TRUE)
image(t(chl[nrow(chl):1, ]), col = pal$cols[-1], breaks = pal$breaks)
```

palr *palr*

Description

palr: colours for data

Author(s)

Maintainer: Michael D. Sumner <mdsumner@gmail.com> [copyright holder]

Authors:

- Michael D. Sumner <mdsumner@gmail.com> [copyright holder]

Other contributors:

- Abigael Proctor (Named the package) [contributor]
- Tomas Remenyi (Provided colours for element_pal) [contributor]
- R Core Team and contributors worldwide (source code of image.default) [contributor]

See Also

Useful links:

- https://australianantarcticdivision.github.io/palr/
- https://github.com/AustralianAntarcticDivision/palr
- Report bugs at https://github.com/AustralianAntarcticDivision/palr/issues

---

sst_pal *SST colours*

Description

SST colours

Usage

```
sst_pal(x, palette = FALSE, alpha = 1, ...)

sstPal(x, palette = FALSE, alpha = 1, ...)
```

Arguments

x	a vector of data values or a single number
palette	logical, if TRUE return a list with matching colours and values
alpha	value in 0,1 to specify opacity
...	currently ignored

Value

colours, palette, or function, see Details

References

Derived from a file once found at 'http://oceancolor.gsfc.nasa.gov/DOCS/palette_sst.txt'

Examples

```
data(oisst)
sstcols <- sst_pal(palette = TRUE)
image(oisst, col = sstcols$cols, zlim = range(sstcols$breaks))
```

---

stretch_linear *Stretch numeric values to [0, 1]*

Description

A family of contrast-stretch functions that rescale numeric vectors, matrices, or arrays to the unit interval. These are the data-normalization step that naturally precedes colour mapping with [d_pal()], [image_pal()], or [image_hex()]. Dimensions of the input are preserved, so a stretched matrix can go straight to an image plot.

Usage

```
stretch_linear(x, low = 0.02, high = 0.98, lim = NULL)

stretch_log(x, low = 0.02, high = 0.98, offset = NULL, lim = NULL)

stretch_sqrt(x, low = 0.02, high = 0.98, offset = NULL, lim = NULL)

stretch_histeq(x, n = 256)
```

Arguments

x	numeric vector, matrix, or array (NA values are preserved)
low, high	quantile probabilities for the clip bounds (default 2nd and 98th percentiles), ignored if 'lim' is set
lim	optional absolute clip bounds in data units, a range like 'c(0, 3000)' (for 'stretch_log()' and 'stretch_sqrt()' this is given on the original data scale and transformed internally)
offset	numeric offset added before the log or sqrt transform. If 'NULL' (default), autodetected so that all values are non-negative.
n	number of bins for histogram equalisation (default 256)

Details

'stretch_linear()' clips to quantile bounds then linearly rescales, this is the workhorse default.

'stretch_log()' applies 'log1p' after shifting values to be non-negative, then linearly rescales. Useful for data with high dynamic range (e.g. overview-resolution satellite imagery where integer counts span several orders of magnitude).

'stretch_sqrt()' applies square-root after shifting, gentler compression than log, a good middle ground.

'stretch_histeq()' maps values to their empirical quantile rank, producing an approximately uniform distribution. Good for scenes with detail in both deep shadow and bright areas simultaneously.

By default the clip bounds are quantiles of the data itself, so the result is scene-dependent. Use 'lim' to give absolute bounds in data units instead (for example 'lim = c(0, 3000)' for Sentinel-2 surface reflectance), which fixes the mapping so that it is stable across tiles and timesteps (no seams in mosaics, no flicker in animations). When 'lim' is set the quantile arguments are ignored.

Values that stretch outside [0, 1] are clamped. Non-finite input values (NA, NaN) are returned as NA; constant input maps to 0.5.

For multi-band data (e.g. RGB), apply per-band for maximum per-channel contrast, or pool all bands into one vector for a joint stretch that preserves inter-band luminance relationships.

Value

numeric vector, matrix, or array matching the input, with values in [0, 1] (NA where input was not finite)

References

worked examples here sentinel2_stretch

Examples

```
x <- sort(rnorm(200))
plot(x, stretch_linear(x), pch = ".")
plot(x, stretch_sqrt(x), pch = ".")

## high dynamic range data benefits from log stretch
x <- c(rexp(500, 0.01), rexp(500, 1))
plot(x, stretch_linear(x), pch = ".", main = "linear")
plot(x, stretch_log(x), pch = ".", main = "log")

## histogram equalisation
plot(x, stretch_histeq(x), pch = ".", main = "histeq")

## absolute bounds, stable across scenes
stretch_linear(c(-100, 0, 1500, 3000, 9000), lim = c(0, 3000))

## feed into colour mapping
plot(1:200, col = d_pal(stretch_linear(sort(rnorm(200)))), pch = 19)
```

Index

bathy_deep_pal, 2
bathyDeepPal *(*bathy_deep_pal*)*, 2

chl *(*oisst*)*, 10
chl_pal, 3
chlPal *(*chl_pal*)*, 3
col2hex, 4
colorRampPalette, *3*

d_pal, 5
data_pal *(*d_pal*)*, 5
dirty_pal, 4

ice_pal, 6
icePal *(*ice_pal*)*, 6
image_hex, 7
image_pal, 8
image_raster *(*image_pal*)*, 8
image_stars *(*image_pal*)*, 8

mk_timePal, 9

oisst, 10

palr, 11
palr-package *(*palr*)*, 11

sst_pal, 11
sstPal *(*sst_pal*)*, 11
stretch_histeq *(*stretch_linear*)*, 12
stretch_linear, 12
stretch_log *(*stretch_linear*)*, 12
stretch_sqrt *(*stretch_linear*)*, 12