Package 'inlcolor'

July 22, 2025

Title Color Schemes for the USGS Idaho National Laboratory Project Office

Version 1.0.6

Description A collection of functions for creating color schemes. Used to support packages and scripts written by researchers at the United States Geological Survey (USGS) Idaho National Laboratory Project Office.

Depends R (>= 4.1)

Imports checkmate, graphics, grDevices, rlang, scales

Suggests connectapi, covr, datasets, dichromat, httr, methods, pkgbuild, pkgdown, pkgload, rcmdcheck, renv, rmarkdown, roxygen2, rsconnect, stats, tinytest, tinytex, tools, utils, xtable

License CC0

URL https://rconnect.usgs.gov/INLPO/inlcolor-main/, https://code.usgs.gov/inl/inlcolor

BugReports https://code.usgs.gov/inl/inlcolor/-/issues

Copyright This software is in the public domain because it contains materials that originally came from the United States Geological Survey (USGS), an agency of the United States Department of Interior. For more information, see the official USGS copyright policy at https://www.usgs.gov/information-policies-and-instructions/copyrights-and-credits

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation no

Author Jason C. Fisher [aut, cre] (ORCID: <https://orcid.org/0000-0001-9032-8912>)

Maintainer Jason C. Fisher <jfisher@usgs.gov>

Repository CRAN

Date/Publication 2024-01-30 18:20:02 UTC

Contents

get_colors 2
plot.inlpal 8
set_hinge 9

Index **13**

`get_colors` *Get color palette*

Description

Create a vector of `n` colors from qualitative, diverging, and sequential color schemes.

Usage

```
get_colors(
  n,
  scheme = "smooth rainbow",
  alpha = NULL,
  stops = c(0, 1),
  bias = 1,
  reverse = FALSE,
  blind = NULL,
  gray = FALSE,
  ...
)
```

Arguments

`n`	'integer' count. Number of colors to be in the palette. The maximum number of colors in a generated palette is dependent on the specified color scheme, see 'Details' section for maximum values.
`scheme`	'character' string. Name of color scheme, see 'Details' section for scheme descriptions. Argument choices may be abbreviated as long as there is no ambiguity.
`alpha`	'numeric' number. Alpha transparency, values range from 0 (fully transparent) to 1 (fully opaque). Specify as `NULL` to exclude the alpha channel value from colors.
`stops`	'numeric' vector of length 2. Color stops defined by interval endpoints (between 0 and 1) and used to select a subset of the color palette. Only suitable for schemes that allow for color interpolations.
`bias`	'numeric' number. Interpolation bias where larger values result in more widely spaced colors at the high end.
`reverse`	'logical' flag. Whether to reverse the order of colors in the scheme.

`blind` 'character' string. Type of color blindness to simulate: specify `"deutan"` for green-blind vision, `"protan"` for red-blind vision, `"tritan"` for green-blue-blind vision, or `"monochrome"` for total-color blindness. A partial-color blindness simulation requires that the **dichromat** package is available, see `dichromat::dichromat` function for additional information. Argument choices may be abbreviated as long as there is no ambiguity.

`gray` 'logical' flag. Whether to subset/reorder the `"bright"`, `"high-contrast"`, `"vibrant"`, and `"muted"` schemes to work well after conversion to gray scale.

`...` Not used

Details

The suggested data type for color schemes and the characteristics of generated palettes are given in the tables below. [**Type**: is the type of data being represented, either qualitative, diverging, or sequential. **Max n**: is the maximum number of colors in a generated palette. And the maximum `n` value when scheme colors are designed for gray-scale conversion is enclosed in parentheses. A value of infinity indicates that the scheme allows for color interpolations. **N**: is the not-a-number color. **B**: is the background color. **F**: is the foreground color. **Abbreviations**: –, not available]

Table 1. Scheme by Anton Mikhailov (2019); released under an open license.

Type	Scheme	Palette	Max n	N	B	F
Sequential	turbo		∞	–	–	–

Table 2. Schemes by Paul Tol (2018) with permission granted to distribute in Oct 2018.

Type	Scheme	Palette	Max n	N	B	F
Diverging	BuRd		∞		–	–
	PRGn		∞		–	–
	sunset		∞		–	–
Qualitative	bright		7 (3)	–	–	–
	dark		6	–	–	–
	ground cover		14	–	–	–
	high-contrast		5 (5)	–	–	–
	light		9	–	–	–
	muted		9 (5)	–	–	–
	pale		6	–	–	–
	vibrant		7 (4)	–	–	–
Sequential	discrete rainbow		23		–	–
	iridescent		∞		–	–
	smooth rainbow		∞		–	–
	YlOrBr		∞		–	–

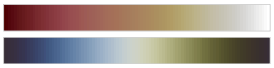
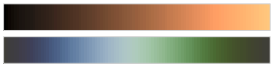
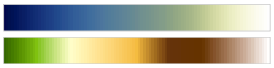
Table 3. Schemes by Thomas Dewez (2004) with permission granted to distribute in Oct 2018.

Type	Scheme	Palette	Max n	N	B	F
Sequential	DEM poster		∞			
	DEM print		∞			
	DEM screen		∞			

Table 4. Scheme by unknown author; discovered on gnuplot-info by Edzer Pebesma.

Type	Scheme	Palette	Max n	N	B	F
Sequential	bpy		∞	–	–	–

Table 5. Schemes collected by Wessel and others (2013) and released under an open license.

Type	Scheme	Palette	Max n	N	B	F
Diverging	polar		∞	–	–	–
	red2green		∞	–	–	–
	romaO		∞		–	–
	split		∞	–	–	–
Sequential	abyss		∞			
	acton		∞			
	bam		∞			
	bamako		∞			
	bamO		∞		–	–
	bathy		∞			
	batlow		∞			
	batlowK		∞			
	batlowW		∞			
	bilbao		∞			
	brocO		∞		–	–
	buda		∞			
	cool		∞	–	–	–
	copper		∞	–	–	–
	corkO		∞		–	–
	cubhelix		∞			
	davos		∞			
	dem1		∞			
	dem2		∞			
	dem3		∞			
	dem4		∞			
	devon		∞			
	drywet		∞	–	–	–
	elevation		∞			
	glasgow		∞			
	grayC		∞			
	hawaii		∞			
	haxby		∞	–	–	–
	hot		∞	–	–	–
	imola		∞			
	inferno		∞	–	–	–
	jet		∞	–	–	–
	lajolla		∞			
	lapaz		∞			
	lipari		∞			
	magma		∞	–	–	–
	navia		∞			
	nuuk		∞			
	ocean		∞	–		
	oslo		∞			
	plasma		∞	–	–	–
	seafloor		∞	–	–	–
	seis		∞	–	–	–
	tokyo		∞			
	turku		∞			
	vanimo		∞			
	vikO		∞		–	–
	viridis		∞	–	–	–
	wysiwyg		∞	–	–	–

Schemes "pale", "dark", and "ground cover" are intended to be accessed in their entirety and subset using vector element names.

Value

When argument n is specified, the function returns an object of class 'inlpal'. When n is unspecified a variant of the get_colors function is returned that has default argument values set equal to the values specified by the user.

Note

Sequential color schemes "YlOrBr" and "iridescent" work well for conversion to gray scale.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

References

Dewez, Thomas, 2004, Variations on a DEM palette, accessed October 15, 2018 at http://soliton.vm.bytemark.co.uk/pub/cpt-city/td/index.html

Mikhailov, Anton, 2019, Turbo, an improved rainbow colormap for visualization: Google AI Blog, accessed August 21, 2019 at https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html.

Tol, Paul, 2018, Colour Schemes: SRON Technical Note, doc. no. SRON/EPS/TN/09-002, issue 3.1, 20 p., accessed September 24, 2018 at https://personal.sron.nl/~pault/data/colourschemes.pdf.

Wessel, P., Smith, W.H.F., Scharroo, R., Luis, J.F., and Wobbe, R., 2013, Generic Mapping Tools: Improved version released, AGU, v. 94, no. 45, p. 409–410 doi:10.1002/2013EO450001.

See Also

plot method for drawing color palettes.

set_hinge function to set the hinge location in a color palette derived from one or two color schemes.

grDevices::col2rgb function to express palette colors represented in the hexadecimal format as RGB triplets (R, G, B).

Examples

```
pal <- get_colors(n = 10)
print(pal)
plot(pal)

get_pal <- get_colors(scheme = "turbo")
formals(get_pal)
filled.contour(datasets::volcano,
  color.palette = get_pal,
  plot.axes = FALSE
```

```
)

# Diverging color schemes (scheme)
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
get_colors(9, scheme = "BuRd") |> plot()
get_colors(255, scheme = "BuRd") |> plot()
get_colors(9, scheme = "PRGn") |> plot()
get_colors(255, scheme = "PRGn") |> plot()
get_colors(11, scheme = "sunset") |> plot()
get_colors(255, scheme = "sunset") |> plot()
par(op)

# Qualitative color schemes (scheme)
op <- par(mfrow = c(7, 1), oma = c(0, 0, 0, 0))
get_colors(7, scheme = "bright") |> plot()
get_colors(6, scheme = "dark") |> plot()
get_colors(5, scheme = "high-contrast") |> plot()
get_colors(9, scheme = "light") |> plot()
get_colors(9, scheme = "muted") |> plot()
get_colors(6, scheme = "pale") |> plot()
get_colors(7, scheme = "vibrant") |> plot()
par(op)

# Sequential color schemes (scheme)
op <- par(mfrow = c(7, 1), oma = c(0, 0, 0, 0))
get_colors(23, scheme = "discrete rainbow") |> plot()
get_colors(34, scheme = "smooth rainbow") |> plot()
get_colors(255, scheme = "smooth rainbow") |> plot()
get_colors(9, scheme = "YlOrBr") |> plot()
get_colors(255, scheme = "YlOrBr") |> plot()
get_colors(23, scheme = "iridescent") |> plot()
get_colors(255, scheme = "iridescent") |> plot()
par(op)

# Alpha transparency (alpha)
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
get_colors(34, alpha = 1.0) |> plot()
get_colors(34, alpha = 0.8) |> plot()
get_colors(34, alpha = 0.6) |> plot()
get_colors(34, alpha = 0.4) |> plot()
get_colors(34, alpha = 0.2) |> plot()
par(op)

# Color stops (stops)
op <- par(mfrow = c(4, 1), oma = c(0, 0, 0, 0))
get_colors(255, stops = c(0.0, 1.0)) |> plot()
get_colors(255, stops = c(0.0, 0.5)) |> plot()
get_colors(255, stops = c(0.5, 1.0)) |> plot()
get_colors(255, stops = c(0.3, 0.9)) |> plot()
par(op)

# Interpolation bias (bias)
op <- par(mfrow = c(7, 1), oma = c(0, 0, 0, 0))
```

```
get_colors(255, bias = 0.4) |> plot()
get_colors(255, bias = 0.6) |> plot()
get_colors(255, bias = 0.8) |> plot()
get_colors(255, bias = 1.0) |> plot()
get_colors(255, bias = 1.2) |> plot()
get_colors(255, bias = 1.4) |> plot()
get_colors(255, bias = 1.6) |> plot()
par(op)

# Reverse colors (reverse)
op <- par(
  mfrow = c(2, 1),
  oma = c(0, 0, 0, 0),
  cex = 0.7
)
get_colors(10, reverse = FALSE) |> plot()
get_colors(10, reverse = TRUE) |> plot()
par(op)

# Color blindness (blind)
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
get_colors(34, blind = NULL) |> plot()
get_colors(34, blind = "deutan") |> plot()
get_colors(34, blind = "protan") |> plot()
get_colors(34, blind = "tritan") |> plot()
get_colors(34, blind = "monochrome") |> plot()
par(op)

# Gray-scale preparation (gray)
op <- par(mfrow = c(8, 1), oma = c(0, 0, 0, 0))
get_colors(3, "bright", gray = TRUE) |> plot()
get_colors(3, "bright", gray = TRUE, blind = "monochrome") |> plot()
get_colors(5, "high-contrast", gray = TRUE) |> plot()
get_colors(5, "high-contrast", gray = TRUE, blind = "monochrome") |> plot()
get_colors(4, "vibrant", gray = TRUE) |> plot()
get_colors(4, "vibrant", gray = TRUE, blind = "monochrome") |> plot()
get_colors(5, "muted", gray = TRUE) |> plot()
get_colors(5, "muted", gray = TRUE, blind = "monochrome") |> plot()
par(op)
```

`plot.inlpal` *Plot method for color palettes*

Description

Plot a sequence of shaded rectangles showing colors in the palette.

Usage

```
## S3 method for class 'inlpal'
plot(x, ..., label = TRUE)
```

Arguments

x	'inlpal' object that inherits behavior from the 'character' class. Palette colors represented in hexadecimal format.
...	Not used
label	'logical' flag. Whether to include the plot title.

Value

Invisibly returns NULL, called for side effect, plotting a color palette in an R graphics device.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

Examples

```
get_colors(10) |> plot()

set_hinge(c(-10, 10), hinge = 0)(20) |> plot()
```

---

set_hinge *Set hinge location in color palette*

Description

The *hinge* indicates a dramatic color change in a palette that is typically located at the midpoint of the data range. An asymmetrical data range can result in an undesired hinge location, a location that does not necessarily coincide with the break-point in the user's data. This function can be used to specify a hinge location that is appropriate for your data.

Usage

```
set_hinge(
  x,
  hinge,
  scheme = "sunset",
  alpha = NULL,
  reverse = FALSE,
  buffer = 0,
  stops = c(0, 1),
  allow_bias = TRUE,
  nan = NA_character_
)
```

Arguments

x	'numeric' object that can be passed to the range function with NA's removed. The user's data range.
hinge	'numeric' number. Hinge value (such as, at sea-level) in data units.
scheme	'character' vector of length 1 or 2, value is recycled as necessary. Name of color scheme(s). The color palette is derived from one or two color schemes. The scheme(s) must be suitable for continuous data types and allow for color interpolation. See get_colors function for a list of possible scheme names. Argument choices may be abbreviated as long as there is no ambiguity.
alpha	'numeric' vector of length 1 or 2, value is recycled as necessary. Alpha transparency applied separately on either side of the hinge. Values range from 0 (fully transparent) to 1 (fully opaque). Specify as NULL to exclude the alpha channel value from colors.
reverse	'logical' vector of length 1 or 2, value is recycled as necessary. Whether to reverse the order of colors in the scheme(s). Values applied separately on either side of the hinge.
buffer	'numeric' vector of length 1 or 2, value is recycled as necessary. Color buffer around the hinge measured as a fraction of the color range. Values applied separately on either side of the hinge.
stops	'numeric' vector of length 2. Color stops defined by interval endpoints (between 0 and 1) and used to select a subset of the color palette(s).
allow_bias	'logical' flag. Whether to allow bias in the color spacing.
nan	'character' string. Color meant for missing data, in hexadecimal format, where NA indicates no color is specified.

Value

A 'function' that takes an 'integer' argument (the required number of colors) and returns a vector of colors of class 'inlpal'.

Author(s)

J.C. Fisher, U.S. Geological Survey, Idaho Water Science Center

See Also

plot method for drawing color palettes.

Examples

```
f <- set_hinge(c(-3, 7), hinge = 0)
f(n = 19) |> plot()

x <- datasets::volcano
f <- set_hinge(x, hinge = 140, scheme = c("abyss", "dem1"))
filled.contour(x, color.palette = f, nlevels = 50, plot.axes = FALSE)
```

```
# Data range (x)
hinge <- 0
n <- 20
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
set_hinge(c(-10, 0), hinge)(n) |> plot()
set_hinge(c(-7, 3), hinge)(n) |> plot()
set_hinge(c(-5, 5), hinge)(n) |> plot()
set_hinge(c(-3, 7), hinge)(n) |> plot()
set_hinge(c(0, 10), hinge)(n) |> plot()
par(op)

# Hinge value (hinge)
x <- c(-5, 5)
n <- 255
op <- par(mfrow = c(5, 1), oma = c(0, 0, 0, 0))
set_hinge(x, hinge = -6)(n) |> plot()
set_hinge(x, hinge = -2)(n) |> plot()
set_hinge(x, hinge = 0)(n) |> plot()
set_hinge(x, hinge = 2)(n) |> plot()
set_hinge(x, hinge = 6)(n) |> plot()
par(op)

# Color scheme (scheme)
x <- c(-10, 10)
hinge <- -3
n <- 255
op <- par(mfrow = c(3, 1), oma = c(0, 0, 0, 0))
set_hinge(x, hinge, scheme = "romaO")(n) |> plot()
set_hinge(x, hinge, scheme = "BuRd")(n) |> plot()
set_hinge(x, hinge, scheme = c("ocean", "copper"))(n) |> plot()
par(op)

# Alpha transparency (alpha)
x <- c(-5, 5)
hinge <- 0
scheme <- c("drywet", "hawaii")
n <- 255
op <- par(mfrow = c(4, 1), oma = c(0, 0, 0, 0))
set_hinge(x, hinge, scheme, alpha = 1.0)(n) |> plot()
set_hinge(x, hinge, scheme, alpha = 0.5)(n) |> plot()
set_hinge(x, hinge, scheme, alpha = c(1.0, 0.5))(n) |> plot()
set_hinge(x, hinge, scheme, alpha = c(0.5, 1.0))(n) |> plot()
par(op)

# Reverse colors (reverse)
x <- c(-10, 10)
hinge <- -3
n <- 255
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
set_hinge(x, hinge, "romaO", reverse = FALSE)(n) |> plot()
set_hinge(x, hinge, "romaO", reverse = TRUE)(n) |> plot()
set_hinge(x, hinge, c("davos", "hawaii"), reverse = FALSE)(n) |> plot()
set_hinge(x, hinge, c("davos", "hawaii"), reverse = TRUE)(n) |> plot()
```

```
set_hinge(x, hinge, c("davos", "hawaii"), reverse = c(TRUE, FALSE))(n) |> plot()
set_hinge(x, hinge, c("davos", "hawaii"), reverse = c(FALSE, TRUE))(n) |> plot()
par(op)

# Buffer around hinge (buffer)
x <- c(-5, 5)
hinge <- -2
n <- 20
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
set_hinge(x, hinge, buffer = 0.0)(n) |> plot()
set_hinge(x, hinge, buffer = 0.2)(n) |> plot()
set_hinge(x, hinge, buffer = c(0.4, 0.2))(n) |> plot()
set_hinge(x, hinge, c("gray", "plasma"), buffer = 0.0)(n) |> plot()
set_hinge(x, hinge, c("gray", "plasma"), buffer = 0.2)(n) |> plot()
set_hinge(x, hinge, c("gray", "plasma"), buffer = c(0.2, 0.4))(n) |> plot()
par(op)

# Color stops (stops)
x <- c(-5, 5)
hinge <- 1
n <- 20
op <- par(mfrow = c(6, 1), oma = c(0, 0, 0, 0))
set_hinge(x, hinge, stops = c(0.0, 1.0))(n) |> plot()
set_hinge(x, hinge, stops = c(0.2, 0.8))(n) |> plot()
set_hinge(x, hinge, stops = c(0.4, 0.6))(n) |> plot()
set_hinge(x, hinge, c("gray", "plasma"), stops = c(0.0, 1.0))(n) |> plot()
set_hinge(x, hinge, c("gray", "plasma"), stops = c(0.2, 0.8))(n) |> plot()
set_hinge(x, hinge, c("gray", "plasma"), stops = c(0.4, 0.6))(n) |> plot()
par(op)

# Allow bias (allow_bias)
x <- c(-3, 7)
n <- 20
op <- par(mfrow = c(4, 1), oma = c(0, 0, 0, 0))
set_hinge(x, hinge = 0, allow_bias = TRUE)(n) |> plot()
set_hinge(x, hinge = 0, allow_bias = FALSE)(n) |> plot()
set_hinge(x, hinge = 4, allow_bias = TRUE)(n) |> plot()
set_hinge(x, hinge = 4, allow_bias = FALSE)(n) |> plot()
par(op)
```

Index

∗ **color**
    get_colors, 2
    set_hinge, 9

dichromat::dichromat, *3*

get_colors, 2, *10*
grDevices::col2rgb, *6*

inlpal, *6*, *10*

plot, *6*, *10*
plot.inlpal, 8

range, *10*

set_hinge, *6*, 9