

Package ‘barsurf’

August 28, 2020

Title Multivariate Function Visualization and Smooth Multiband Color Interpolation

Version 0.6.1

Date 2020-08-28

License GPL (>= 2)

Maintainer Abby Spurdle <spurdle.a@gmail.com>

Author Abby Spurdle

URL <https://sites.google.com/site/spurdlea/r>

Description Combined contour/heatmap plots, 3d bar/surface plots, isosurface plots, triangular plots and 2d/3d vector fields. Builds on the colorspace package (Zeileis, A., et al. (2019) <arxiv.org/abs/1903.06490>), by supporting smooth multi-band color interpolation, in sRGB, HSV and HCL color spaces. The default color functions for heatmaps, use HCL (Zeileis, A., et al. (2009) <doi:10.1016/j.csda.2008.11.033>) and support near-uniform perception.

Depends methods

Imports kubik, colorspace

Suggests misc3d

Contact Primary <spurdle.a@gmail.com>, Secondary <spurdle.a@aol.com>

NeedsCompilation no

Repository CRAN

Date/Publication 2020-08-28 07:10:06 UTC

R topics documented:

11_main_plotting_functions	2
12_functional_versions	13
21_global_options	16
22_standardized_palette_interface	18
31_litmus_objects	19
32_multi-litmus_objects	21
33_litmus-fitting_functions	22

41_supporting_methods	24
51_simple_color_palettes	25
52_predefined_litmus_objects	26
53_predefined_litmus-fitting_functions	27
54_color_sequence_functions	29
61_check_xy_relationship	31
62_matrix_plot_margins	32
63_color_conversion	32
71_deprecated	33
81_sample_functions	34

Index 35

11_main_plotting_functions
Main Plotting Functions

Description

Selected multivariate plots, including:

Combined contour-heatmap plots, 3d bar and surface plots, triangular plots, isosurface plots, 3d-based contour-heatmap plots, and 2d/3d vector fields.

Note that there are wrappers for most of these plotting functions, which take a mathematical function as their main argument.

Refer to [Functional Versions](#).

Usage

```
#####
#combined contour-heatmap plots
#####
plot_dfield (x, y, fv, ..., fb,
             main="", xlabel="x", ylabel="y",
             contours=TRUE, grid=FALSE, heatmap=TRUE,
             contour.labels=FALSE, bin.labels=FALSE,
             add=FALSE, axes=TRUE, reverse=FALSE, swap.sides=FALSE,
             xyrel, continuous.axes=FALSE,
             ncontours=2, clabs, blabs,
             xat, yat, xlabs, ylabs,
             contour.color="#000000", grid.color="#888888",
             colf, colff, colors, hcv=FALSE)

plot_cfield (x, y, fv, ..., fb,
             main="", xlabel="x", ylabel="y",
             contours=TRUE, heatmap=TRUE, contour.labels=FALSE,
             add=FALSE, axes=TRUE, reverse=FALSE, swap.sides=FALSE,
             ncontours=6, clabs,
```

```
xyrel, xat, yat, xlabs, ylabs,
contour.color="#000000", colf, colff, hcv=FALSE)

#####
#3d plots
#2 variables, f (x, y)
#####
plot_bar (x, y, fv, ...,
          main="", xlab="x", ylab="y", zlab="z",
          axes=TRUE, z.axis=FALSE,
          ref.arrows = opt.ref.arrows (), reverse=FALSE,
          continuous.axes=FALSE,
          zlim, xat, yat, xlabs, ylabs,
          nhl = opt.nhl (),
          top.color = opt.top.color (), side.color = opt.side.color () )

plot_surface (x, y, fv, ...,
              main="", xlab="x", ylab="y", zlab="z",
              grid=TRUE, gradient.shading=TRUE,
              axes=TRUE, z.axis=FALSE,
              ref.arrows = opt.ref.arrows (), reverse=FALSE,
              zlim, xat, yat, xlabs, ylabs,
              nhl = opt.nhl (),
              grid.color = opt.sgrid.color (),
              colf, colff)

#####
#triangular plots
#####
plot_tricontour (x, y, fv, ..., fb,
                 main="", xlab="x", ylab="y",
                 contours=TRUE, heatmap=TRUE, contour.labels=FALSE,
                 axes=TRUE, ncontours=6, clabs, xyrel="s",
                 xat, yat, xlabs, ylabs,
                 contour.color="#000000", colf, colff, hcv=FALSE)

plot_trisurface (x, y, fv, ...,
                 main="", xlab="x", ylab="y", zlab="z",
                 grid=TRUE,
                 axes=TRUE, z.axis=FALSE,
                 ref.arrows = opt.ref.arrows (),
                 zlim,
                 xat, yat, xlabs, ylabs,
                 nhl = opt.nhl (),
                 grid.color = opt.sgrid.color (),
                 colf, colff)

#####
```

```

#3d plots
#3 variables, f (x, y, z)
#####
plot_isosurface (x, y, z, fv, ..., fb, fq,
  main="", xlab="x", ylab="y", zlab="z",
  wire.frame=FALSE, texture=TRUE,
  axes=TRUE, z.axis = all (axes),
  ref.arrows = opt.ref.arrows (), z.ref.arrow = any (ref.arrows),
  reverse=FALSE, z.reverse=FALSE,
  nsurfaces=2,
  nested=TRUE, maximal,
  xat, yat, xlabs, ylabs,
  nhl = opt.nhl (),
  wire.color="#808080", iso.colors = opt.iso.colors () )

plot_cfield3 (x, y, z, fv, ..., fb,
  main="", xlab="x", ylab="y", zlab="z",
  contours=TRUE, heatmap=TRUE,
  axes=TRUE, ref.arrows = opt.ref.arrows (),
  slide.labels = any (axes),
  reverse=FALSE, z.reverse=FALSE,
  ncontours=6, emph="n",
  xat, yat, xlabs, ylabs, zlabs,
  nhl = opt.nhl (),
  colf, colff)

#####
#matrix visualization
#####
plot_matrix (x, y, fv, ..., transpose=TRUE,
  xlab, ylab,
  contours=FALSE, grid, bin.labels,
  reverse = c (FALSE, transpose), swap.sides=reverse,
  blabs, colors,
  hcv=TRUE)

#####
#vector fields
#####
plot_vec2 (x, y, dx, dy, ...,
  main="", xlab="x", ylab="y",
  vectors=TRUE, heatmap=TRUE,
  include.outermost.vectors=FALSE,
  add=FALSE, axes=TRUE, reverse=FALSE, swap.sides=FALSE,
  xyrel, xat, yat, xlabs, ylabs,
  arrowh.length=1.75, arrowh.width = 0.75 * arrowh.length,
  arrow.color="#000000", fill.color="#08080810",
  colf, colff, hcv=FALSE)

```

```

plot_vec3 (x, y, z, dx, dy, dz, ..., P = 1 / 3,
  main="", xlab="x", ylab="y", zlab="z",
  include.outermost.vectors=FALSE,
  axes=TRUE, z.axis = all (axes),
  ref.arrows = opt.ref.arrows (), z.ref.arrow = any (ref.arrows),
  xat, yat, xlabs, ylabs,
  nhl = opt.nhl (),
  head.color="#0000B0", mid.color="#8080FF", tail.color="#B0B0B0")

```

Arguments

x, y, z

In plot_dfield, plot_bar and plot_matrix:

Optional sorted numeric vectors of bin midpoints or breakpoints.
(Refer to the details section for plot_dfield).

In plot_cfield, plot_surface, plot_vec2 and plot_vec3:

Optional sorted numeric vectors of grid point coordinates.
Note, I recommend that the lengths of x and y are different for 3d vector fields.

In plot_tricontour and plot_trisurface:

Ignored.

In plot_isosurface:

Same as plot_cfield, except that:

- (1) They can also be iso-lists of such vectors.
- (2) They're required, if fv is an iso-list, and the corresponding array dimension, has different sizes.

Refer to the details section for plot_isosurface.

In plot_cfield3:

Same as plot_cfield, except that z is the vertical slide/slice position.

Note that where x, y and z are missing, they default to:
1:nrows, 1:ncols, 1:nsheets (the third array dimension) or 1:nslides.

fv

In plot_dfield, plot_bar and plot_matrix:

A numeric matrix of function (bin) values.
By default, if fv has row/column names, they will be used.

In plot_cfield and plot_surface:

A numeric matrix of function (point) values.

In plot_tricontour and plot_trisurface:

A numeric square matrix of function values.
Only the top-left triangle is used, including the diagonal.

In plot_isosurface:

A numeric 3d array, or an iso-list of such arrays.
(Refer to the details section for plot_isosurface).

In plot_cfield3:

A list of numeric matrices of function values.
Each matrix corresponds to one z value.

Note that in discretely-spaced plots, fv may contain missing values.
In other plots, missing values are not allowed.

dx, dy, dz

In plot_vec2:

Numeric matrices, of equal size, giving the x and y components of a vector field.

In plot_vec3:

Numeric 3d arrays, of equal size, giving the x, y and z components of a vector field.

In plot_vec2, they may include missing values.

Note, I recommend that x-dim and y-dim are different for 3d vector fields.
(i.e. $\dim(dx)[1] \neq \dim(dx)[2]$).

This is to minimize possible over plotting.

fb

Optional numeric vector of function values (or "levels"), for contours/isosurfaces.

fq

Alternative way of specifying fb, above.

Numeric vector of probabilities (between zero and one), for quantiles of fv.

P

Numeric, between zero and one, giving the proportion of (highest magnitude) vectors to plot.

main, xlab, ylab, zlab

Optional strings, main/axes titles.

transpose

Logical, if true, transpose fv, along with the any bin labels and colors.

contours

Logical, include contour lines.

grid

Logical, include grid lines.

For plot_matrix, defaults to true, if both the number of rows and columns don't exceed twenty.

wire.frame

Logical, include the wire.frame lines.

gradient.shading

Logical, use gradient shading.

vectors

Logical, include vector arrows.

heatmap

Logical, include heatmap.

contour.labels

Logical, include contour labels.

slide.labels

Logical, include the slide labels.

bin.labels

Logical, include bin labels.

For plot_matrix, defaults to true, if both the number of rows and columns don't exceed ten.

include.outermost.vectors	Logical, plot all vector arrows. Otherwise (the default), exclude the outermost rows and columns.
texture	Logical, if true, add a small amount of random variation to isosurface colors.
add	Logical, if true, add contours/heatmap to an existing plot.
axes	Logical vector of length one or two, if true, plot reference arrows or axis ticks with labels.
z.axis	Logical, if true, include a z axis.
ref.arrows	Logical vector of length one or two, if true, plot reference arrows, if false plot axis labels.
z.ref.arrow	Logical, if true, plot a z reference arrow, if false, plot z axis labels.
reverse	Logical vector of length one or two, if true, reverse the x and y axes.
z.reverse	Logical, if true, reverse the z axis.
swap.sides	Logical vector of length one or two, if true, swap the axes.
xyrel	Single character, either "f", "s" or "m". "f" produces a plot with a fixed aspect ratio of one, "s" produces a square plot, and "m" a maximized plot. The default is determined by the test.xyrel function.
continuous.axes	Logical of length one or two. If true, plot the axes in the same way as continuous plots.
zlim	Optional length-2 numeric vector, the vertical range, corresponding to function value.
ncontours, nsurfaces	Integer, the number of contours/surfaces. Ignored, if fb or fq supplied.
nested, maximal	nested Logical value, if true (the default), assume that the isosurfaces are nested. maximal Logical value, if true, assume that (nested) isosurfaces are centered around local maxima, if false, assume that they're centered around local minima. These determine the default fb values, including their order. (Ascending for low-valued focal points, and descending for high-valued focal points).
emph	What to emphasize: "n" (for nothing). "b" for both (low and high regions). "l" for low regions. "h" for high regions. "B", "L" or "H" for the same but with a stronger effect.

<code>clabs</code>	Optional character vector of contour labels. Ignored, unless <code>contour.labels</code> is true.
<code>blabs</code>	Optional character matrix of bin labels. Its dimensions need to match <code>fv</code> . Ignored, unless <code>bin.labels</code> is true. Defaults to the values of <code>fv</code> .
<code>xat, yat</code>	Optional numeric vectors, the x and y axes tick/label points.
<code>xlabs, ylabs</code>	Optional character vectors, the x and y axes tick labels. In 3d plots, ignored unless <code>axes</code> is true and <code>arrows</code> is false.
<code>zlabs</code>	Optional character vectors, the slide labels.
<code>nh1</code>	Integer, the number of header lines.
<code>arrowh.length, arrowh.width</code>	Numerics, arrow head length and width, in mm.
<code>contour.color</code>	Character vector (or R color strings) of length one or with the same length as the number of contours, giving the contour line colors.
<code>grid.color</code>	String (R color string), giving the grid line color.
<code>wire.color</code>	String (R color string), giving the wire frame color.
<code>colf</code>	A litmus object (or equivalent color function), refer to details section for color functions.
<code>colff</code>	A litmus-fitting function, refer to details section for color functions.
<code>colors</code>	Optional character matrix (R color strings), for the bin colors. Its dimensions need to match <code>fv</code> .
<code>top.color, side.color</code>	Strings (R color strings) giving the top and side colors of bars. Also, can be character matrices. If matrices, then their dimensions need to match <code>fv</code> .
<code>iso.colors</code>	Character vector (R color strings), giving the (initial) isosurface colors.
<code>arrow.color, fill.color</code>	Strings (R color strings), arrow line and fill colors.
<code>head.color, mid.color, tail.color</code>	Strings (R color strings), the arrow colors.
<code>hcv</code>	Logical, if true, use the high color variation option.
<code>...</code>	Ignored.

Details

FUNCTIONS OF TWO DISCRETE VARIABLES (AND DISCRETELY-SPACED SCALAR FIELDS) (`plot_dfield` and `plot_bar`)

In `plot_dfield` and `plot_bar`, the main argument is `fv`, which is a numeric matrix of function values. (Each `fv` value is a bin value).

Increasing row indices (i.e. moving down the matrix) correspond to increasing x values, and increasing column indices (i.e. moving right across the matrix) correspond to increasing y values.

x and y arguments, are numeric vectors of midpoints or breakpoints, of the bins.

If midpoints, then the length of x needs to be equal to the number of rows and the length of y needs to equal the number of columns. If breakpoints, then the length of x needs to be equal to the number of rows plus one, and the length of y needs to equal the number of columns plus one.

FUNCTIONS OF TWO CONTINUOUS VARIABLES (AND CONTINUOUSLY-SPACED SCALAR FIELDS)

(plot_cfield and plot_surface)

In plot_cfield and plot_surface, the main argument is fv, which is also a numeric matrix of function values.

(However, each fv value is a point value, and the resulting heatmap is for interpolated bin values).

As with the discretely spaced case, increasing row indices (i.e. moving down the matrix) correspond to increasing x values, and increasing column indices (i.e. moving right across the matrix) correspond to increasing y values.

However, optional x and y arguments, give the coordinates of the points.

The length of x needs to be equal to the number of rows, and the length of y needs to equal the number of columns.

TRIANGULAR PLOTS

(plot_tricontour and plot_trisurface)

These functions are similar to plot_cfield and plot_surface, except that the fv matrix, needs to be square, and the x and y coordinates are ignored.

ISOSURFACE PLOTS

(plot_isosurface)

THIS FUNCTION REQUIRES THE misc3d PACKAGE TO BE INSTALLED AND LOADED
(There's no requirement for the package to be attached).

The number of isosurfaces is determined by nsurfaces.

Or by fb or fq, if provided.

The order of fb is important.

If the isosurfaces are nested, and the default colors are used, then the fb values should be in order of the innermost to the outermost.

This package supports two types of isosurface plots, simple isosurface plots, and multi-array isosurface plots.

For simplicity, an "iso-list" is shorthand, for a list, with the same length as the number of isosurfaces.

SIMPLE ISOSURFACE PLOTS

In simple isosurface plots, `fv` is a numeric 3d array of function values.

Increasing along the first array dimension corresponds to increasing `x` values, increasing along the second array dimension corresponds to increasing `y` values, and increasing along the third array dimension corresponds to increasing `z` values.

There are optional `x`, `y` and `z` arguments, which are numeric vectors, giving the coordinates of the points.

The length of `x` needs to be equal the `x`-dim size, the length of `y` needs to equal the `y`-dim size, and the length of `z` needs to equal the `z`-dim size, where `x`-dim, `y`-dim and `z`-dim refer to `dim (fv)[1]`, `dim (fv)[2]` and `dim (fv)[3]`, respectively.

MULTI-ARRAY ISOSURFACE PLOTS

In multi-array isosurface plots, `fv` is an iso-list of numeric 3d arrays.

If the number of arrays is not equal to two (the default value for `nsurfaces`), then you will need to set `nsurfaces`, `fb` or `fq`.

`x`, `y`, `z` can be numeric vectors or iso-lists of numeric vectors.

If coordinates are iso-lists and `fv` is an iso-list, then their sizes need to match, surface-wise. i.e.

length (`x [[k]]`) needs to equal `dim (fv [[k]])[1]`, for all `k`

length (`y [[k]]`) needs to equal `dim (fv [[k]])[2]`, for all `k`

length (`z [[k]]`) needs to equal `dim (fv [[k]])[3]`, for all `k`

where `k` is the `k`th isosurface.

OTHER 3D-BASED CONTOUR PLOTS AND HEATMAPS

(`plot_cfield3`)

`plot_slides` is similar to `plot_cfield`, but produces a set of overlapping slides/slices.

Here, `fv` is a list of numeric matrices.

(One matrix for each slide).

And `z`, is the coordinate vector for the slides/slices.

(One `z` value for each slide, and the corresponding `fv` matrix).

The function contains an `emph` (emphasis) argument.

Note that setting `heatmap` to `false`, will increase the rendering speed.

MATRIX VISUALISATION

(`plot_matrix`)

The `plot_matrix` function is a wrapper for `plot_dfield`.

By default, it transposes the matrix, and then calls `plot_dfield` with different default values.

This causes the resulting plots to have the same orientation as matrices, in textual form.

Note that in principle, this transposes the matrix, twice.

Because `plot_dfield` transposes the matrix, in terms of visual interpretation.

Without transposed input, increasing row indices (going down) map to increasing x values (going right).

Note that if bin labels or bin colors are supplied, they're also transposed.

All other arguments (x, y, etc) apply to the transposed matrix, not the original.

2D VECTOR FIELDS

(plot_vec2)

plot_vec2 is similar to plot_cfield, except that there are two dx and dy matrices, instead of a single fv matrix.

The color of the heatmap bins reflects the magnitude of the four surrounding vectors.

3D VECTOR FIELDS

(plot_vec3)

plot_vec3 is similar to plot_vec2.

Instead of two numeric matrices, we have three numeric 3d arrays, including dz.

Increasing along the first array dimension corresponds to increasing x values, increasing along the second array dimension corresponds to increasing y values, and increasing along the third array dimension corresponds to increasing z values.

Note that this function has had less testing than other plotting functions in this package.

3D COORDINATE SYSTEM

Currently, 3d plots use a diamond-like orthographic/dimetric projection, with a fixed viewing angle.

By default, (min (x), min (y)) is at the bottom center. And increasing x values, run diagonally up-right, and increasing y value run diagonally up-left.

It's not possible to rotate the plots, but the x and y axis can be reversed.

In functions of two variables, the z coordinate represents the function value, and increasing z values run up.

COLOR FUNCTIONS

The color function (colf) can be any function that maps a numeric vector/matrix (as its first argument) to a character vector/matrix of R colors, however, I recommend using litmus objects. The litmus-fitting function (colff), is a function that maps a numeric vector (also, as its first argument) to a valid color function, however, I recommend using a function that returns a litmus object.

Assuming that all `colf`, `colff` and `colors` are missing, a litmus-fitting function is determined by global options, then the litmus object is fitted to the bin values, then the colors are computed from the litmus object.

In some cases, it may be useful to wrap a predefined litmus-fitting function.

e.g. To use nonstandard hot and cold colors, wrap the `hot.and.cold.fit` function, then set the `colff` argument to the wrapper function.

Opaque color functions are preferable in 2d heatmaps.

NOTES

If using `plot_bar`, `plot_surface` or `plot_trisurface` to plot constant or near-constant values, set `zlim`, for a better result.

Noting that functions that are constant in theory, may produce non-constant values when computed via floating point arithmetic.

In applied mathematics and applied statistics, you may want to set `z.axis=TRUE` and `ref.arrows=FALSE` (in 3d plots), and `xyrel="m"` (in 2d plots).

The default for reference arrows, can also be set via global options.

Often, it may be useful to create wrapper functions, that set the default argument values, accordingly.

There's no guarantee that default contour lines will be suitable.

References

Refer to the vignette for an overview, references and better examples.

See Also

[Functional Versions](#)

Which take a function as their main argument.

[Global Options](#)

[litmus](#), [litmus.fit](#)

[Predefined Litmus Objects](#), [Predefined Litmus-Fitting Functions](#)

With predefined colors.

Examples

```
#NOTE THAT THERE ARE MORE EXAMPLES IN:
#(1) THE PACKAGE VIGNETTE
#(2) THE HELP PAGE FOR FUNCTIONAL VERSIONS

#discretely-spaced example
fv <- matrix (1:100, 10, 10, byrow=TRUE)

plot_dfield (,fv)
plot_bar (,fv)
plot_matrix (,fv)
```

```
#continuously-spaced example
x <- y <- seq (-15.5, 15.5,, 40)
fv <- outer (x, y, rotated.sinc)

plot_cfield (x, y, fv, fb=0, hcv=TRUE)
plot_surface (x, y, fv)
```

12_functional_versions

Plotting Functions for Function Objects

Description

These functions wrap the main plotting functions.

They take a suitable function object, along with xlim/ylim/etc, and compute the x/y/fv/etc values.

Refer to [Main Plotting Functions](#).

Usage

```
plotf_dfield (f, xlim, ylim=xlim, ...)
plotf_bar (f, xlim, ylim=xlim, zlim, ...)

plotf_cfield (f, xlim, ylim=xlim, ..., n=30, nc = max (n, 60), nh=n)
plotf_surface (f, xlim, ylim=xlim, zlim, ..., n=30)

plotf_tricontour (f, ..., n=30)
plotf_trisurface (f, ..., n=30)

plotf_isosurface (f, xlim, ylim=xlim, zlim=xlim, ...,
  nsurfaces=2, fb, fq,
  nested=TRUE, maximal, panel.contours=TRUE,
  base.contours=panel.contours, rear.contours=panel.contours,
  pconstants,
  n)

plotf_cfield3 (f, xlim, ylim=xlim, zlim=xlim, ...,
  nslides=6, z.reverse=FALSE, z, n=30)

plotf_vec2 (vf, xlim, ylim=xlim, ..., n=20, nv=n, nh = max (n, 40) )
plotf_vec3 (vf, xlim, ylim=xlim, zlim=xlim, ..., n = c (20, 22, 16) )
```

Arguments

f In `plotf_dfield`, `plotf_bar`, `plotf_cfield` and `plotf_surface`, a numeric-valued function of two variables.
 In `plotf_tricontour` and `plotf_trisurface`, a numeric-valued function of two weight

	variables, that (along with a third implicit variable) sum to one. In <code>plotf_isosurface</code> and <code>plotf_cfield3</code> , a numeric-valued function of three variables.
<code>vf</code>	In <code>plotf_vec2</code> , a function of two variables that returns a two-column matrix, giving the dx and dy values. In <code>plotf_vec3</code> , a function of three variables that returns a three-column matrix, giving the dx, dy and dz values.
<code>xlim, ylim, zlim</code>	In <code>plotf_dfield</code> and <code>plotf_bar</code> , a length-two integer vector, giving the plotting ranges. In other plots, a length-two numeric vector, giving the plotting ranges. If descending, reverses the orientation of the plot. Note that in <code>plotf_cfield3</code> , <code>zlim</code> is ignored if <code>z</code> supplied. And currently, limits need to be ascending for 3d vector fields.
<code>nsurfaces, fb, fq</code>	The number of isosurfaces, and function values (or “levels”), for isosurfaces. Refer to <code>plotf_isosurface</code> for more information.
<code>nested, maximal</code>	<code>nested</code> Logical value, if true (the default), assume that the isosurfaces are nested. <code>maximal</code> Logical value, if true, assume that (nested) isosurfaces are centered around local maxima, if false, assume that they’re centered around local minima. These determine the default <code>fb</code> values, including their order. (Ascending for low-valued focal points, and descending for high-valued focal points). The nested argument also determines the default <code>n</code> values.
<code>panel.contours, base.contours, rear.contours</code>	Logical, include contour lines on the base or rear panels of the plot.
<code>pconstants</code>	Optional length-3 numeric vector of panel constants. Defaults to the midpoints of <code>xlim</code> , <code>ylim</code> and <code>zlim</code> . Refer to details and examples.
<code>nslides</code>	Integer, the number of slides. Ignored, if <code>z</code> supplied.
<code>z.reverse</code>	Logical, reverse the <code>z</code> axis. Ignored, unless <code>z</code> supplied. (i.e. If <code>z</code> is missing, <code>zlim</code> determines the vertical orientation).
<code>z</code>	Optional numeric vector, of <code>z</code> coordinates. If supplied, there will be one slide for each <code>z</code> value.
<code>n</code>	In most plots, an integer of length one or two, giving the number of bin/grid points, in each x/y direction. In <code>plotf_isosurface</code> and <code>plotf_vec3d</code> , an integer of length one or three, giving the number of grid points, in each x/y/z direction.

In `plotf_isosurface`, can also be a list of such vectors, one for each isosurface.

Note, I recommend that the `n` values for `x` and `y` be different, for 3d vector fields.

`nc, nv, nh` Same as `n`, but computes separate input matrices, for contour lines (`nc`), vector arrows (`nv`) and heatmaps (`nh`).

`...` Additional arguments for the main plotting functions.

Details

These functions wrap the main plotting functions.
i.e. `plotf_isosurface` calls `plot_isosurface`.

They take a suitable function object, along with `xlim/ylim/etc`, and compute the `x/y/fv/etc` values.
i.e. `plotf_surface` takes a numeric-valued function of two variables, a computes the `x` and `y` vectors, along with the `fv` matrix.

If you need to plot different functions to those described by the `f` argument above, it may be possible to create a wrapper function.

ISOSURFACE PLOTS

(`plotf_isosurface`)

THIS FUNCTION REQUIRES THE `misc3d` PACKAGE TO BE INSTALLED AND LOADED
(There's no requirement for the package to be attached).

If the nested argument is true, different `n` sizes are used for each isosurface.
The resolution of the `fv` array is highest for the first isosurface, and each `fv` array has a progressively lower resolution. Resulting plots should result in smaller file sizes, and render more quickly.

Optionally, 2D contour lines may be added to the base panel and rear panels.
This needs a vector of panel constants, which defaults to the `xlim/ylim/zlim` midpoints.
The function (of three variables), `f`, is called three times, holding one variable constant, each time.

The third constant is for the base panel contours.
And the first and second constants, are for the right-rear and left-rear panel contours, respectively.

References

Refer to the vignette for an overview, references and better examples.

See Also

[Main Plotting Functions](#)

[Global Options](#)

[litmus, litmus.fit](#)

[Predefined Litmus Objects, Predefined Litmus-Fitting Functions](#)

With predefined colors.

Examples

```
#####
#functions of two variables
#(discretely-spaced)
#####
df <- function (x, y, n=10, p=0.5)
  dbinom (x, n, p) * dbinom (y, n, p)

plotf_dfield (df, c (0, 10) )
plotf_bar (df, c (0, 10) )

#####
#functions of two variables
#(continuously-spaced)
#####
plotf_cfield (rotated.sinc, c (-15.5, 15.5), fb=0, n=40, hcv=TRUE)
plotf_surface (rotated.sinc, c (-15.5, 15.5), n=40)

#####
#triangular functions
#####
tf <- function (w1, w2, w3 = 1 - w1 - w2)
  (w1 - 1 / 3)^2 + (w2 - 1 / 3)^2 + (w3 - 1 / 3)^2

plotf_tricontour (tf, xlab="w1", ylab="w2")
plotf_trisurface (tf, xlab="w1", ylab="w2")

#####
#functions of three variables
#####
plotf_isosurface (bispherical.dist, c (-3, 3),, c (-2, 2) )
plotf_cfield3 (bispherical.dist, c (-3, 3),, c (-2, 2), emph="l")

#####
#vector fields
#####
plotf_vec2 (circular.field, c (-1.5, 1.5) )
plotf_vec3 (plughole.field, c (-1.5, 1.5),, c (-0.5, 1) )
```

Description

Set/get global options.

Usage

```
set.bs.options (...,
```

```
nhl=1, ref.arrows=TRUE,
rendering.style="R", theme="blue",
top.color, side.color, sgrid.color, iso.colors,
main, hcv, glass="glass.rainbow.fit", flow, lum,
test.mode=FALSE)
```

```
set.bs.nhl (nhl)
set.bs.theme (theme)
```

```
opt.nhl ()
opt.ref.arrows ()
```

Arguments

<code>nhl</code>	Integer, the number of header (main title) lines, in 3d plots.
<code>ref.arrows</code>	Logical vector of length one or two, if true plot reference arrows.
<code>rendering.style</code>	Single string, either "R" or "pdf", refer to details.
<code>theme</code>	String, either "gold", "blue", "green", "purple" or "heat".
<code>top.color, side.color</code>	Strings (R colors), the bar colors.
<code>sgrid.color</code>	String (R color), the surface grid color.
<code>iso.colors</code>	Character vector (R colors), the initial isosurface colors.
<code>main</code>	String, name of the main litmus-fitting function, which is the default for heatmaps, with <code>hcv=FALSE</code> .
<code>hcv</code>	String, name of the <code>hcv</code> (high color variation) litmus-fitting function, which is the default for heatmaps, with <code>hcv=TRUE</code> .
<code>glass</code>	String, name of the glass litmus-fitting function, which is the default for <code>plot_cfield3</code> .
<code>flow</code>	String, name of the flow litmus-fitting function, which is the default for vector fields.
<code>lum</code>	String, name of the lum litmus-fitting function, which is the default for surface plots.
<code>test.mode</code>	If true, and the main plotting functions are called with unmatched arguments, then an error is generated.
	This should be false, except for testing purposes.
<code>...</code>	Ignored.

Details

The set functions, set global options, and the opt functions gets them.

The "R" rendering style is for standard R graphic devices, and raster images.

The "pdf" rendering style is for PDF documents and other scalable vector formats.

The "pdf" option is the same as the "R" option, except that vector arrow lines, grid lines and wire frame lines are narrower.

However, the end result is dependent on the PDF viewer.

The blue and green themes are designed for relatively high perceptual uniformity.

And the heat theme is designed for high impact.

Calling the `set.bs.options` function will set all the options.

The default values of most color-based arguments are dependent on the theme.

In contrast, calling the `set.bs.nhl` functions will only change the `nhl` option, and calling the `set.bs.theme` function, will only change color-related options.

Note

Do not change global options, directly.

But rather, use `set.bs.options`, `set.nhl` or `set.theme` functions.

References

Refer to the vignette for an overview, references and better examples.

Examples

```
set.bs.options (nhl=2, ref.arrows=FALSE)
set.bs.theme ("gold")
```

22_standardized_palette_interface
Standardized Palette Interface

Description

Functions to get the default color objects.

Usage

```
st.top.color (theme)
st.side.color (theme)
st.sgrid.color (theme)
st.iso.colors (theme)

st.litmus.fit (theme)
st.litmus.fit.hcv (theme)
st.litmus.fit.flow (theme)
st.litmus.fit.lum (theme)

opt.top.color ()
opt.side.color ()
```

```
opt.sgrid.color ()  
opt.iso.colors ()  
  
opt.litmus.fit ()  
opt.litmus.fit.hcv ()  
opt.litmus.fit.glass ()  
opt.litmus.fit.flow ()  
opt.litmus.fit.lum ()
```

Arguments

theme String, the name of a supported theme.

Details

The st functions, return objects, for a particular theme.
And the opt functions, return objects, set by global options.

Value

The top, side and sgrid functions, return a single (R color) string.
The iso functions, return a character (R color) vector.
And the litmus.fit functions, return a litmus-fitting function.

References

Refer to the vignette for an overview, references and better examples.

See Also

[set.bs.options](#)
This provides information about themes.

Examples

```
st.litmus.fit ("blue")
```

31_litmus_objects *Litmus Objects*

Description

Color functions for heatmaps and surface plots.

Usage

```
#equally spaced knots
litmus (a=0, b=1, colvs, ...,
       color.space="sRGB", na.color="#FFFFFF")

#arbitrary knots
litmus.spline (cx, colvs, ...,
              color.space="sRGB", na.color="#FFFFFF")
```

Arguments

a, b	Numeric, the lower and upper limits.
cx	A numeric vector of knots (including the outermost values), which should be unique and sorted.
colvs	A 3-column or 4-column numeric matrix, where each row is one color vector, and the optional fourth column is alpha values.
color.space	A string giving the color space, refer to details.
na.color	A single string representing an R color.
...	Ignored.

Details

A litmus object maps a numeric vector to a character vector, representing R colors. (Noting that these functions return litmus objects, so you call these functions, and then if necessary you can evaluate the resulting function).

Color vectors are mapped from the input color space into sRGB color space. And a set of cubic Hermite splines interpolates over each component.

Input color spaces include "XYZ", "RGB", "LAB", "polarLAB", "HSV", "HLS", "LUV" and "polarLUV" (from the colorspace package), in addition to "HCL" (which is the same as polarLUV, except that the color components are in the reverse order.). Input color vectors may have an alpha component, in which case, the mapping preserves it, such that the resulting sRGB colors will have the same alpha values.

There are two constructors, one for equally spaced knots and one for arbitrary knots.

Note that in theory, the interpolation is smooth, however, a smooth appearance (or a not so smooth appearance) is dependent on the choice of knots and colors. Similar consecutive colors tend to produce smoother looking results. In general, interpolating over hue, with constant chroma and luminance produces the smoothest results, however, there are many situations where it's desirable for one area (within a plot) to appear brighter than others.

Note that if a litmus object is evaluated with x values outside the knots, then the function will return the first or last color.

Value

Both functions return litmus objects.

References

Refer to the vignette for an overview, references and better examples.

See Also

[Main Plotting Functions](#)

[Functional Versions](#)

Which take a function as their main argument.

[Global Options](#)

[mlitmus, litmus.fit](#)

[Predefined Litmus Objects, Predefined Litmus-Fitting Functions](#)

With predefined colors.

Examples

```
rainbow.litmus.3 <- function (a=0, b=1)
{   colvs <- cbind (c (110, 170, 230, 290), 42.5, 75)
    litmus (a, b, colvs, color.space="HCL")
}

colf <- rainbow.litmus.3 (-1, 1)

plot (colf)

colf (seq (-1, 1,, 10) )
```

32_multi-litmus_objects

Multi-Litmus Objects

Description

Color functions that combine multiple litmus objects.

Usage

```
mlitmus (... , default.color="#D0D0D0", na.color=default.color)
```

Arguments

...	One or more litmus objects.
default.color	String, color that is returned for values outside the litmus objects' knots.
na.color	String, color that is returned for NA values.

Details

This function creates a color function similar to a litmus object, containing one or more litmus objects.

(Noting that these functions return mlitmus objects, so you call these functions, and then if necessary you can evaluate the resulting function).

The color function works out which litmus object to use for each input value.

An example is using one set of colors for positive values and another set of colors for negative values.

Note that it's possible for litmus objects to overlap.

This may be changed, so this feature should not be used inside packages.

Value

An mlitmus object.

References

Refer to the vignette for an overview, references and better examples.

See Also

[litmus](#), [litmus.fit](#)

[Predefined Litmus Objects, Predefined Litmus-Fitting Functions](#)

With predefined colors.

[hot.and.cold](#)

Examples

```
colf <- mlitmus (blue.litmus (0, -1), green.litmus (0, 1) )

plot (colf)
colf (c (-1, 0, 1) )
```

33_litmus-fitting_functions

Fit Litmus Objects to Data

Description

Functions to fit litmus objects to vectors of data.

Usage

```
litmus.fit (x, colvs, ...,
           color.space="sRGB", reverse=FALSE, equalize=0.85, na.color="#FFFFFF")
```

Arguments

x	A numeric vector.
colvs	A 3-column or 4-column numeric matrix, where each row is one color vector, and the optional fourth column is alpha values.
color.space	A string giving the color space, refer to the details section for litmus objects.
reverse	Logical, reverse the order of the colors.
equalize	Numeric, between zero and one, refer to details.
na.color	A single string representing an R color.
...	Ignored.

Details

Refer to the litmus function for background information.

The litmus.fit function constructs a litmus object.

Given n colors, it computes a length-n vector of knots computed from a vector of data.

If equalize is zero, the knots are equally spaced from the lowest x value to the highest. If equalize is one, then knots are selected, such that there's an approximately equal number of points between each pair of knots. And equalization values between zero and one result in an intermediate effect.

Note that high equalize values (higher than the default) may cause color interpolation to appear less smooth.

In general, it's easiest to wrap the litmus.fit function inside another function.

Value

All functions return litmus objects, except the hot.and.cold function which returns a mlitmus object.

References

Refer to the vignette for an overview, references and better examples.

See Also

[Main Plotting Functions](#)

[Functional Versions](#)

Which take a function as their main argument.

[Global Options](#)

[litmus, mlitmus](#)

[Predefined Litmus Objects, Predefined Litmus-Fitting Functions](#)

With predefined colors.

Examples

```
rainbow.litmus.fit.3 <- function (x, ..., reverse=FALSE)
{   colvs <- cbind (c (110, 170, 230, 290), 42.5, 75)
    litmus.fit (x, colvs, color.space="HCL", reverse=reverse)
}

x <- rnorm (30)
colf <- rainbow.litmus.fit.3 (x)

plot (colf)

colf (seq (-1, 1,, 10) )
```

41_supporting_methods *Supporting Methods*

Description

Print and plot methods for litmus and mlitmus objects.

Usage

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

## S3 method for class 'litmus'
plot(x, n=200, ...)
## S3 method for class 'mlitmus'
plot(x, n=200, ...)
```

Arguments

x	A litmus or mlitmus object.
n	Integer, number of strips.
...	Ignored.

Details

The plot method calls `colorspace::swatchplot`, with a vector of colors.

References

Refer to the vignette for an overview, references and better examples.

See Also

[litmus](#), [mlitmus](#)

Examples

```
colf <- blue.litmus ()  
  
print (colf)  
plot (colf)
```

51_simple_color_palettes

Simple Color Palettes

Description

Functions to create a single color, or vector of colors

Usage

```
gold.top.color ()  
blue.top.color ()  
green.top.color ()  
purple.top.color ()  
heat.top.color ()  
  
gold.side.color ()  
blue.side.color ()  
green.side.color ()  
purple.side.color ()  
heat.side.color ()  
  
gold.iso.colors ()  
blue.iso.colors ()  
green.iso.colors ()  
purple.iso.colors ()  
heat.iso.colors ()
```

Value

The top and side functions, return a single (R color) string.
The iso functions return a character (R color) vector.

References

Refer to the vignette for an overview, references and better examples.

Examples

```
gold.top.color ()
gold.side.color ()

gold.iso.colors ()
```

52_predefined_litmus_objects

Predefined Litmus Objects

Description

Functions that wrap the main litmus constructor, with predefined colors.

Usage

```
blue.litmus (a=0, b=1)
green.litmus (a=0, b=1)
heat.litmus (a=0, b=1)
```

```
blue.litmus.hcv (a=0, b=1)
green.litmus.hcv (a=0, b=1)
blue.litmus.flow (a=0, b=1)
green.litmus.flow (a=0, b=1)
```

```
rainbow.litmus (a=0, b=1, ..., c=42.5, l=75, start=65, end=315)
rainbow.litmus.2 (a=0, b=1, ..., c=50, l=70, start=0, end=360)
glass.litmus (a=0, b=1, alpha=0.3, ..., c=42.5, l=62.5, start=42.5, end=260)
```

```
hot.and.cold (t=0, d=1, ..., hot.hue=35, cold.hue=255)
```

Arguments

a, b	Numerics, the lower and upper limits.
c, l, start, end	Same as <code>colorspace::rainbow_hcl</code> .
alpha	A numeric vector giving the alpha component. If it has two or more values, then each alpha value is assigned to each color.
t	Numeric of length one or two, giving the transition points between "hot" and "cold". Two points can be used for a softer effect, but they should be relatively close.
d	Numeric of length one or two, giving the distance about the transition point.
hot.hue, cold.hue	Numerics, the hot and cold hues.
...	Ignored.

Details

Most of these functions wrap the litmus function.

The rainbow.litmus and rainbow.litmus.2 functions are based on colorspace::rainbow_hcl.

Value

All functions return litmus objects.

(Except for hot.and.cold, which returns a mlitmus object).

References

Refer to the vignette for an overview, references and better examples.

See Also

[Main Plotting Functions](#)

[Functional Versions](#)

Which take a function as their main argument.

[Global Options](#)

[litmus, mlitmus, litmus.fit](#)

[Predefined Litmus-Fitting Functions](#)

Examples

```
colf1 <- blue.litmus (-1, 1)
colf2 <- hot.and.cold ()

p0 <- par (mfrow = c (2, 1) )
plot (colf1)
plot (colf2)
par (p0)

u <- seq (-1, 1, , 5)
colf1 (u)
colf2 (u)
```

53_predefined_litmus-fitting_functions
Predefined Litmus-Fitting Functions

Description

Functions that wrap the main litmus-fitting function, with predefined colors.

Usage

```

blue.litmus.fit (x, ..., reverse=FALSE, equalize=0.85)
green.litmus.fit (x, ..., reverse=FALSE, equalize=0.85)
heat.litmus.fit (x, ..., reverse=FALSE, equalize=0.85)

blue.litmus.fit.hcv (x, ..., reverse=FALSE, equalize=0.85)
green.litmus.fit.hcv (x, ..., reverse=FALSE, equalize=0.85)
blue.litmus.fit.flow (x, ..., reverse=FALSE, equalize=0.85)
green.litmus.fit.flow (x, ..., reverse=FALSE, equalize=0.85)

gold.litmus.fit.lum (x, ..., reverse=FALSE, equalize=0.85)
blue.litmus.fit.lum (x, ..., reverse=FALSE, equalize=0.85)
green.litmus.fit.lum (x, ..., reverse=FALSE, equalize=0.85)
purple.litmus.fit.lum (x, ..., reverse=FALSE, equalize=0.85)
heat.litmus.fit.lum (x, ..., reverse=FALSE, equalize=0.85)

rainbow.litmus.fit (x, ..., c=42.5, l=75, start=65, end=315,
  equalize=0.85)
rainbow.litmus.fit.2 (x, ..., c=50, l=70, start=0, end=360,
  equalize=0.85)
glass.rainbow.fit (x, alpha=0.3, ..., c=42.5, l=62.5, start=42.5, end=260,
  equalize=0.85)

hot.and.cold.fit (x, ..., t=0, hot.hue=35, cold.hue=255,
  equalize=0.85)

```

Arguments

x	A numeric vector.
reverse	Logical, reverse the order of the colors.
equalize	Numeric, between zero and one, refer to details, for litmus.fit.
c, l, start, end	Same as colorspace::rainbow_hcl.
alpha	A numeric vector giving the alpha component. If it has two or more values, then each alpha value is assigned to each knot.
t	Numeric of length one or two, giving the transition points between "hot" and cold". Two points can be used for a softer effect, but they should be relatively close.
hot.hue, cold.hue	Numerics, the hot and cold hues.
...	Ignored.

Details

Most of these functions wrap the litmus.fit function.

The rainbow.litmus.fit and rainbow.litmus.fit.2 functions are based on colorspace::rainbow_hcl.

Value

All functions return litmus objects.
(Except for `hot.and.cold.fit`, which returns a `mlitmus` object).

References

Refer to the vignette for an overview, references and better examples.

See Also

[Main Plotting Functions](#)

[Functional Versions](#)

Which take a function as their main argument.

[Global Options](#)

[litmus](#), [mlitmus](#), [litmus.fit](#)

[Predefined Litmus Objects](#)

Examples

```
x <- rnorm (30)

colf1 <- blue.litmus.fit (x)
colf2 <- hot.and.cold.fit (x)

p0 <- par (mfrow = c (2, 1) )
plot (colf1)
plot (colf2)
par (p0)

u <- seq (min (x), max (x),, 5)
colf1 (u)
colf2 (u)
```

54_color_sequence_functions

Color Sequences and Color Fitting Functions

Description

These are convenience functions that combine creating a litmus object and evaluating it, into one step.

Usage

```
blue.seq (n, ..., hcv=FALSE)
green.seq (n, ..., hcv=FALSE)
rainbow.seq (n, ...)
heat.seq (n, ...)

blue.fit (x, ..., hcv=FALSE)
green.fit (x, ..., hcv=FALSE)
rainbow.fit (x, ...)
heat.fit (x, ...)
```

Arguments

n	Integer, the number of points, in the sequence.
x	Numeric vector.
hcv	Logical, if true, use the high color variation option.
...	Ignored.

Details

The `.seq` functions construct a litmus object over the interval one to n, and then evaluate it with one to n.

The `.fit` function fit a litmus object to x, and then evaluate it with x.

Note that `rainbow.seq` is similar to `colorspace::rainbow_hcl`.

Value

A character vector of R colors.

References

Refer to the vignette for an overview, references and better examples.

See Also

[litmus](#), [litmus.fit](#)

[Predefined Litmus Objects](#), [Predefined Litmus-Fitting Functions](#)

Examples

```
blue.seq (10)
blue.fit (rnorm (30) )
```

`61_check_xy_relationship`*Test Axis Relationship*

Description

Estimate the (plotting) relationship between the x and y axes.

Usage

```
test.xyrel (x, y, fv)
```

Arguments

<code>x, y</code>	Optional numeric vectors.
<code>fv</code>	A numeric matrix.

Details

This function is designed to work with the `xyrel` argument in the main plotting functions.

It returns "f" (for a fixed aspect ratio of one) or "m" (for maximized).

If the ratio between the x-size and the y-size is between 0.1 and 10, it will return "f".

Otherwise, it returns "m".

If both `x` and `y` are missing, the "size" refers to the dimensions of the matrix.

(So, returns "f" for up to ten times more columns than rows, or vice versa).

If both `x` and `y` are supplied, the "size" refers to `xlim` and `ylim`, computed from the ranges of `x` and `y`.

If one is supplied but the other is not, then "m" is returned.

Value

Refer to details.

Examples

```
fv1 <- matrix (1:40, 2, 20)
fv2 <- matrix (1:50, 2, 25)

test.xyrel (, fv1)
test.xyrel (seq (0, 1, , 2), seq (0, 20, , 20), fv1)

test.xyrel (, fv2)
test.xyrel (seq (0, 1, , 2), seq (0, 1, , 25), fv1)
```

`62_matrix_plot_margins`*Matrix Margins*

Description

Reverse the top and bottom margins.

Usage

```
matrix.margins ()
```

Details

The function changes the par settings for the margins, for subsequent plots.

It reverses the top and bottom margins, based on the assumption that the user wants the x-axis ticks/labels on the top, and a possible main title on the bottom.

Value

A named list giving the original par settings for the margins.

Examples

```
fv <- matrix (sample (1:24), 4, 6)

p0 <- matrix.margins ()
plot_matrix (,fv, main="my plot")
par (p0)
```

`63_color_conversion` *Color Conversion*

Description

Convert color vectors between different color spaces.

Usage

```
map.color (colvs, string=FALSE, ..., from="sRGB", to="HCL", correction=TRUE)
map.color.3 (x, y, z, alpha, string=FALSE, ..., from="sRGB", to="HCL", correction=TRUE)
```

Arguments

colvs	A length-3 or length-4 numeric vector, or 3-column or 4-column numeric matrix. They give the color components for a particular color space, determined by the from argument.
x, y, z, alpha	Same as colvs, except separate vectors for each color component. If alpha is missing, it's equivalent to a 3-column matrix, and if alpha supplied, it's equivalent to a 4-column matrix.
string	Logical, if true, return (R color) strings, if false, return another colvs matrix.
from, to	Strings giving the name of the input and output color spaces. This includes "XYZ", "RGB", "LAB", "polarLAB", "HSV", "HLS", "LUV" and "polarLUV", in addition to "HCL".
	The to color space is ignored, if string is true.
correction	Logical, if true, apply a correction to sRGB values. Ignored, unless string is false and the to color space is sRGB.
...	Ignored.

Details

These functions are wrappers for functions in the colorspace package.

At the time of writing, there are problems converting HCL (and PolarLUV) to HSV and HLS.

Value

Either a character vector of R color strings, or a 3-column or 4-column matrix of color vectors.

Examples

```
map.color.3 (0, 0, c (0.25, 0.75) )
map.color.3 (0, 0, c (0.25, 0.75),, TRUE)
```

71_deprecated

Deprecated Functions

Description

Deprecated functions, please do not use.

Usage

```
plotf_cfield_3d (... , reverse.z=FALSE)
barface ()
litmus.rainbow.fit (...)
```

Arguments

```
..., reverse.z .
```

Description

Functions to produce sample scalar and vector fields.

Usage

```
#scalar-valued (theoretically)
rotated.sinc (x, y)
bispherical.dist (x, y, z)
```

```
#vector-valued (theoretically)
circular.field (x, y)
plughole.field (x, y, z)
```

Arguments

x, y, z Numeric vectors, where the functions are evaluated.

Details

The rotated.sinc function was adapted from the graphics::persp examples.

The bispherical.dist function gives the smaller of distances from two points at:

(-1, 1, 0)

(1, -1, 0)

The circular.field function generates a vector field with circular flow, and highest magnitude at $r=1$, where r is the distance from the origin.

Value

The (theoretically) scalar-valued functions, return a numeric vector.

The (theoretically) vector-valued functions, return a two-column or three-column matrix.

Examples

```
rotated.sinc (0, 0)
bispherical.dist (0, 0, 0)
circular.field (0, 0)
plughole.field (0, 0, 0)
```

Index

- 11_main_plotting_functions, 2
- 12_functional_versions, 13
- 21_global_options, 16
- 22_standardized_palette_interface, 18
- 31_litmus_objects, 19
- 32_multi_litmus_objects, 21
- 33_litmus_fitting_functions, 22
- 41_supporting_methods, 24
- 51_simple_color_palettes, 25
- 52_predefined_litmus_objects, 26
- 53_predefined_litmus_fitting_functions, 27
- 54_color_sequence_functions, 29
- 61_check_xy_relationship, 31
- 62_matrix_plot_margins, 32
- 63_color_conversion, 32
- 71_deprecated, 33
- 81_sample_functions, 34

- barface (71_deprecated), 33
- bispherical.dist (81_sample_functions), 34
- blue.fit (54_color_sequence_functions), 29
- blue.iso.colors (51_simple_color_palettes), 25
- blue.litmus (52_predefined_litmus_objects), 26
- blue.litmus.fit (53_predefined_litmus_fitting_functions), 27
- blue.seq (54_color_sequence_functions), 29
- blue.side.color (51_simple_color_palettes), 25
- blue.top.color (51_simple_color_palettes), 25
- circular.field (81_sample_functions), 34

- Functional Versions, 2, 12, 21, 23, 27, 29
- Functional Versions (12_functional_versions), 13

- glass.litmus (52_predefined_litmus_objects), 26
- glass.rainbow.fit (53_predefined_litmus_fitting_functions), 27

- Global Options, 12, 15, 21, 23, 27, 29
- Global Options (21_global_options), 16
- gold.iso.colors (51_simple_color_palettes), 25
- gold.litmus.fit.lum (53_predefined_litmus_fitting_functions), 27
- gold.side.color (51_simple_color_palettes), 25
- gold.top.color (51_simple_color_palettes), 25
- green.fit (54_color_sequence_functions), 29
- green.iso.colors (51_simple_color_palettes), 25
- green.litmus (52_predefined_litmus_objects), 26
- green.litmus.fit (53_predefined_litmus_fitting_functions), 27
- green.seq (54_color_sequence_functions), 29
- green.side.color (51_simple_color_palettes), 25
- green.top.color (51_simple_color_palettes), 25

- heat.fit (54_color_sequence_functions),
29
- heat.iso.colors
(51_simple_color_palettes), 25
- heat.litmus
(52_predefined_litmus_objects),
26
- heat.litmus.fit
(53_predefined_litmus-fitting_functions),
27
- heat.seq (54_color_sequence_functions),
29
- heat.side.color
(51_simple_color_palettes), 25
- heat.top.color
(51_simple_color_palettes), 25
- hot.and.cold, 22
- hot.and.cold
(52_predefined_litmus_objects),
26
- hot.and.cold.fit
(53_predefined_litmus-fitting_functions),
27
- litmus, 12, 15, 22, 23, 25, 27, 29, 30
- litmus (31_litmus_objects), 19
- litmus.fit, 12, 15, 21, 22, 27, 29, 30
- litmus.fit
(33_litmus-fitting_functions),
22
- litmus.rainbow.fit (71_deprecated), 33
- Main Plotting Functions, 13, 15, 21, 23,
27, 29
- Main Plotting Functions
(11_main_plotting_functions), 2
- map.color (63_color_conversion), 32
- matrix.margins
(62_matrix_plot_margins), 32
- mlitmus, 21, 23, 25, 27, 29
- mlitmus (32_multi-litmus_objects), 21
- opt.iso.colors
(22_standardized_palette_interface),
18
- opt.litmus.fit
(22_standardized_palette_interface),
18
- opt.nhl (21_global_options), 16
- opt.ref.arrows (21_global_options), 16
- opt.sgrid.color
(22_standardized_palette_interface),
18
- opt.side.color
(22_standardized_palette_interface),
18
- opt.top.color
(22_standardized_palette_interface),
18
- plot.litmus (41_supporting_methods), 24
- plot.mlitmus (41_supporting_methods), 24
- plot_bar (11_main_plotting_functions), 2
- plot_cfield
(11_main_plotting_functions), 2
- plot_cfield3
(11_main_plotting_functions), 2
- plot_dfield
(11_main_plotting_functions), 2
- plot_isosurface
(11_main_plotting_functions), 2
- plot_matrix
(11_main_plotting_functions), 2
- plot_surface
(11_main_plotting_functions), 2
- plot_tricontour
(11_main_plotting_functions), 2
- plot_trisurface
(11_main_plotting_functions), 2
- plot_vec2 (11_main_plotting_functions),
2
- plot_vec3 (11_main_plotting_functions),
2
- plotf_bar (12_functional_versions), 13
- plotf_cfield (12_functional_versions),
13
- plotf_cfield3 (12_functional_versions),
13
- plotf_cfield_3d (71_deprecated), 33
- plotf_dfield (12_functional_versions),
13
- plotf_isosurface
(12_functional_versions), 13
- plotf_surface (12_functional_versions),
13
- plotf_tricontour
(12_functional_versions), 13

- plotf_trisurface
 - (12_functional_versions), 13
- plotf_vec2 (12_functional_versions), 13
- plotf_vec3 (12_functional_versions), 13
- plughole.field (81_sample_functions), 34
- Predefined Litmus Objects, 12, 15, 21–23, 29, 30
- Predefined Litmus Objects
 - (52_predefined_litmus_objects), 26
- Predefined Litmus-Fitting Functions, 12, 15, 21–23, 27, 30
- Predefined Litmus-Fitting Functions
 - (53_predefined_litmus-fitting_functions), 27
- print.litmus (41_supporting_methods), 24
- print.mlitmus (41_supporting_methods), 24
- purple.iso.colors
 - (51_simple_color_palettes), 25
- purple.litmus.fit.lum
 - (53_predefined_litmus-fitting_functions), 27
- purple.side.color
 - (51_simple_color_palettes), 25
- purple.top.color
 - (51_simple_color_palettes), 25
- rainbow.fit
 - (54_color_sequence_functions), 29
- rainbow.litmus
 - (52_predefined_litmus_objects), 26
- rainbow.litmus.fit
 - (53_predefined_litmus-fitting_functions), 27
- rainbow.seq
 - (54_color_sequence_functions), 29
- rotated.sinc (81_sample_functions), 34
- set.bs.nhl (21_global_options), 16
- set.bs.options, 19
- set.bs.options (21_global_options), 16
- set.bs.theme (21_global_options), 16
- st.iso.colors
 - (22_standardized_palette_interface), 18
- st.litmus.fit
 - (22_standardized_palette_interface), 18
- st.sgrid.color
 - (22_standardized_palette_interface), 18
- st.side.color
 - (22_standardized_palette_interface), 18
- st.top.color
 - (22_standardized_palette_interface), 18
- test.xyrel, 7
- test.xyrel (61_check_xy_relationship), 31