barsurf 0.4.0

Heatmap-Related Plots and Smooth Multiband Color Interpolation

Abby Spurdle

February 14, 2020

Supports combined contour-heat plots and 3D bar/surface plots, for plotting scalar fields, either discretely-spaced or continuously-spaced. Also, supports matrix visualization (per se), isosurfaces (for scalar fields over three variables), triangular plots and vector fields. All plots use static vector graphics (suitable for Sweave documents), but high resolution heatmaps can produce smooth raster-like visual effects. Contains a flexible system for smooth multiband color interpolation in RGB, HSV and HCL color spaces.

Introduction

This package contains a variety of plots.

Firstly, there's combined contour-heat plots, 3D bar plots and surface plots.

These can be used to plot scalar fields over two variables, both discretely-spaced and continuously-spaced. Which in turn, can be used to plot functions of two variables.

Also, there's support for matrix visualization (per se), 3D contour plots (or "Isosurfaces"), 3D-based combined contour-heat plots, triangular plots and vector fields.

3D contour plots and 3D-based combined contour-heat plots can be used to plot scalar fields over three variables.

Most plots have a functional version and a matrix version.

The functional versions, take a function with other arguments such as xlim, ylim and n (number of grid points). The matrix versions, take a matrix of values, along with optional x and y coordinates.

Novel "litmus" objects support smooth multiband color interpolation, primarily for heatmaps and surface plots, but may be used for other purposes.

Note that:

- The main references for this package are the help files.
  This vignette is only designed to give an overview, references and provide better examples.

- All plots use vector graphics, but high resolution heatmaps can produce smooth raster-like visual effects.
- Currently, 3D plots use a diamond-based projection, with fixed viewing angles. There's no vertical axis, and by default, 3D plots use reference arrows rather than detailed axes.
- Currently, this package doesn't use color-filled contours.

Preliminary Code (And Required Packages)

I will load (and attach) the barsurf and misc3d packages.

```
> library (barsurf)
> library (misc3d)
```

Note that the barsurf package imports the kubik and colorspace packages.
Also (at the time of writing), the kubik package imports the intoo package, however, this is likely to removed in the near future.

Note that the misc3d package needs to be installed and on the search path, in order to use the functions for 3D contour plots.

```
> set.bs.options (rendering.style="e", theme="blue")
```

I'm setting the theme to blue.
(In principle, this is unnecessary because blue is the default).

The "e" rendering style, uses options designed for viewing PDF documents, electronically.

Functions of Two Variables (Equivalent to Discretely-Spaced Scalar Fields)

The functions **plot_dfield** and **plot_bar** can used to plot matrices representing discretely-spaced scalar fields.

Each function requires at least one argument, fv, which is a matrix of values.

Increasing rows correspond to increasing (discrete) x values and increasing columns correspond to increasing (discrete) y values.

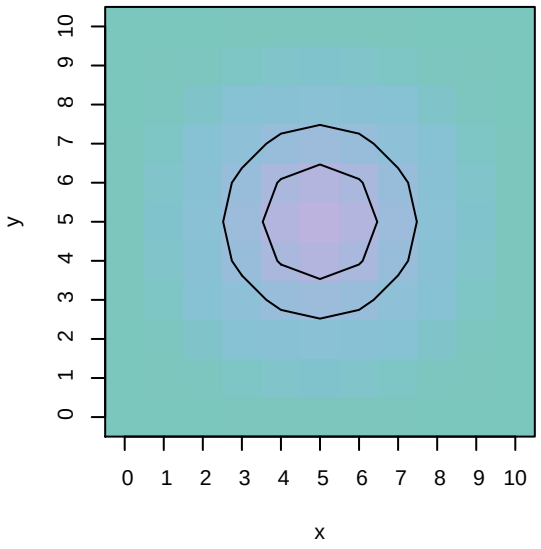
Alternatively, the functions **plotf_dfield** and **plotf_bar** may be used, which take a function, along with xlim and ylim arguments.

Let's construct a simple matrix representing the product of two binomial distributions:

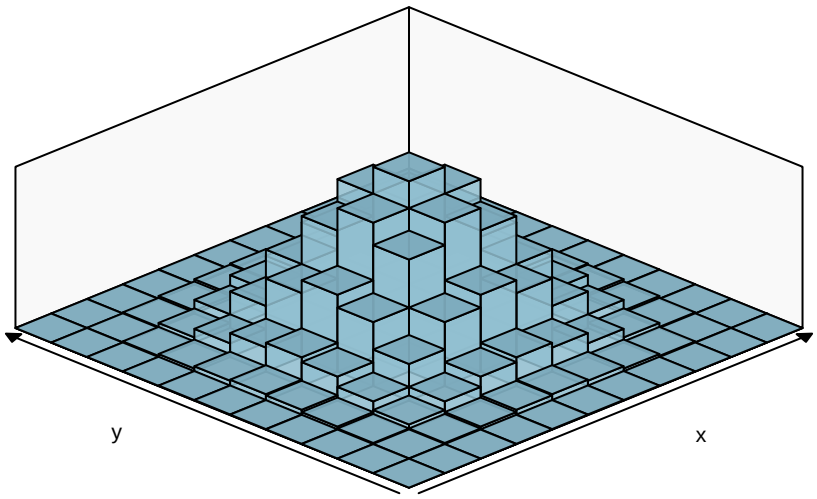
```
> n = 10
> p = 0.5

> x = y = 0:10
> f = function (x, y, n, p)
      dbinom (x, n, p) * dbinom (y, n, p)
> fv = outer (x, y, f, n, p)
```

Then plot in 2D and 3D:

```
> plot_dfield (x, y, fv)
```




```
> plot_bar (,,fv)
```



Note that it's possible to specify the third argument, using two commas, as in the above example.

Also note that if x and y are supplied, they can be specified in one of two ways, using midpoints or breakpoints.

There are many other arguments:

```
> args (plot_dfield)

function (x, y, fv, fb, ..., grid.lines, contours = TRUE, heatmap = TRUE,
    contour.labels = FALSE, main, xlab = "x", ylab = "y", xat,
    yat, xlabs, ylabs, xyrel = test.xyrel(x, y, fv), as.matrix = FALSE,
    add = FALSE, axes = TRUE, reverse = c(FALSE, as.matrix),
    ncontours = 2, clabs, grid.col, contour.col = "#000000",
    color.function, color.fit, cols, hcv = FALSE)
NULL
```

```
> args (plotf_dfield)

function (f, xlim, ylim = xlim, ...)
NULL
```

Refer to the help files for more information.

Functions of Two Variables (Equivalent to Continuously-Spaced Scalar Fields)

The functions **plot_cfield** and **plot_surface** can used to plot matrices representing continuously-spaced scalar fields.

Similar to the discrete case, each function requires at least one argument, fv, which is a matrix of values.

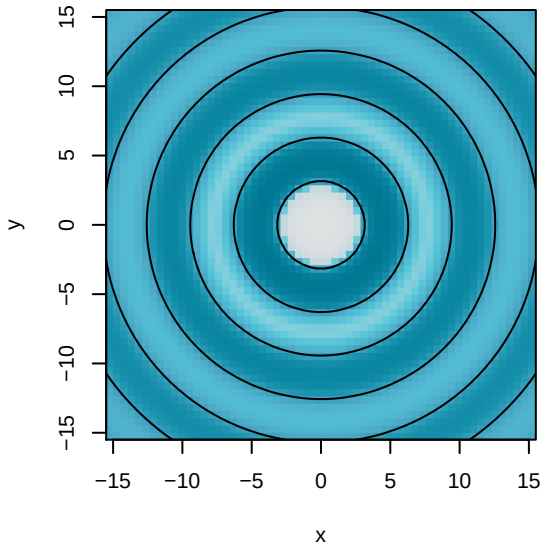
Increasing rows correspond to increasing (continuous) x values and increasing columns correspond to increasing (continuous) y values.

Alternatively, the functions **plotf_cfield** and **plotf_surface** may be used, which take a function, along with xlim, ylim and n arguments.

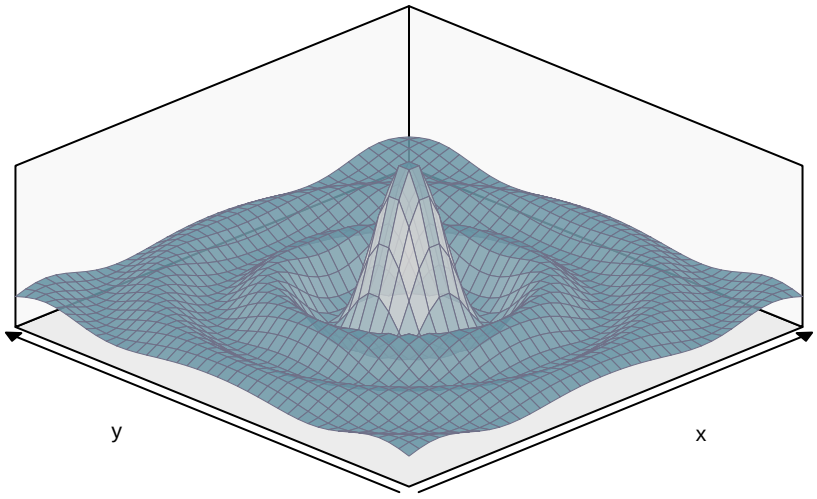
The n argument is a vector of length one or two, giving the number of grid points in each x and y direction.

Let's plot the rotated sinc function, adapted from the graphics::**persp** examples:

```
> plotf_cfield (rotated.sinc, c (-15.5, 15.5), fb=0, n=60, hcv=TRUE)
```




```
> plotf_surface (rotated.sinc, c (-15.5, 15.5), n=40)
```



Note that in the contour-heat plot, the contour values have been set to zero, and the high color variation option has been used.

Also note that the heatmap is not smooth.

We could use a larger n value, or (more efficiently) superimpose a smaller heatmap on top of a larger heatmap.

There's an example in an appendix.

Matrix Visualization (Per Se)

By matrix visualization per se, I'm referring to situations where matrices are of interest, in themselves.

In contrast, all the main plotting functions in this package use matrices, but mainly as intermediate objects, to get from a (mathematical) function to a plot.

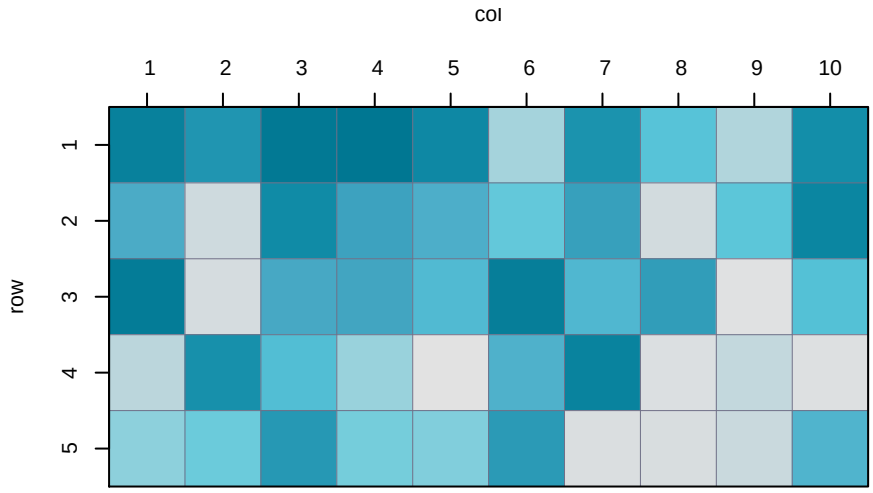
The **plot_matrix** function is a wrapper for **plot_dfield**.
It transposes the fv matrix, and it has different default argument values.

By default, the plotted fv matrix has the same orientation as a matrix in text form.

Here's a simple example:

```
> M = matrix (sample (1:50), 5, 10)
```

```
> plot_matrix (,,M)
```



Functions of Three Variables (Equivalent to Scalar Fields over Three Variables)

This package contains two sets of functions for plotting scalar-fields over three variables.

In principle, they're designed for continuously-spaced scalar fields, but can be used for discretely-spaced scalar fields with care.

Contour Plots, in 3D

Re-iterating, these functions require the misc3d package to be installed and on the search path.

The **plot_contour_3d**, **plotf_contour_3d** and **nested_isosurfaces** functions can be used to plot 3D contour plots.

Unlike previous plotting functions, the fv argument is a three dimensional array rather than a matrix.
(And there's a z coordinate, separate from the function value).

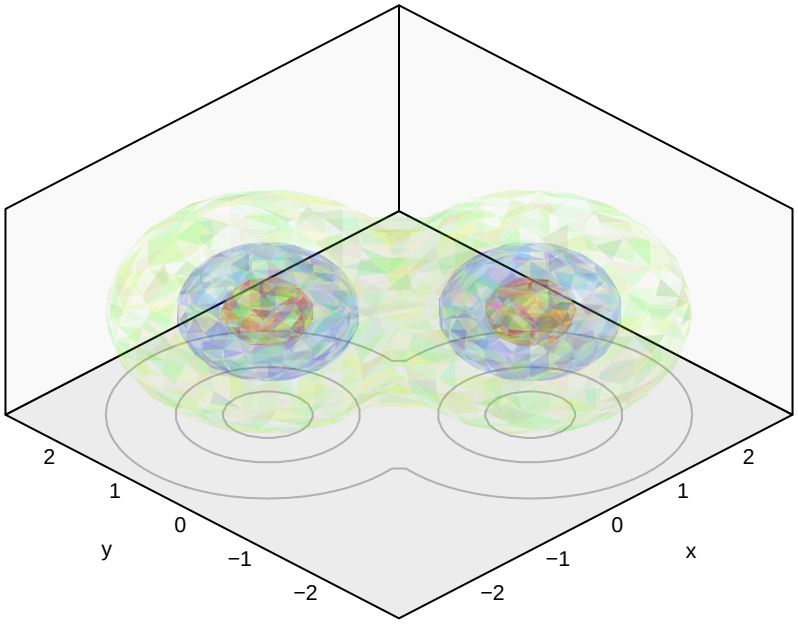
The first two variables have the same interpretation as other 3D plots in this package, with the third variable giving the height. Except that x, y and z describe coordinates of the fv array, not the resulting isosurfaces.

Also, its possible for the the x, y, z, fv and n arguments to be lists, one list element for each isosurface.

The **bispherical.dist** function gives the smallest of two distances from two points, in 3D space.

The following example plots the **bispherical.dist** function, giving isosurfaces where the function equals 0.5, 1 or 1.75.

```
> nested_isosurfaces (bispherical.dist,
      c (-3, 3),, c (-2, 2), c (0.5, 1, 1.75),
      arrows=FALSE)
```



Note that my functions use misc3d::**computeContour3d** function to compute isosurfaces, which in turn, uses the "Marching Cubes" algorithm.

Combined Contour-Heat Plots, 3D-Based

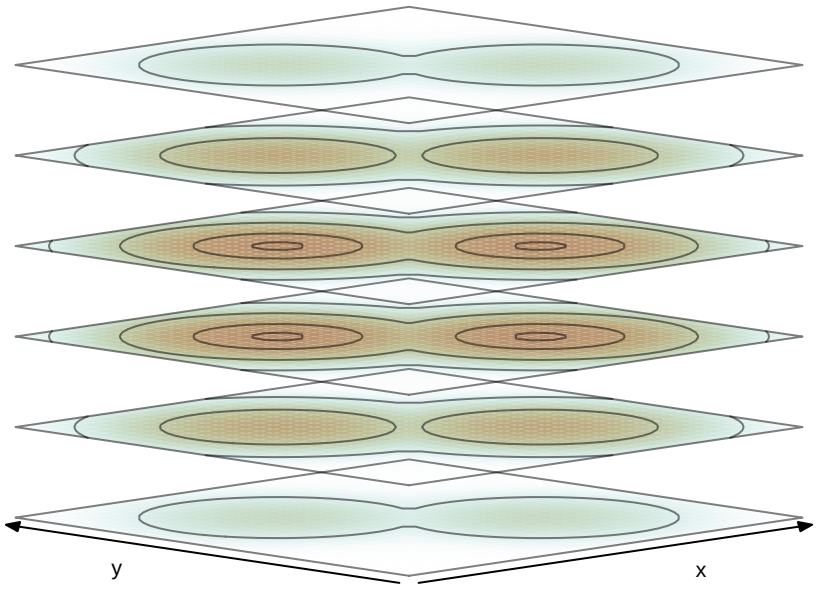
Here, 3D-based combined contour-heat plots contain a set of 2D slides (or slices).

The **plot_cfield_3d** and **plotf_cfield_3d** functions are similar to the **plot_cfield** and **plotf_cfield** functions.

Unlike previous plotting functions, the fv argument is a list of two or more matrices. One matrix for each slide.

The following example also plots the **bispherical.dist** function, but gives slides rather than isosurfaces:

```
> plotf_cfield_3d (bispherical.dist, c (-3, 3),, c (-2, 2),
      fb = c (0.5, 1, 1.75, 2.5), emph="l")
```



Note that there's another example, later.

Triangular Plots

The functions **plot_tricontour**, **plot_trisurface**, **plotf_tricontour** and **plotf_trisurface** can be used to produce triangular plots.

They're similar to the **plot_cfield**, **plot_surface**, **plotf_cfield** and **plotf_surface** functions.

In the matrix version, the matrix must be square, and only upper left part of the matrix (including the diagonal) is used.

In functional versions, the function needs to be a function of three variables (w1, w2, w3) or (x, y and z) etc, which takes values between 0 and 1, and in general, sum to one.
If you need to plot a function that doesn't have these properties, then you need to create a wrapper function.

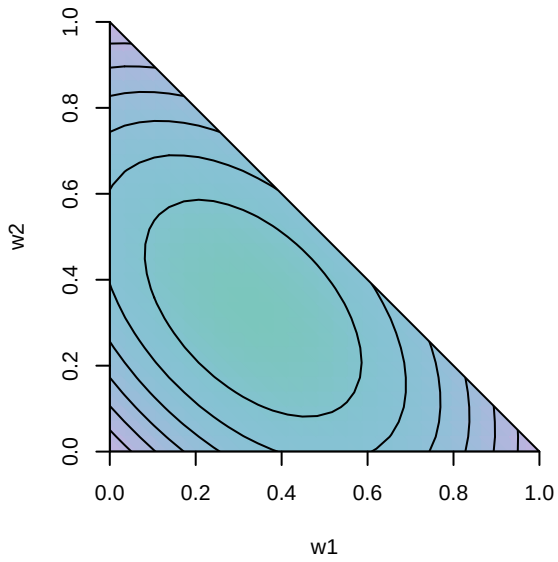
Here's a simple example:

```
> f = function (w1, w2, w3)
      (w1 - 1 / 3)^2 + (w2 - 1 / 3)^2 + (w3 - 1 / 3)^2
```

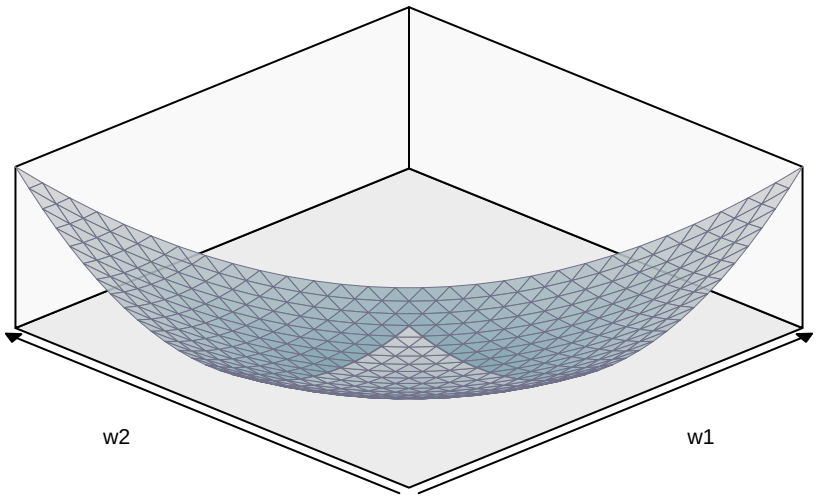
Note that this function doesn't require it arguments (w1, w2 and w3) to sum to one.

And plots:

```
> plotf_tricontour (f, xlab="w1", ylab="w2")
```




```
> plotf_trisurface (f, xlab="w1", ylab="w2")
```



Note that the x and y arguments are ignored.

Vector Fields

The **plot_vecfield** and **plotf_vecfield** functions can be used to produce plots of vector fields.

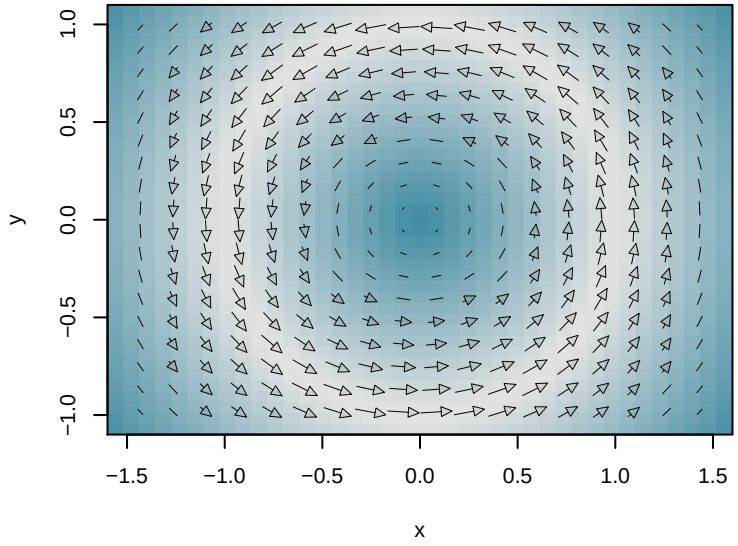
They're similar to the **plot_cfield** and **plotf_cfield** functions.

In the matrix version, there's two input matrices, fx (for the x component) and fy (for the y component).

In the functional version, the function needs to return a length-two list, with the first element being the x component and the second element being the y component.

Here's a simple example:
(The **concentric.field** function generates a vector field with circular flow).

```
> plotf_vecfield (concentric.field, c (-1.6, 1.6), c (-1.1, 1.1),
      m = c (80, 60) )
```



Color Themes

Default colors are determined by global options.

A color theme can be set by the **set.bs.theme** function.

```
> set.bs.theme ("heat")
> set.bs.theme ("blue")
```

Supported themes include heat, gold, blue, green and purple.

The heat theme is designed for higher impact than the other themes.

Also, note that the heat, gold and purple themes use some colors from the blue theme.

Color Customization (Overview)

Colors are determined by color functions, which map logical or numeric vectors to character vectors representing R colors.

For bar plots, the color function maps a logical vector to colors, and for other plots, the color function maps a numeric vector to colors. This package uses a system of "barface" objects and "litmus" objects to simplify this process, however, you can use any functions that meet these requirements.

```
> colf = barface.heat ()
> colf (c (TRUE, FALSE) )

[1] "#CE7A6BF2" "#DC9186D9"

> colf = litmus.heat ()
> colf (c (0.25, 0.75) )

[1] "#FF6600" "#FFE646"
```

In **plot_bar**, the color function (if not supplied) is determined by global options, and for other plots the color function (if not supplied) is determined by a color fit function. Similarly, the color fit function (if not supplied) is determined by global options.

Note that the **plot_dfield** and **plot_bar** functions also accept a color matrix.
Furthermore, **plot_bar** also accepts a list of two color matrices, one for tops and one for sides.

Barface and litmus objects are discussed in more detail in the following sections.

Barface Objects

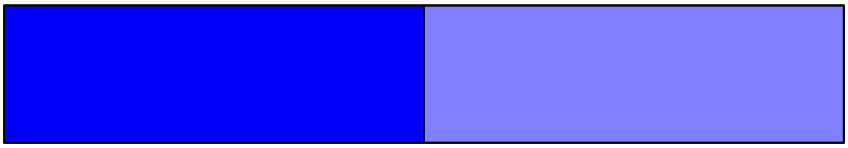
The **barface** function can be used to create new barface objects.

It takes one or two colors (for the bar tops and bar sides) as length-3 or length-4 vectors.

By default, colors assumed to be in sRGB color space, however, other color spaces can be used, including "HCL".

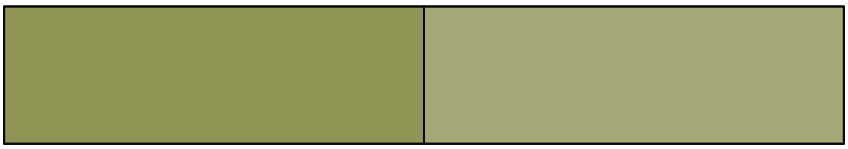
A simple sRGB example:

```
> colf = barface (c (0, 0, 1) )
> plot (colf)
```



And a HCL-based example, specifying both tops and sides:

```
> colf = barface (c (90, 42.5, 60), c (90, 35, 67.5), "HCL")
> plot (colf)
```



Alternatively, this package has five predefined barface objects, for the heat, gold, blue, green and purple themes:

```
> plot (barface.blue () )
```

```
> plot (barface.green () )
```

Litmus Objects

The **litmus** and **litmus.spline** functions can be used to create litmus objects.

The **litmus** function takes a 3-column or 4-column matrix, representing length-3 or length-4 color vectors.
Also, it takes lower and upper limits, which default to zero and one.

The **litmus.spline** function is the same, except that it takes a vector of two or more knots with the same length as the number of colors (number of rows).

A matrix of color vectors:

```
> colvs = cbind (c (c (100, 150, 200, 250) ), 35, 75)
> rownames (colvs) = c ("first color", "(2nd)", "(3rd)", "last color")
> colnames (colvs) = c ("H", "C", "L")

> colvs

              H  C  L
first color 100 35 75
(2nd)       150 35 75
(3rd)       200 35 75
last color  250 35 75
```

Note that its not necessary to set row and column names.
(I've just done that to make the matrix easier to interpret).

A litmus object, with the default input range:

```
> colf = litmus (,,colvs, "HCL")
> colf

litmus object, x in [0, 1]

> plot (colf)
```

Alternatively, this package has several predefined litmus objects, for the heat, gold, blue, green and purple themes, in addition to rainbow litmus objects adapted from the colorspace package.

```
> plot (litmus.heat () )
```

```
> plot (litmus.blue () )
```

```
> plot (litmus.green () )
```

```
> plot (litmus.rainbow () )
```

Note that in general, it's easier to create litmus object using the **litmus.fit** function, or one of its wrappers, discussed in later sections.

Multi-Litmus Objects

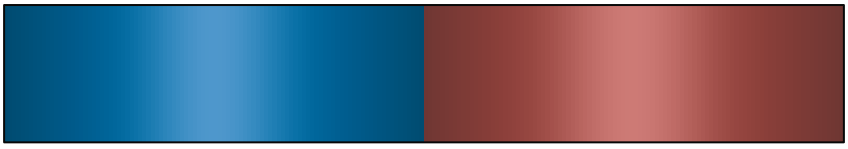
Multi-litmus objects are litmus-like objects containing other litmus objects.

The **mlitmus** function constructs a "mlitmus" (multi-litmus) object from one or more litmus objects:

```
> colf = mlitmus (litmus.green (-1, 0, reverse=TRUE), litmus.blue (0, 1) )
> plot (colf)
```

This package contains one predefined multi-litmus object, for a hot and cold effect:

```
> plot (hot.and.cold () )
```



Note that the above example is questionable, and I recommend caution mixing highly contrasting colors.

Lets create a less trivial example:

```
> colf1 = glass.rainbow (0, 1.25, 1, start=-200, end=200)
```

```
> colf2 = glass.rainbow (1.25, 1.75, c (1, 0.7, 0.4, 0.1), start=200, end=220)
> colf3 = glass.rainbow (1.75, 5, 0.1, start=220)
```

Note that the third argument is the alpha value.
Other arguments are ranges.

```
> colf = mlitmus (colf1, colf2, colf3)
```

Note that if using multi-litmus objects in heatmaps, sudden "jumps" may cause artifacts in the plot.

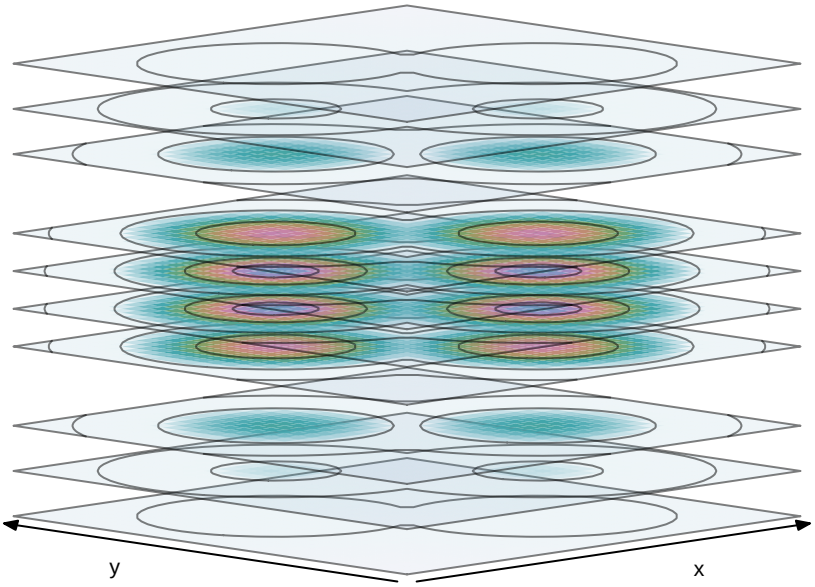
In general, such multi-litmus objects, should contain at least three litmus objects, with the central litmus object interpolating between the end of the first litmus and the start of the third litmus.

```
> plot (colf)
```

Also note that this multi-litmus object is semi-transparent, and it's possible that (in PDF viewers) lines may appear in it.

Now, I will to recreate the earlier 3D-based contour-heat plot, using different z coordinates, and using the multi-litmus object:

```
> z = c (-2, -1.6, -1.2, -0.5, -0.1667, 0.1667, 0.5, 1.2, 1.6, 2)

> plotf_cfield_3d (bispherical.dist, c (-3, 3), z=z,
      fb = c (0.5, 1, 1.75, 2.5), color.function=colf)
```



Fitting Litmus Objects to Data

In some cases, one may create a litmus object, directly.

However, if creating color functions for heatmaps, then the color function needs to match the function/matrix values.

We can create a function to fit litmus objects by wrapping the **litmus.fit** function:

```
> custom.litmus.fit = function (x)
  {   colvs = cbind (c (c (100, 150, 200, 250) ), 35, 75)
      litmus.fit (x, colvs, "HCL")
  }
```

Let's say we have a vector, x:

```
> x = runif (10)
```

We can fit a litmus object using our function:

```
> range (x)

[1] 0.06434287 0.97369252

> custom.litmus.fit (x)

litmus object, x in [0.06434287, 0.9736925]
```

This package contains wrappers for the **litmus.fit** function, with predefined colors:

```
> litmus.blue.fit (x)

litmus object, x in [0.06434287, 0.9736925]
```

The **litmus.fit** function and most of its wrappers have an "equalize" parameter.

Let say we have function values from a function with high curvature:

```
> x = y = seq (0, 1,, 30)
> fv = outer (x, y)
> fv = fv^3
```

We can create litmus objects with different equalize values:

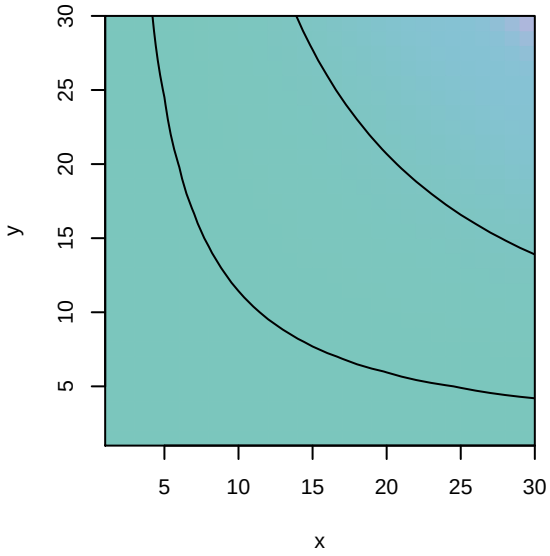
```
> colf.simple = litmus.blue.fit (fv, equalize=0)
> colf.equalized = litmus.blue.fit (fv, equalize=1)
```

With no equalization the resulting litmus object appears relatively even:

```
> plot (colf.simple)
```

However, in the resulting heatmap, the opposite happens:
(The plot is mostly green).

```
> fb = c (0.0014, 0.0881)

> plot_cfield (,,fv, fb=fb, color.function=colf.simple)
```

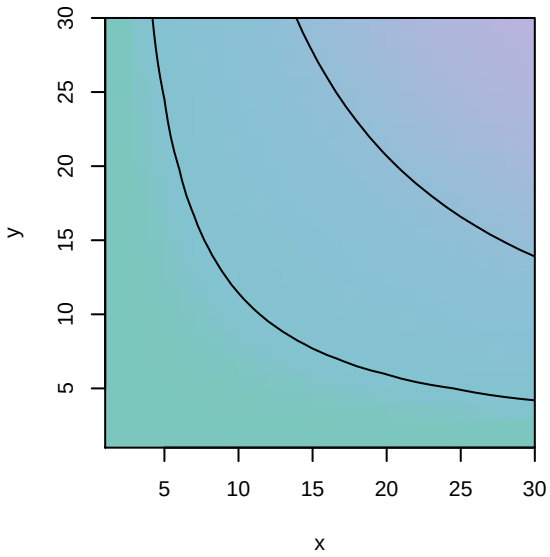


In contrast, the completely equalized litmus object, doesn't appear even:

```
> plot (colf.equalized)
```

But the resulting heatmap is:

```
> plot_cfield (,,fv, fb=fb, color.function=colf.equalized)
```



Note that by default, the equalize parameter is 0.85.

This results in a softer effect, than the completely equalized example above.

References

R Packages

Spurdle, A. (2019). kubik: Cubic Hermite Splines

Ihaka, R., Murrell, P., Hornik, K., Fisher, J. Stauffer, R., Wilke, C., McWhite, C., & Zeileis, A. (2019). colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes

Feng, D., & Tierney, L. (2013) misc3d: Miscellaneous 3D Plots

Notes

The colorspace package contains further color-related references.

Appendix:
Improved Contour-Heat Plot

```
> x = y = seq (-15.5, 15.5, length.out=60)
> fv = outer (x, y, rotated.sinc)
> colf = litmus.blue.fit.hcv (fv)

> #larger heatmap
> plot_cfield (x, y, fv,
      contours=FALSE, color.function=colf)
> #smaller heatmap
> plotf_cfield (rotated.sinc, c (-4, 4),
      add=TRUE, contours=FALSE, color.function=colf, n=60)
> #contour lines
> plot_cfield (x, y, fv,
      add=TRUE, fb=0, heatmap=FALSE)
```