barsurf 0.3.0

Bar, Surface and Related Plots

Abby Spurdle

May 16, 2019

Produces heat maps, contour plots, bar plots (in 3D) and surface plots (also, in 3D). Is designed for plotting functions of two variables, however, can plot relatively arbitrary matrices. Uses HCL color space, extensively. Also, supports triangular plots and nested matrices.

Introduction

This is a relatively minimal R package for mathematical and statistical graphics.

Currently, it contains six main plotting functions:

- plot2d.cell()
- plot2d.contour()
- plot2d.tricontour()
- plot3d.bar()
- plot3d.surface()
- plot3d.trisurface()

The functions plot2d.cell() and plot3d.bar() are designed for functions over a discrete rectangular domain, however, can also be used for relatively small but otherwise arbitrary (integer and numeric) matrices, and bivariate categorical data.

The functions plot2d.contour() and plot3d.surface() are designed for functions over a continuous rectangular domain. And the functions plot2d.tricontour() and plot3d.trisurface() are designed functions over a continuous triangular domain.

The function plot2d.cell() produces a heat map. By default, the functions plot2d.contour() and plot2d.tricontour produce contour plots with heat maps. The 3D plots use orthographic projection (rather than perspective projection) and currently, use a fixed viewing angle (so, you can't rotate the plots).

All the plots use colors defined in HCL color space.

All of these functions require at least one argument, z. In general, z is a standard matrix object, however, z can also be a nested matrix object. Either way, increasing row indices represent increasing x values, and increasing column indices represent increasing y values.

Note that all the main plotting functions except plot3d.bar() have a contrast parameter, that allows us to decrease the contrast, if required.

Loading The Packages

First, we need to load (and attach) the intoo and barsurf packages.

```
> library (intoo)
> library (barsurf)
```

Note that the barsurf package imports the colorspace package.

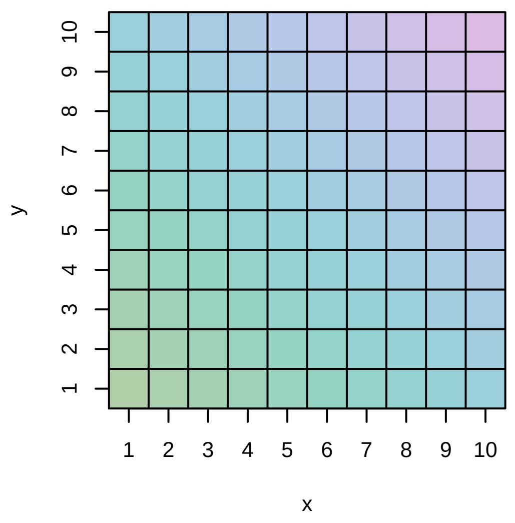
Heat Maps and Bar Plots

Let's construct a simple matrix (representing cells, or discrete areas).

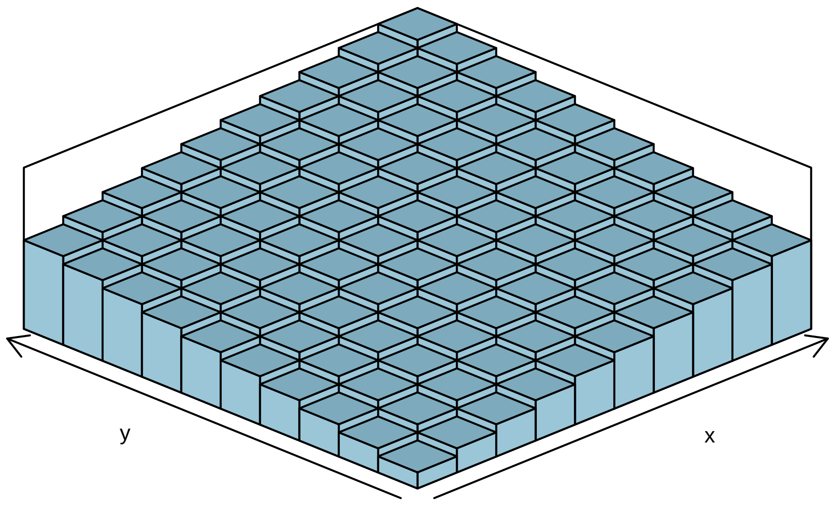
```
> z1 = outer (1:10, 1:10, "+")
```

And then produce a heat map (in 2D) and a bar plot (in 3D).

```
> plot2d.cell (,,z1)
```



```
> plot3d.bar (,,z1, zlim=c (0, 20) )
```



Note the two commas.

Also note that the origin of the 3D plot is at the bottom center.

The cells (or bars) don't have to be the same size. We can specify x and y break points, in which case their lengths should be one more than the dimensions of z, and they should be sorted in ascending order.

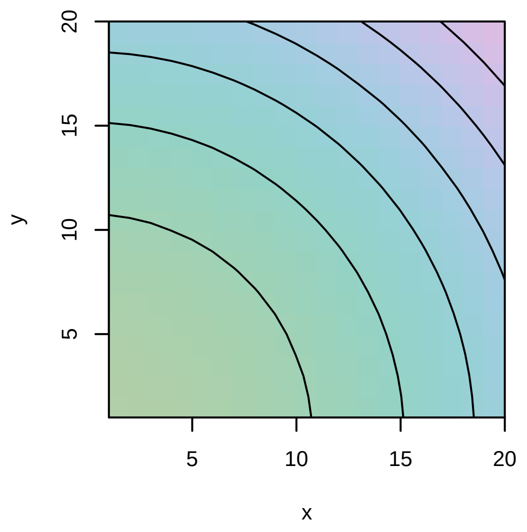
Contour Plots and Surface Plots

Let's construct another matrix (but representing points on a surface rather than cells).

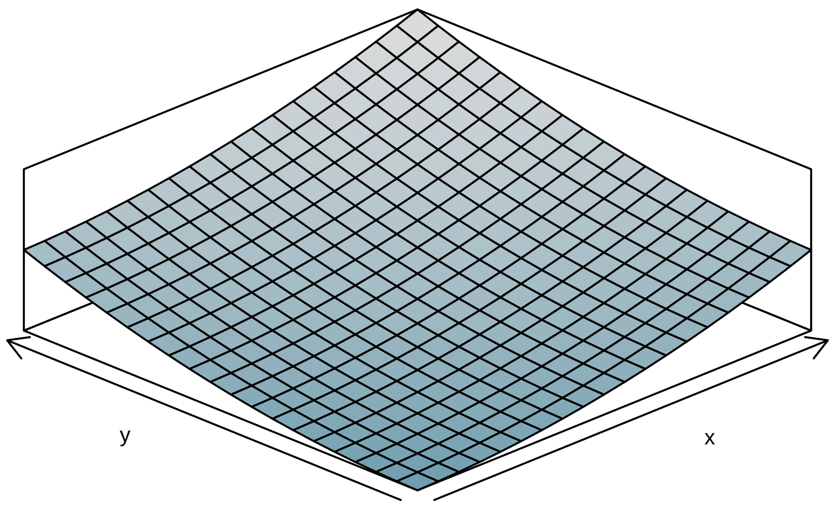
```
> x = y = 1:20
> f = function (x, y) x ^ 2 + y ^ 2
> z2 = outer (x, y, f)
```

And then we can produce a contour plot (in 2D) and a surface plot (in 3D).

```
> plot2d.contour (,,z2)
```



```
> plot3d.surface (,,z2)
```



It doesn't have to be a regularly spaced grid. We can specify x and y values, in which case their lengths should equal to the dimensions of z.

Color Conversion

I've supplied functions for mapping RGB to HCL, and vice versa.

These are just wrappers for functions in the colorspace package, however, they map a vector to a vector, rather than a color object to a color object.

```
> colv.1.rgb = c (1, 0, 0)
> colv.1.hcl = rgb2hcl (colv.1.rgb)
> round (colv.1.hcl, 2)
```

```
[1]  12.17 179.04  53.24
```

```
> colv.2.hcl = c (0, 50, 60)
> colv.2.rgb = hcl2rgb (colv.2.hcl)
> round (colv.2.rgb, 2)
```

```
        R    G    B
[1,] 0.78 0.48 0.54
```

By default, the hcl2rgb() function, corrects the RGB values, if they're not in the interval [0, 1].

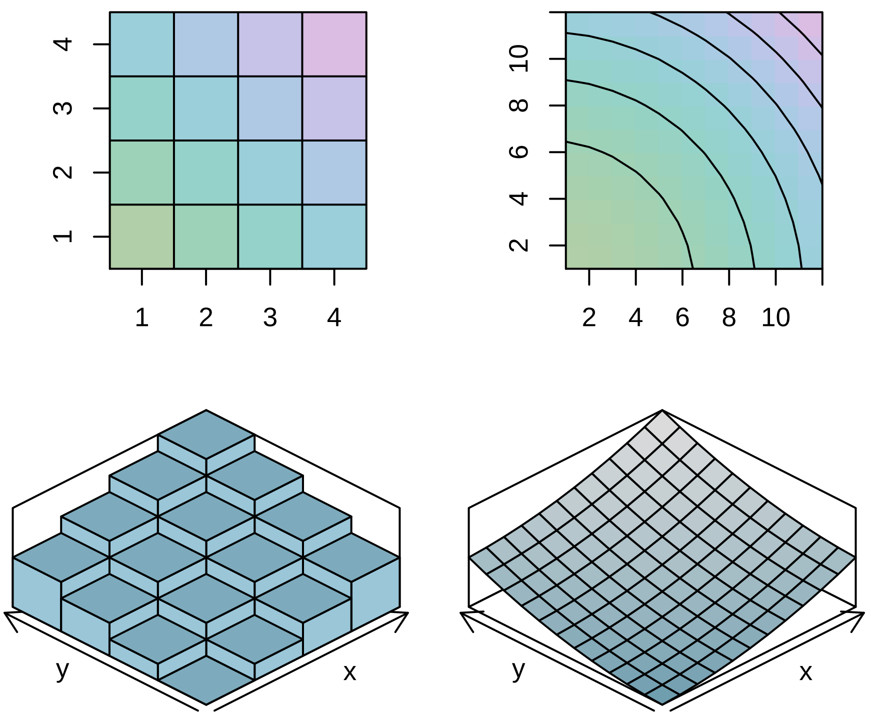
Changing Colors

Currently, there are three ways to change the colors:

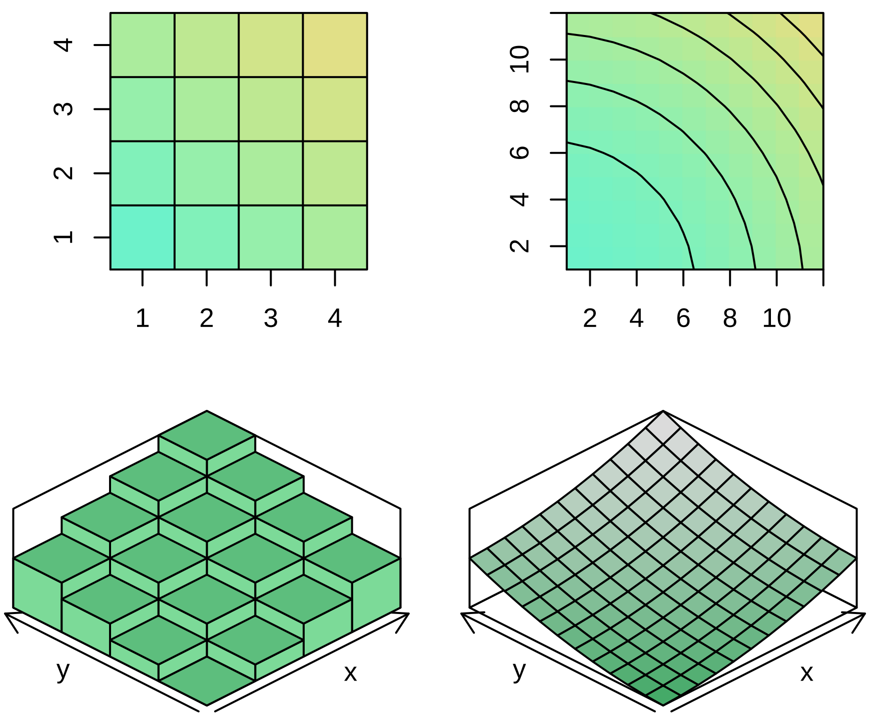
1. Use the use.theme() function.
2. Set global options, directly.
3. Specify color arguments (mainly, colv.1 and colv.2), in each function call.

We can set the theme by either:

```
> use.theme ("blue")
> test.theme ()
```



```
> use.theme ("green")
> test.theme ()
```



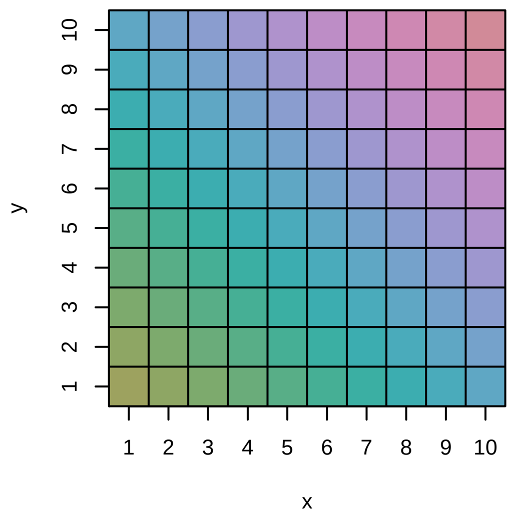
Global options are stored in a list named barsurf.

```
> bso = getOption ("barsurf")
> bso

$plot2d.cell.colv.1
[1] 120  30  80

$plot2d.cell.colv.2
[1] 300  30  80

$plot2d.cell.colv.na
[1]   0   0 90

$plot2d.contour.colv.1
[1] 120  30  80

$plot2d.contour.colv.2
[1] 300  30  80

$plot3d.bar.colv.1
[1] 220.0  30.0  67.5

$plot3d.bar.colv.2
[1] 220.0  30.0  77.5

$plot3d.surface.colv.1
[1] 220.0  30.0  62.5

$plot3d.surface.colv.2
[1]   0.0   0.0 87.5
```

```
> bso$plot2d.cell.colv.1 = c (90, 45, 65)
> bso$plot2d.cell.colv.2 = c (360, 45, 65)
```

```
> options (barsurf=bso)
```

```
> plot2d.cell (,,z1)
```



Note that in this section, I've omitted some R input which resets the theme to blue.

Triangular Plots

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

Let's create another matrix.

```
> x = y = -9:10
> f = function (x, y) 200 - (x ^ 2 + y ^ 2)
> z3 = outer (x, y, f)
```

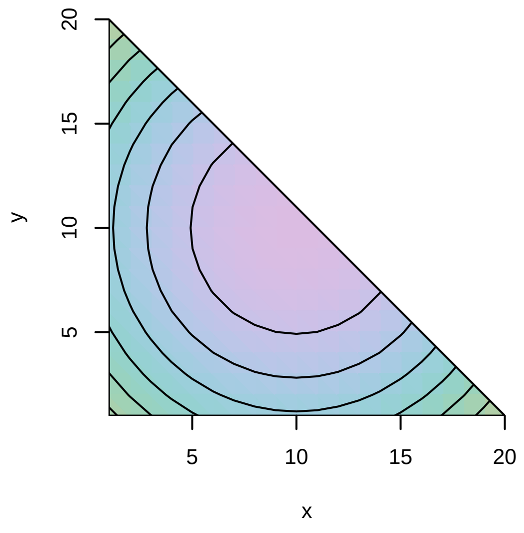
I'm going to set the lower right part of the matrix to NAs to make it obvious what's happening here, however, this step is unnecessary.

```
> z3 = lr2na (z3)
> P.ht (z3)
```

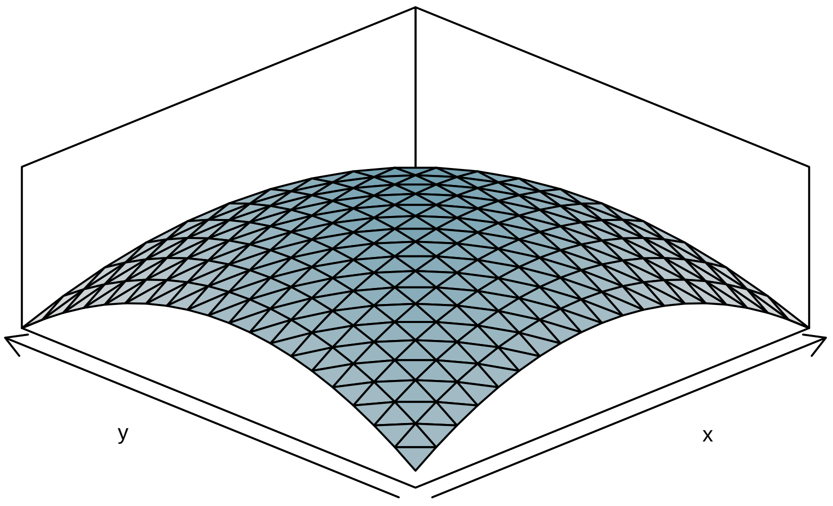
```
       [,1] [,2] [,3] [,4]   [,17] [,18] [,19] [,20]
[1,]    38   55   70   83  |  70    55    38    19
[2,]    55   72   87  100  |  87    72    55
[3,]    70   87  102  115  | 102    87
[4,]    83  100  115  128  | 115
       ---  ---  ---  ---  + ---   ---   ---   ---
[17,]   70   87  102  115  |
[18,]   55   72   87       |
[19,]   38   55            |
[20,]   19                 |
```

And plot them.

```
> plot2d.tricontour (,,z3)
```



```
> plot3d.trisurface (,,z3)
```



Note that in plot3d.trisurface(), the x and y arguments are ignored.

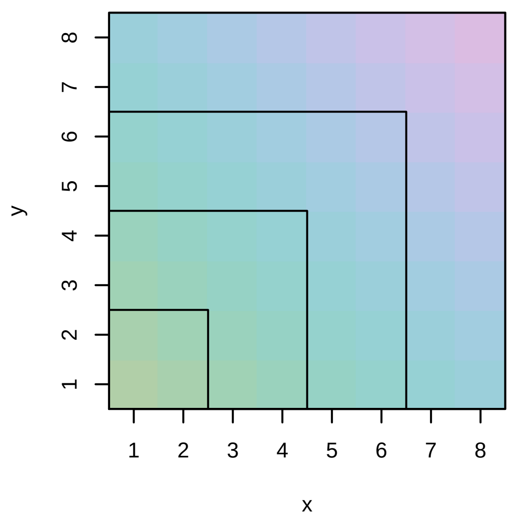
Nested Matrices

My other package, intoo, contains early prototypes for nested matrices, which can be used for grouped heat maps and grouped bar plots.

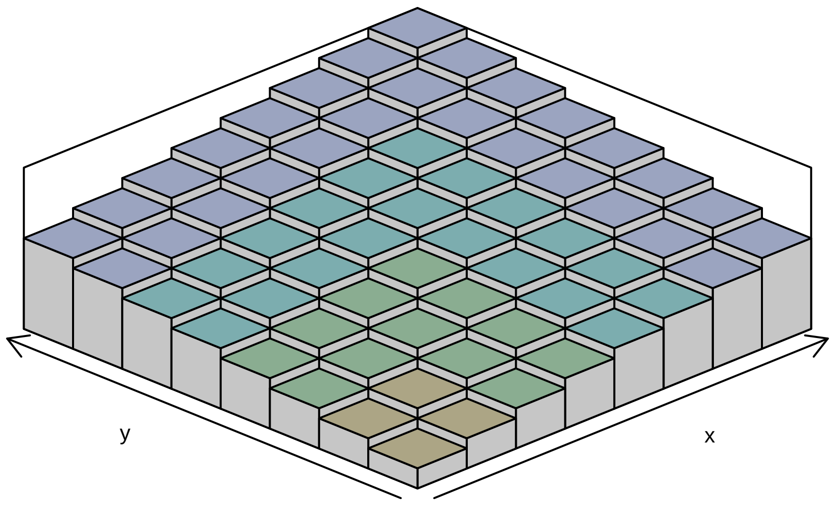
Let's create a nested matrix object.

```
> z4 = outer (1:8, 1:8, "+")
```

```
> sm.1 = P.smatrix (1, 2, 1, 2)
> sm.2 = P.smatrix (1, 4, 1, 4)
> sm.3 = P.smatrix (1, 6, 1, 6)
> sm.4 = P.smatrix (1, 8, 1, 8)

> z4 = P.nmatrix (z4, list (sm.1, sm.2, sm.3, sm.4) )

> z4

     [,1] [,2]    [,3] [,4]    [,5] [,6]   [,7] [,8]
[1,]  2    3   |  4    5   |  6    7   | 8    9
[2,]  3    4   |  5    6   |  7    8   | 9   10
     --   --   +           |           |
[3,]  4    5      6    7   |  8    9   | 10   11
[4,]  5    6      7    8   |  9   10   | 11   12
     --   --   -- --   --  +           |
[5,]  6    7      8    9     10   11   | 12   13
[6,]  7    8      9   10     11   12   | 13   14
     --   --   -- --   --  -- --   --  +
[7,]  8    9     10   11     12   13     14   15
[8,]  9   10     11   12     13   14     15   16

> plot2d.cell (,,z4)
```



```
> plot3d.bar (,,z4, zlim=c (0, 16), colv.2=c (0, 0, 80), reverse=TRUE)
```


y
x