hyperSpec Plotting functions

Claudia Beleites <Claudia.Beleites@chemometrix.gmbh>
DIA Raman Spectroscopy Group, University of Trieste/Italy (2005 – 2008)
Spectroscopy · Imaging, IPHT, Jena/Germany (2008 – 2017)
ÖPV, JKI, Berlin/Germany (2017 – 2019)
Arbeitskreis Lebensmittelmikrobiologie und Biotechnologie, Hamburg University, Hamburg/Germany (2019 – 2021)
Chemometric Consulting and Chemometrix GmbH, Wölfersheim/Germany (since 2016)

September 13, 2021

Contents

1 Predefined functions ............................................ 2
2 Arguments for plot ......................................... 5
3 Spectra ..................................................................... 8
   3.1 Stacked spectra .............................................. 10
4 Calibration Plots, (Depth) Profiles, and Time Series Plots .......... 12
   4.1 Calibration plots ........................................... 12
   4.2 Time series and other Plots of the Type Intensity-over-Something .......... 13
5 Levelplot .................................................................. 14
6 Spectra Matrix .................................................. 14
7 False-Colour Maps: plotmap .................................... 16
8 3D plots (with rgl) ............................................... 20
9 Using ggplot2 with hyperSpec objects ............................. 21
10 Troubleshooting ................................................. 22
    10.1 No output is produced .................................. 22
11 Interactive Graphics

11.1 spc.identify: finding out wavelength, intensity and spectrum .................. 23
11.2 map.identify: finding a spectrum in a map plot .................................. 23
11.3 map.sel.poly: selecting spectra inside a polygon in a map plot ............... 23
11.4 Related functions provided by base graphics and lattice ........................ 23

Suggested Packages

latticeExtra: available

deldir: available

rgl: available

ggplot2: available

In addition tripack, and latticist are mentioned, but not used in this vignette.

Preliminary Calculations

For some plots of the chondro dataset, the pre-processed spectra and their cluster averages ± one standard deviation are more suitable:

> chondro.preproc <- chondro - spc.fit.poly.below (chondro)
> chondro.preproc <- chondro.preproc / rowMeans (chondro)
> chondro.preproc <- chondro.preproc - quantile (chondro.preproc, 0.05)
> cluster.cols <- c ("dark blue", "orange", "#C02020")
> cluster.meansd <- aggregate (chondro.preproc, chondro$clusters, mean_pm_sd)
> cluster.means <- aggregate (chondro.preproc, chondro$clusters, mean)

For details about the pre-processing, please refer to the example work flow in vignette ("chondro"), or the help ? chondro.

1 Predefined functions

hyperSpec comes with 6 major predefined plotting functions.

plot main switchyard for most plotting tasks

levelplot hyperSpec has a method for lattice [?] function levelplot

plotspc plots spectra

plotmat plots the spectra matrix

plotc calibration plot, time series, depth profile
  plotc is a lattice function

plotmap more specialized version of levelplot for map or image plots.
  plotmap is a lattice function

plotvoronoi more specialized version of plotmap that produces Voronoi tessellations.
  plotvoronoi is a lattice function
plotmap, plotvoronoi, and levelplot are lattice functions. Therefore, in loops, functions, Sweave chunks, etc. the lattice object needs to be printed explicitly by e.g. `print(plotmap(object))` (R FAQ: Why do lattice/trellis graphics not work?).
plots the spectra, i.e. the intensities $spc$ over the wavelengths $\text{wavelength}$.
> plotspc (flu)

plots the spectra, i.e. the colour coded intensities $spc$ over the wavelengths $\text{wavelength}$ and the row number.
> plotmat (flu)

plots an intensity over a single other data column, e.g.
- calibration
- time series
- depth profile
> plotc (flu)

plots a false colour map, defined by a formula.
> levelplot (spc ~ x * y, chondro, aspect = "iso")
Warning: Only first wavelength is used for plotting
2 Arguments for plot

`hyperSpec`'s `plot` method uses its second argument to determine which of the specialized plots to produce. This allows some handy abbreviations. All further arguments are handed over to the function actually producing the plot.

```r
> plot (flu, "spc")
```

is equivalent to `plotspc (flu)`

```r
> plotmap (chondro)
```

plotmap is a specialized version of `levelplot`. It uses a single value (e.g. average intensity or cluster membership) over two data columns (default $x$ and $y$)

```r
> plotmap (chondro)
```

Function `plotvoronoi` is a special version of `plotmap` that produces Voronoi diagram of the `hyperSpec` object.

```r
> plotvoronoi (sample (chondro, 300), clusters ~ x * y)
```
plots mean spectrum ± 1 standard deviation
> plot (chondro.preproc, "spcmeansd")

plots median, 16th and 84th percentile for each wavelength. For Gaussian distributed data, 16th, 50th and 84th percentile are equal to mean ± standard deviation. Spectroscopic data frequently are not Gaussian distributed. The percentiles give a better idea of the true distribution. They are also less sensitive to outliers.
> plot (chondro.preproc, "spcprctile")

like "spcprctl" plus 5th and 95th percentile.
> plot (chondro.preproc, "spcprctl5")

> plot (flu, "c")
is equivalent to plotc (flu)
plot (x, "ts") plots a time series plot

> plot (laser [, , 405], "ts")
equivalent to plotc (laser, spc ~ t)

plot (x, "depth") plots a depth profile plot

> depth.profile <- new ("hyperSpec",
  +   spc = as.matrix (rnorm (20) + 1:20),
  +   data = data.frame (z = 1 : 20),
  +   labels = list (spc = "I / a.u.",
  +                     z = expression (\text{\mu m})
  +                     \text{lambda})),
> plot (depth.profile, "depth")
the same as plotc (laser, spc ~ z)

plot (x, "mat") plots the spectra matrix.

> plot (laser, "mat")
Equivalent to
> plotmat (laser)
A lattice alternative is:
> levelplot (spc ~ .wavelength * .row, laser)

plot (x, "map") is equivalent to plotmap (chondro)

> plot (chondro, "map")
plot (x, "voronoi")

> plot (sample (chondro, 300), "voronoi")
See plotvoronoi

3 Spectra

plotspc offers a variety of parameters for customized plots. To plot ...
in different colours

\[ \lambda / \text{nm} \]

\[ I / \text{a.u.} \]

\text{use} \ col = \text{vector.of.colours}
\text{use} \ col = \text{matlab.dark.palettne (6)}

\text{dots instead of lines}

\[ \Delta \nu / \text{cm}^{-1} \]

\[ I / \text{a.u.} \]

\text{use} \ lines.args = \text{list (pch = 20, type = "p")}
\text{use} \ lines.args = \text{list (pch = 20, type = "p")}

\text{mass spectra}

\[ m/z \]

\[ I / \text{a.u.} \]

\text{use} \ lines.args = \text{list (type = "h")}
\text{use} \ lines.args = \text{list (type = "h")}

\text{more spectra into an existing plot}

\[ \Delta \nu / \text{cm}^{-1} \]

\[ I / \text{a.u.} \]

\text{use add = TRUE}
\text{use add = TRUE, col = "blue"}
Summary characteristics

*func* may be used to calculate summary characteristics prior to plotting. To plot e.g. the standard deviation of the spectra, use:

```r
> plotspc (chondro.preproc, func = sd)
```

with different line at *I* = 0

zeroline takes a list with parameters to *abline*, NA suppresses the line.

```r
> plotspc (paracetamol,
+       zeroline = list (col = "red"))
```

adding to a spectra plot

plotspc uses base graphics. After plotting the spectra, more content may be added to the graphic by *abline*, *lines*, *points*, etc.

```r
> plot (laser, "spcmeansd")
> abline (v = c(405.0063, 405.1121, 405.2885, 405.3591),
+        col = c("black", "blue", "red", "darkgreen"))
```

3.1 Stacked spectra

stacked

use stacked = TRUE

```r
> plotspc (cluster.means,
+       col = cluster.cols,
+       stacked = TRUE)
```
Stacking groups of spectra

The spectra to be stacked can be grouped: `stacked = factor`. Alternatively, the name of the grouping extra data column can be used:

```r
> plot (cluster.meansd, 
+       stacked = "aggregate", 
+       fill = "aggregate", 
+       col = cluster.cols)
```

Manually giving yoffset

Stacking values can also be given manually as numeric values in `yoffset`:

```r
> plotspc (cluster.meansd, 
+       yoffset = rep (0:2, each = 3), 
+       col = rep (cluster.cols, each = 3))
```

Dense stacking

To obtain a denser stacking:

```r
> yoffsets <- apply (cluster.means \[ \], 2, diff) 
> yoffsets <- - apply (yoffsets, 1, min) 
> plot (cluster.means, yoffset = c (0, cumsum (yoffsets)), 
+       col = cluster.cols)
```

Elaborate example

```r
> yoffset <- apply (chondro.preproc, 2, quantile, c(0.05, 0.95)) 
> yoffset <- range (yoffset) 
> plot(chondro.preproc[1], 
+       plot.args = list (ylim = c (0, 2) * yoffset), 
+       lines.args = list (type = "n")) 
> yoffset <- (0:1) * diff (yoffset) 
> for (i in 1 : 3){
+       plot(chondro.preproc, "spcprctl5", yoffset = yoffset [i], 
+            col = "gray", add = TRUE) 
+       plot (chondro.preproc [i], yoffset = yoffset [i], 
+            col = matlab.dark.palette (3) [i], add = TRUE, 
+            lines.args = list (lwd = 2)) 
+     }
```

`plotspc` allows fine grained customization of almost all aspects of the plot. This is possible by
giving arguments to the functions that actually perform the plotting `plot` for setting up the plot area, `lines` for the plotting of the lines, `axis` for the axes, etc. The arguments for these functions should be given in lists as `plot.args`, `lines.args`, `axis.args`, etc.

4 Calibration Plots, (Depth) Profiles, and Time Series Plots

4.1 Calibration plots

Intensities over concentration

Plotting the Intensities of one wavelength over the concentration for univariate calibration:

```r
> plotc (flu [, , 450])
```

The default is to use the first intensity only.

Summary Intensities over concentration

A function to compute a summary of the intensities before drawing can be used:

```r
> plotc (flu, func = range, groups = .wavelength)
```

If `func` returns more than one value, the different results are accessible by `.wavelength`.

Conditioning: plotting more traces separately

```r
> plotc (flu [, , c (405, 445)], spc ~ c | .wavelength, +    cex = .3, scales = list (alternating = c(1, 1)))
```
Grouping: plot more traces in one panel

> plotc (flu [,, c (405, 445)], groups = .wavelength)

Changing Axis Labels (and other parameters)

Arguments for xyplot can be given to plotc:
> plotc (flu [,, 450],
+ ylab = expression (I ["450 nm"] / a.u.),
+ xlim = range (0, flu$c + .01),
+ ylim = range (0, flu$spc + 10),
+ pch = 4)

Adding things to the plot: customized panel function

As plotc uses the lattice function xyplot, additions to the plot must be made via the panel function:
> panelcalibration <- function (x, y, ..., clim = range (x), level = .9
+ panel.xyplot (x, y, ...)
+ lm <- lm (y ~ x)
+ panel.abline (coef (lm), ...)
+ cx <- seq (clim [1], clim [2], length.out = 50)
+ cy <- predict (lm, data.frame (x = cx),
+ interval = "confidence",
+ level = level)
+ panel.lines (cx, cy [,2], col = "gray")
+ panel.lines (cx, cy [,3], col = "gray")
+ }
> plotc (flu [,,405], panel = panelcalibration,
+ pch = 4, clim = c (0, 0.35), level = .99)

4.2 Time series and other Plots of the Type Intensity-over-Something
Other abscissae may be specified by explicitly giving the model formula:

```r
> plotc (laser [, , c(405.0063, 405.1121, 405.2885, 405.3591)],
+     spc ~ t,
+     groups = .wavelength,
+     type = "b",
+     col = c ("black", "blue", "red", "darkgreen"))
```

### 5 Levelplot

`hyperSpec`'s `levelplot` can use two special column names:

- `.wavelength` for the wavelengths
- `.row` for the row index (i.e. spectrum number) in the data

Besides that, it behaves exactly like `levelplot`. Particularly, the data is given as the second argument:

```r
> levelplot (spc ~ x * y, chondro)
```

#### factors as z

If the colour-coded value is a factor, the display is adjusted to this fact:

```r
> levelplot (clusters ~ x * y, chondro)
```

### 6 Spectra Matrix

It is often useful to plot the spectra against an additional coordinate, e.g. the time for time series, the depth for depth profiles, etc.

This can be done by `plot (object, "mat")`. The actual plotting is done by `image`, but `levelplot`
can produce spectra matrix plots as well and these plots can be grouped or conditioned.

**different palette**

![Palette Example]

```
> plot (laser, "mat", col = heat.colors (20))
> plotmat (laser, col = heat.colors (20))
```

**different y axis**

![Y Axis Example]

Using a different extra data column for the y axis:

```
> plotmat (laser, y = "t")
```

Alternatively, y values and axis label can be given separately.

```
> plotmat (laser, y = laser$t, ylab = labels (laser, "t"))
```

**contour lines**

![Contour Lines Example]

Contour lines may be added:

```
> plotmat (flu, col = matlab.dark.palette (20))
> plotmat (flu, col = "white",
+        contour = TRUE, add = TRUE)
```

**colour-coded points: levelplot with special panel function**

![Levelplot Example]

```
> require ("latticeExtra")
> barb <- do.call (collapse, barbiturates[1:50])
> barb <- orderwl (barb)
> levelplot (spc ~ .wavelength * z, barb,
+        panel = panel.levelplot.points,
+        cex = .35, col.symbol = NA,
+        col.regions = matlab.palette)
```
7 False-Colour Maps: plotmap

plotmap is a specialized version of levelplot. The spectral intensities may be summarized by a function before plotting (default: mean). The same scale is used for x and y axes (aspect = "iso").

plotting map

> plotmap (chondro)

plotting maps with other than x and y

specify the colour-coded variable, abscissa and ordinate as formula: colour.coded ~ abscissa * ordinate
> plotmap (chondro, spc ~ y * x)

colour-coded factors

> plotmap (chondro, clusters ~ x * y)
If the colour-coded variable is a factor, each level gets its own colour, and the legend is labeled accordingly.

different palette

To plot with a different palette, use argument col.regions.
> plotmap (chondro, clusters ~ x * y,
+       col.regions = cluster.cols)
Fine tuning lattice parameters

The plotting of color maps is done via R package `lattice` (aka Trellis graphic approach), which is highly customizable. Use `trellis.par.get` and `trellis.par.set` to get/set the settings for the current graphics device.

```r
> my.theme = trellis.par.get()
> names(my.theme)  # note how many parameters are tunable

[1] "grid.pars"     "fontsize"     "background"     "panel.background"
[5] "clip"          "add.line"     "add.text"       "plot.polygons"
[9] "box.dot"       "box.rectangle" "box.umbrella"   "dot.line"
[13] "dot.symbol"    "plot.line"    "plot.symbol"   "reference.line"
[17] "strip.background" "strip.shingle" "strip.border"   "superpose.line"
[21] "superpose.symbol" "superpose.polygon" "regions"       "shade.colors"
[25] "axis.line"     "axis.text"    "axis.components" "layout.heights"
[29] "layout.widths" "box.3d"       "par.title.text" "par.xlab.text"
[33] "par.ylab.text" "par.zlab.text" "par.main.text"  "par.sub.text"
```

Any of these parameters can be fine-tuned to produce the desired output. For example, parameter `my.theme$region` is responsible for the appearance of color maps, and it contains elements `$alpha` and `$col`. By changing this parameters you can create your own theme for plotting and pass it to the plotting function via `par.settings`.

```r
> my.theme$regions$col = grDevices::terrain.colors
> plotmap (chondro, par.settings = my.theme)
```

This plot uses a customized lattice theme.

```r
> my.theme$regions$col = grDevices::terrain.colors
> plotmap (chondro, par.settings = my.theme)
```

It is possible to persistently (i.e. inside of the current R session) set `lattice` parameters, so they would apply to all further plots. This is done via a call to `trellis.par.set`, for example `trellis.par.set(my.theme)`.

The current settings can be visualized via a call to `show.settings()`

```r
> # Display current trellis parameters
> show.settings()
```
An overview of different color palettes, and ways to create your own, can be found in the R color cheatsheet.

To plot a map of the average intensity at particular wavelengths use extraction:
```r
> plotmap(chondro.preproc[, , c(728, 782, 1098, 1240, 1482, 1577)],
  + col.regions = matlab.palette)
```

Conditioning

```r
> plotmap(chondro, 
  + spc ~ y * x | x > 5,
  + col.regions = matlab.palette(20))
```

Conditioning on .wavelength

```r
> plotmap(scores[, , 1:2],
  + spc ~ y * x | as.factor(.wavelength),
  + func = NULL,
  + col.regions = matlab.palette(20))
```
Conditioning on \( \text{wavelength} \)

Alternatively, use `levelplot` directly:

```r
> levelplot (spc ~ y * x | as.factor(.wavelength),
          + scores [,1:2],
          + aspect = "iso",
          + col.regions = matlab.palette(20))
```

Voronoi plot

Voronoi uses `panel.voronoi` from `latticeExtra[2]`. The tesselation is calculated by default using `deldir[3]`, but `tripack[4]` can also be used. `tripack` seems to faster in general, but may “hang” with certain data sets (particularly regular grids with missing spectra as in this example). Furthermore, it is not FOSS (free and open source software), so users are kindly asked to review `tripack`’s license before using it.

Mark missing spectra

If the spectra come from a rectangular grid, missing positions can be marked with this panel function:

```r
> mark.missing <- function (x, y, z, ...){
  + panel.levelplot (x, y, z, ...)
  + miss <- expand.grid (x = unique (x), y = unique (y))
  + miss <- merge (miss, data.frame (x, y, TRUE),
                  + all.x = TRUE)
  + miss <- miss [is.na (miss[, 3]),]
  + panel.xyplot (miss [, 1], miss [, 2], pch = 4, ...)
  + }
> plotmap (sample (chondro, length(chondro) - 20),
           + col.regions = matlab.palette(20),
           + col = "black",
           + panel = mark.missing)
```

Unevenly spaced measurement grid

The panel function used by `plotmap` defaults to `panel.levelplot.raster` which assumes an evenly spaced measurement grid. Even if the spectra are measured on a nominally evenly spaced grid, the actual stage position may be slightly varying due to positioning inaccuracy and some manufacturers (e.g. Kaiser) record the position reported by the stage rather than the position requested by the stage control.
This leads to weird looking output with holes, and possibly wrong columns:

```r
> plotmap (uneven)
```

The symptom of this situation are warnings about values in x and/or y not being equispaced; and that the output therefore may be wrong.

One possibility to obtain a correct map is using `plotvoronoi` instead which will construct a mosaic-like image with the respective “pixel” areas being centered around the actually recorded \(x\) and \(y\) position:

```r
> plotvoronoi (uneven)
```

Another possibility that underlines a point shape of the measurements is switching to `latticeExtra::panel.levelplot.points`:

```r
> plotmap (uneven, panel = panel.levelplot.points, 
+ cex = 0.75, col.symbol=NA)
```

Alternatively, the measurement raster positions can be rounded to their nominal raster, e.g.:

```r
> rx <- makeraster (uneven$x, start = -11.55, d = 1, tol = 0.3)
> uneven$x <- rx$x
> ry <- makeraster (uneven$y, start = -4.77, d = 1, tol = 0.3)
> uneven$y <- ry$y
> plotmap (uneven)
```

8 3D plots (with rgl)
rgl[5] offers fast 3d plotting in R. As rgl’s axis annotations are sometimes awkward, they may better be set manually:

```r
> laser <- laser [,404.8 : 405.6] / 10000
> laser$t <- laser$t / 3600
> cols <- rep (matlab.palette (nrow (laser)), nwl (laser))
> surface3d (y = wl (laser), x = laser$t, +
    z = laser$spc, col = cols)
> aspect3d (c (1, 1, 0.25))
> axes3d (c ("x+-", "y--", "z--"))
> axes3d ("y--", nticks = 25, labels= FALSE)
> mtext3d ("t / h", "x+-", line = 2.5)
> mtext3d ("lambda / nm", "y--", line = 2.5)
> mtext3d ("I / a.u.", edge = "z--", line = 2.5)
```

9 Using ggplot2 with hyperSpec objects

hyperSpec objects do not yet directly support plotting with ggplot2 [6]. Nevertheless, ggplot2 graphics can easily be obtained, and qplot* equivalents to plotspc and plotmap are defined:

**plot spectra with as.long.df**

```r
> qplotspc (flu) + aes (colour = c)
```

**Map with ggplot2**

```r
> qplotmap (chondro) +
  + scale_fill_gradientn ("spc", colours = matlab.palette ())
```

The two special columns .wavelength and .rownames contain the wavelength axis and allow to distinguish the spectra.

For more general plotting, as.long.df transforms a hyperSpec object into a long-form data.frame that is suitable for qplot, while as.t.df produces a data.frame where each spectrum is one column, and an additional first column gives the wavelength (see “plotting mean ± sd” below for an example). Long data.frames can be very memory consuming as they are of size nrow·nwl×(ncol+2) with
respect to the dimensions of the *hyperSpec* object. Thus, e.g. the *chondro* data set (2 MB) as *hyperSpec* object) needs 28 MB as long-format *data.frame*. It is therefore highly recommended to calculate the particular data to be plotted beforehand.

**Mean ± standard deviation with **\texttt{qplot2}**

\begin{verbatim}
> qplotspc (mean (chondro)) +
+ geom.ribbon (aes (ymin = mean + sd,
+ ymax = mean - sd,
+ y = 0, group = NA),
+ alpha = 0.25,
+ data = as.t.df (mean_sd (chondro)))
\end{verbatim}

Note that \texttt{qplotspc} specifies aesthetics \texttt{y = spc} and \texttt{groups = .rownames}, which do not have corresponding columns in the \texttt{data.frame} returned by \texttt{as.t.df}. These aesthetics must therefore be set manually in the aesthetics definition in \texttt{geom.ribbon} (or any other \texttt{geom_} that uses \texttt{as.t.df}). Otherwise, errors occur that object \texttt{spc} (and/or \texttt{.rownames}) cannot be found.

**Cut spectra with **\texttt{qplot2}**

\begin{verbatim}
> qplotspc (paracetamol / 1e4,
+ wl.range = c( min ~ 1800, 2800 ~ max)) +
+ scale_x_continuous (breaks = seq (0, 3200, 400))
\end{verbatim}

\section{10 Troubleshooting}

10.1 No output is produced

\texttt{plotmap}, \texttt{plotvoronoi}, \texttt{levelplot}, and \texttt{plotc} use \texttt{lattice} functions. Therefore, in loops, functions, Sweave chunks, etc. the lattice object needs to be printed explicitly by \texttt{print (plotmap (object))} (R FAQ: Why do lattice/trellis graphics not work?). The same holds for \texttt{ggplot2} graphics.

For suggestions how the lattice functions can be redefined so that the result is printed without external print command, see the file \texttt{vignettes.defs}.

\section{11 Interactive Graphics}

*hyperSpec* offers basic interaction, \texttt{spc.identify} for spectra plots, and \texttt{map.identify} and \texttt{map.sel.poly} for maps. The first two identify points in spectra plots and map plots, respectively. \texttt{map.sel.poly} selects the part of a *hyperSpec* object that lies inside the user defined polygon.
11.1 spc.identify: finding out wavelength, intensity and spectrum

spc.identify allows to measure points in graphics produced by plotspc. It works correctly with reversed and cut wavelength axes.

> spc.identify (plotspc (paracetamol, wl.range = c (600 ~ 1800, 2800 ~ 3200), zoffset = 800))

The result is a data.frame with the indices of the spectra, the wavelength, and its intensity.

11.2 map.identify: finding a spectrum in a map plot

map.identify returns the spectra indices of the clicked points.

> map.identify (chondro)

11.3 map.sel.poly: selecting spectra inside a polygon in a map plot

map.sel.poly returns a logical indicating which spectra are inside the polygon drawn by the user:

> map.sel.poly (chondro)

11.4 Related functions provided by base graphics and lattice

For base graphics (as produced by plotspc), locator may be useful as well. It returns the clicked coordinates. Note that these are not transformed according to xoffset & Co.

For lattice graphics, grid.locator may be used instead. If it is not called in the panel function, a preceding call to trellis.focus is needed:

> plot (laser, "mat")
> trellis.focus ()
> grid.locator ()

identify (or panel.identify for lattice graphics) allows to identify points of the plot directly. Note that the returned indices correspond to the plotted object.

References

Session Info

sysname "Linux"
release "5.11.0-27-generic"
nodename "cx17007"
machine "x86_64"
login "unknown"
user "cb"
effective_user "cb"

R version 4.1.1 (2021-08-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.3 LTS

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3

locale:
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C LC_TIME=de_DE.UTF-8
[4] LC_COLLATE=C LC_MONETARY=de_DE.UTF-8 LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=de_DE.UTF-8 LC_NAME=C LC_ADDRESS=C
[10] LC_TELEPHONE=C LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] tools   grid    stats   graphics grDevices utils   datasets   methods   base
other attached packages:
[1] latticeExtra_0.6-29 baseline_1.3-1 MASS_7.3-54 hyperSpec_0.100.0
[5] xml2_1.3.2 ggplot2_3.3.5 lattice_0.20-44

loaded via a namespace (and not attached):
[1] pillar_1.6.1 compiler_4.1.1 RColorBrewer_1.1-2 R.methodsS3_1.8.1 R.utils_2.10.1
[6] testthat_3.0.4 digest_0.6.27 jsonlite_1.7.2 lifecycle_1.0.0 tibble_3.1.3
[11] gtable_0.3.0 R.cache_0.15.0 pkgconfig_2.0.3 png_0.1-7 rlang_0.4.11
[16] rstudioapi_0.13 crosstalk_1.1.1 SparseM_1.81 mvtnorm_1.1-2 xfun_0.24
[21] R.rsp_0.44.0 knitr_1.33 withr_2.4.2 dplyr_1.0.7 htmlwidgets_1.5.3
[26] generics_0.1.0 vcrs_0.3.8 tidyselect_1.1.1 limSolve_1.5.6 glue_1.4.2
[31] R_6_2.5.0 jpeg_0.1-9 plotrix_3.8-1 fansi_0.5.0 deldir_0.2-10
[36] farver_2.1.0 purrr_0.3.4 magrittr_2.0.1 htmltools_0.5.1.1 scales_1.1.1
[41] ellipsis_0.3.2 lpSolve_5.6.15 colorspace_2.0-0 labeling_0.4.2 quadprog_1.5-8
[46] utf8_1.2.2 lazyeval_0.2.2 munsell_0.5.0 crayon_1.4.1 R.oo_1.24.0