ChemoSpec: An R Package for Chemometric Analysis of Spectroscopic Data (Chromatograms too!)
Version 1.48-1

Bryan A. Hanson*
e-mail: hanson@depauw.edu

with contributions from Matt J. Keinsley

DePauw University
Department of Chemistry & Biochemistry
Greencastle Indiana USA

github.com/bryanhanson/ChemoSpec
CRAN.R-project.org/package=ChemoSpec

October 10, 2011

Abstract

ChemoSpec[1] is a collection of functions for plotting spectra (NMR, IR etc, as well as chromatograms) and carrying out various forms of top-down exploratory data analysis, such as hierarchical cluster analysis (HCA), principal components analysis (PCA) and model-based clustering. Robust methods appropriate for this type of high-dimensional data are employed. ChemoSpec is designed to facilitate comparison of samples from treatment and control groups. It is designed to be user friendly and suitable for people with limited background in R. This vignette gives some background on ChemoSpec and takes the reader through a typical workflow.

Contents

1   Introduction

Chemometrics, as defined by Varmuza and Filzmoser[2], is

> ". . . the extraction of relevant information from chemical data by mathematical and statistical tools."

This is an appropriately broad definition, considering the wealth of questions and tasks that can be treated by chemometric approaches. In our case, the focus is on spectral data sets, which typically have many variables (frequencies) and relatively few samples. Such multivariate, *high p, low n* data sets present some algorithmic challenges, but these have been addressed by knowledgeable folks. In particular, for both the practical and theoretical background to multivariate chemometric analysis, I strongly recommend the Varmuza/Filzmoser book. Some of the functions described here are not much more than wrappers for the functions they have made available to the R community in their packages.

ChemoSpec was developed for the chemometric analysis of spectroscopic data, such as UV-Vis, NMR or IR data (it also works with chromatographic data, see below). The approach is entirely exploratory and unsupervised, in other words, "top-down"[3]. I developed it while beginning a new research focus on plant metabolomics, and I needed software to analyze the data I was collecting. My research involves ecological experiments on plant stress, so ChemoSpec was designed to accommodate samples that have different histories, i.e., they fall into different classes, categories or groups. Examples would be treatment and control groups, or simply different specimens (red flowers vs. blue flowers). Since my research is done with undergraduates, who are true novices with R, ChemoSpec is designed to be as user friendly as possible, with plenty of error checking, helpful warnings and a consistent interface. It also produces graphics that are consistent in style and annotation, and are suitable for use in publications and posters. Careful attention was given to writing the documentation for the functions, but this vignette serves as the best starting point for learning data analysis with ChemoSpec.

The centerpiece of ChemoSpec is the Spectra object. This is the place where your data is stored and made available to R. Once your data in stored this way and checked, all analyses are easily carried out. ChemoSpec currently ships with several built-in data sets; we'll use one called CuticleIR for our demonstrations. You will see in just a moment how to access it and inspect it.

I assume you have at least a bare-bones knowledge of R as you begin to learn ChemoSpec, and have a good workflow set up. For detailed help on any function discussed here, type ?function_name at the console.

Finally, some conventions for this document: names of R "objects" such as packages, functions, function arguments, and data sets are in `typewriter` font. The commands you issue at the console and the output are in *`slanted typewriter font`*. Names of files stored on a computer are shown in blue to distinguish them from R objects which are kept in RAM.

By the way, if you try `ChemoSpec` and find it useful, have questions, have opinions, or have suggestions, please do let me know. The current version has already been improved by users.

2 A Sample Exploration

This sample exploration is designed to illustrate a typical `ChemoSpec` workflow. The point is to illustrate how to carry out the commands, what options are available and typically used, and the order in which one might do the analysis. The `CuticleIR` data set will be used for illustration only – we are not trying to analyze it to reach useful conclusions. You may wish to put your versions of these commands into a script file that you can source as you go along. This way you can easily make changes, and it will all be reproducible. To do this, open a blank R document, and type in your commands. Save it as something like My First ChemoSpec.R. Then you can either cut and paste portions of it to the console for execution, or you can source the entire thing:

```
> source("My First ChemoSpec.R")
```

Alternatively, look through your computer after installing `ChemoSpec` for a file called ChemoSpec.Rnw. Point `R` to the folder containing that file, then at the console issue > `Stangle("ChemoSpec.Rnw")`. R will extract all the code used in this vignette into a file which you can then study and modify.

2.1 Getting Data into ChemoSpec

Currently, there is only one means of moving raw data sets into `ChemoSpec`, and that is the function `getManyCsv` (it is relatively easy to write analogous functions for other formats). This function assumes that your raw data files are formatted as .csv files, and contain only the data itself, in two columns.[1] The first column should be the frequency values, and this column must be the same for all files (as it will be if these are data sets from the same instrument and experimental parameters). The second column should contain the intensity values. There should not be a header row. If your data set contains treatment and control groups, or any analogous class/group information, this information should be encoded in the file names. `getManyCsv` argument `gr.crit` will be the basis for a grep process on the file names, and from there, each file, representing a sample, will be assigned to a group and be assigned a color as well. If your samples don't fall into groups, that's fine too, but you still have to give `gr.crit` something to go on—just give it one string that is common to all the file names. Obviously, this approach encourages one to name the files as they come off the instrument with forethought as to how they will be analyzed, which in turn depends upon your experimental design. Nothing wrong with having a plan! Remember that `getManyCsv` acts on all .csv files it finds in a directory, so don't have any extra .csv files hanging around. The output of `getManyCsv` is a `Spectra` object, which is R-speak for a file, readable by R, that contains not only your data, but other information about the experiment, as provided by you via the arguments to `getManyCsv`.

Here's a typical example (we have to talk hypothetically because I don't have your data). Let's say you had a folder containing 30 NMR files of flower essential oils. Imagine that 18 of these were from one hypothetical subspecies, and 12 from another. Further, let's pretend that the question under investigation has something to do with the taxonomy of these two supposed subspecies, in other words, an investigation into whether or not they should be considered subspecies at all. If the files were named like this:

sspA1.csv ... sspA18.csv and sspB1.csv ... sspB12.csv

Then the following command should process the files and create the desired `Spectra` object:

```
> getManyCsv(gr.crit = c("sspA", "sspB"), gr.cols = c("red3", "dodgerblue4"),
+ freq.unit = "ppm", int.unit = "peak intensity", descrip = "Subspecies Study",
+ out.file = "subspecies")
```

---

[1]Users in the EU have different standards for a .csv file: they are delimited by semi-colons, and a comma is used where a decimal point is used in the United States. For these files, use the argument `format = "csv2"` to read the files properly.

This causes getManyCsv to read the file names for the strings sspA and sspB and use these to assign the samples into groups. Samples in sspA*.csv files will be assigned the color red3 and sspB*.csv will be assigned dodgerblue4 (see the help file for some thinking-ahead about colors; ?colorSymbol at the console). After running this command, a new file called subspecies.RData will be in your directory, and you can access the data set and give it whatever name you like as follows:

```
> SubspeciesNMR <- loadObject("subspecies.RData")
```

Now it is ready to use.

Working with Chromatograms

While all the language in this vignette and in the package are geared toward analysis of spectra, ChemoSpec can also work with chromatograms as the raw data. In this case, time replaces frequency of course, but other than that the analysis is virtually the same. So the only real difference is when you issue the command getManyCsv, you will give the frequency unit along these lines: freq.unit = "time (minutes)".

Built-in Data Sets

ChemoSpec ships with several built-in data sets. CuticleIR is the set used for this vignette; it is a series of IR spectra of the cuticle (leaf surface) of the plant *Portulaca oleracea*. The data were taken by gently pinning the leaf against an ATR sampling device. The plants were grown at two different temperatures, and two different genotypes (varieties) were used (a classic G x E, genotype by environment, experiment). In the plots in this vignette you will see the data colored by these categories: GE for example means "golden" genotype and "experimental' conditions (higher temperature, to be precise). For more details about this data set, type > ?CuticleIR at the console.

There are also two data sets from a unpublished study of the medicinal plant Saw Palmetto (*Serenoa repens*). One is a set of IR spectra, the other a set of NMR spectra on the same samples. For more detail on these data sets, type > ?SrE.IR at the console.

Color and Symbol Options

In ChemoSpec, the user may use any color name/format known to R. For ease of comparison, it would be nice to plan ahead and use the same color scheme for all your plots. However, a difficulty to bear in mind is that R plots are generally on a white background, so pale colors should be avoided, while GGobi, which is used by functionplotScoresG, plots on a black background (interactively), so dark colors should be avoided. Hence some compromise is necessary.

In addition to colors, "Spectra" objects also contain a list of symbols, and alternative symbols. These are useful for plotting in black and white, or when color-blind individuals will be viewing the plots. The alternative symbols are simply lower-case letters, as these are needed for plotScoresRGL, and other rgl-graphics driven functions which cannot plot traditional symbols.

NOTE affecting the next section: as of October 10, 2011 GGobi is on "life support" and the communication of color information between ChemoSpec and GGobi is not working perfectly. You are on your own!

If you plan to use rggobi and GGobi to view the data later, keep in mind that GGobi only uses certain color schemes (although there are many options) and in interactive operation plots on a black background. In the case of ChemoSpec, two particular options have been hard-coded into the function plotScoresG for simplicity. If you plan to use plotScoresG, you may wish to choose from one of these two color schemes before you begin if you want all your graphics to use the same scheme. Keep in mind that these colors must be used in order (though you can use the order of argument gr.crit to associate a particular group with a particular color:

primary scheme: c("red3", "dodgerblue4", "forestgreen", "purple4", "orangered", "yellow", "orangered4", "violetred2")

pastel scheme: c("seagreen", "brown2", "skyblue2", "hotpink3", "chartreuse3", "darkgoldenrod2", "lightsalmon3", "gray48")

Example 1 shows these colors. Finally, the current color scheme of a Spectra object may be determined using sumSpectra or changed using conColScheme.

Example 1: Recommended Color Sets in ChemoSpec

ChemoSpec Primary Scheme

ChemoSpec Pastel Scheme

2.2 Preliminary Inspection of Data

One of the first things you should do, and this is very important, is to make sure your data is in good shape. First, you can summarize the data set you created, and verify that the data ranges etc look like you expect them to:

> *sumSpectra(SubspeciesNMR)*

This will display quite a bit of information about your data set. Here are the commands and the result using the built-in data set CuticleIR:

> *data(CuticleIR) # makes the data available*
> *sumSpectra(CuticleIR)*

```
 Kelly's Complete IR Data Set, Summer 2009

        There are 157 spectra in this set.
        The y-axis unit is Absorbance.

        The frequency scale runs from 501.4261 to 3596.768 Wavenumbers
        There are 1242 frequency (x-axis) data points.
        The frequency resolution is 1.9286 Wavenumbers/point.

        This data set is not continuous along the frequency axis.
        Here are the data chunks:

   beg.freq end.freq      size beg.indx end.indx
1  501.4261 1797.420 1295.994        1      673
2 2501.3450 3596.768 1095.423      674     1242

        The spectra are divided into 4 groups:

  group no.        color symbol alt.sym
1    GC  16 dodgerblue4      2       b
```
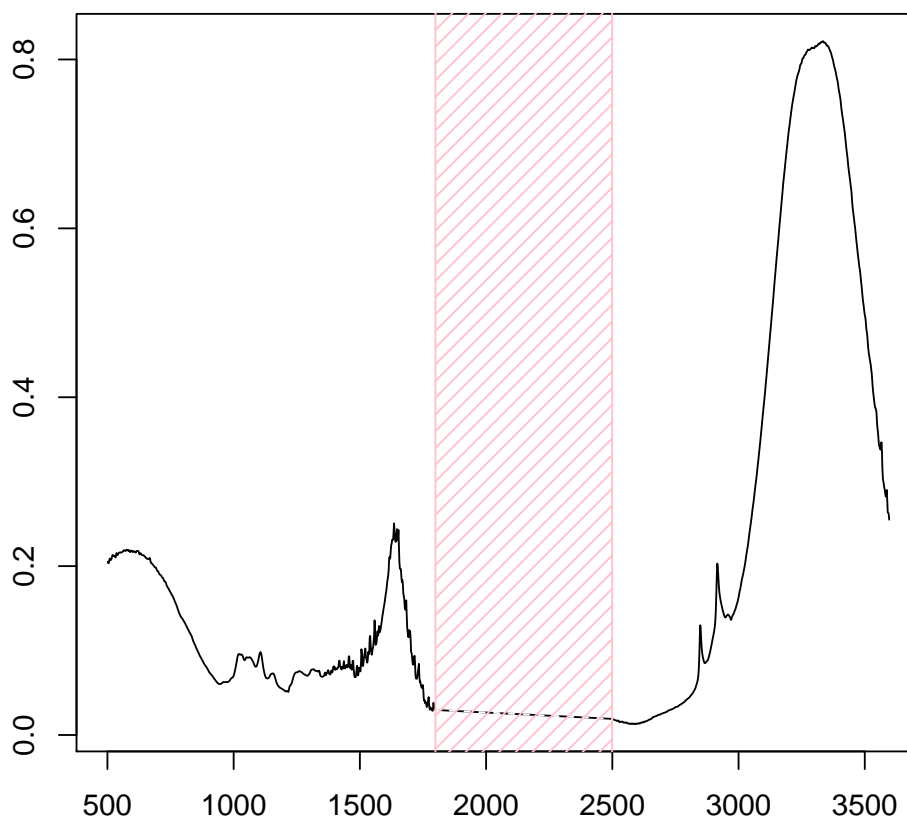
Example 2: Procedure to Find Gaps in a Data Set

```
> check4Gaps(CuticleIR$freq, CuticleIR$data[1,], plot = TRUE)
   beg.freq end.freq     size beg.indx end.indx
1  501.4261 1797.420 1295.994        1      673
2 2501.3450 3596.768 1095.423      674     1242
```

Gaps in Frequency Data


marked regions are skipped in data set

```
2    GE  35      purple4    15      d
3    TC  69         red3     1      a
4    TE  37 forestgreen      3      c
```

```
*** Note: this data is an S3 object of class 'Spectra'
```

Notice that sumSpectra has identified a gap in the data set; this is because a range of frequencies that contain no useful information were removed to compact the data—what's the point of analyzing baseline anyway? You can see this gap in the data as shown in Example 2 (sumSpectra checks for gaps, but doesn't produce the plot); both the numerical results and a figure are provided. Again, the point at this stage is to verify that your data looks like you expect it to. If there are problems, I'll tell you how to fix them in the next few sections.

Example 3: Plotting Spectra

```
> plotSpectra(CuticleIR, title = "Cuticle IR Spectra Demo",
+ which = c(10, 11, 40, 41, 100, 101, 140, 141, 150, 151),
+ yrange = c(0, 10), offset = 0.8, lab.pos = 2000)
```

Cuticle IR Spectra Demo
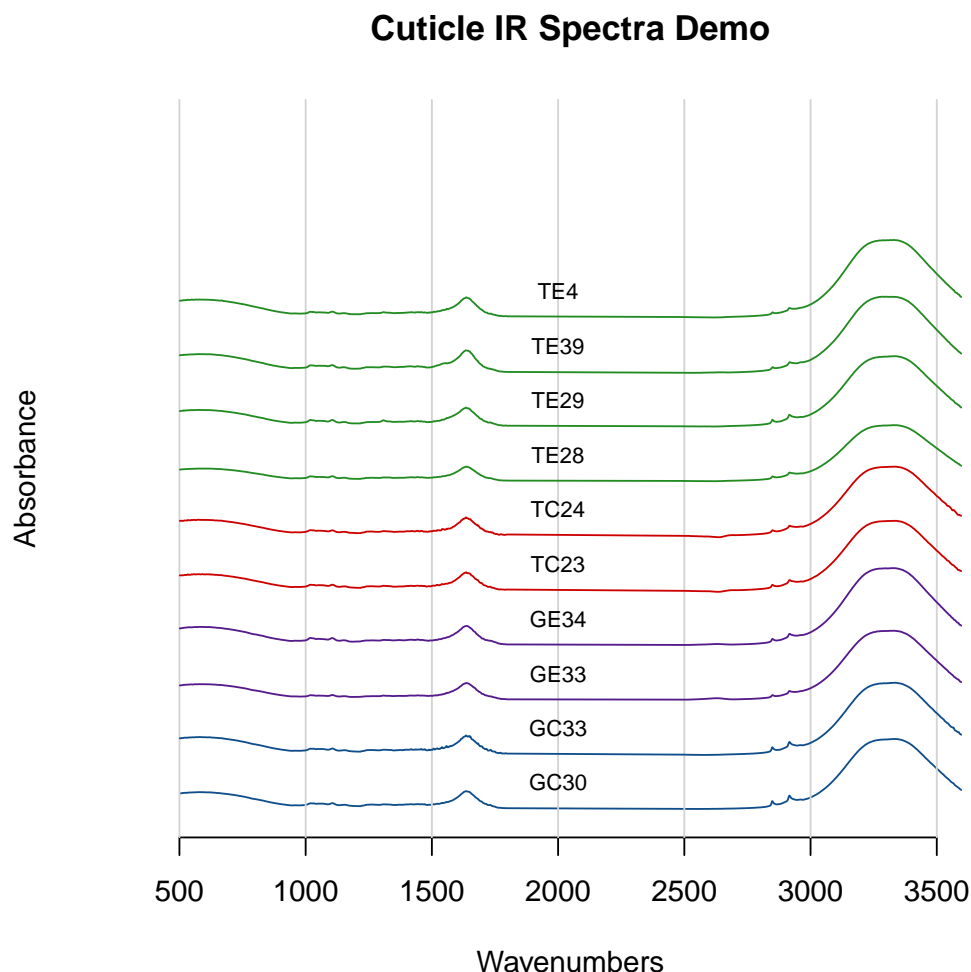
2.2.1 Plotting the Spectra

Assuming that everything looks good so far, it's time to plot the spectra and inspect them. Good practice would be to check every spectrum for artifacts and other potential problems, which might take a while; hopefully you looked at them when you originally recorded them. The basics of creating a plot are shown in Example 3.

Depending upon the intensity range of your data set, and the number of spectra to be plotted, you have to manually adjust the arguments `yrange`, `offset` and `amplify`, but this usually only takes a few iterations. Keep in mind that `offset`, and `amplify` are multiplied in the algorithm, so if you increase one, you may need to decrease the other. Suppose that you wanted to focus just on the hydrocarbon region of these spectra; you can add an argument called `xlim`. To demonstrate, let's look at fewer spectra, and at higher amplitude, so we can see details, as shown in Example 4.

The argument `which` in `plotSpectra` takes a numerical list of the spectra you wish to plot— you can think of this as the row number if you imagine each spectra to be a row in a matrix, with intensities in the columns (with each column corresponding to a particular frequency value). You may be wondering how to determine which particular sample is in each row. This is best accomplished with a grep command. For instance, if you wanted to know what row sample GE7 was in, the following command would locate it for you:

Example 4: Zooming in on a Spectral Region

```
> plotSpectra(CuticleIR, title = "Cuticle IR Spectra: Detail of Hydrocarbon Region",
+ which = c(10, 40, 100, 151), amplify = 10.0, xlim = c(2750, 3000),
+ yrange = c(0, 5), offset = 0.1, lab.pos = 2775)
```

Cuticle IR Spectra: PCA Score Plot
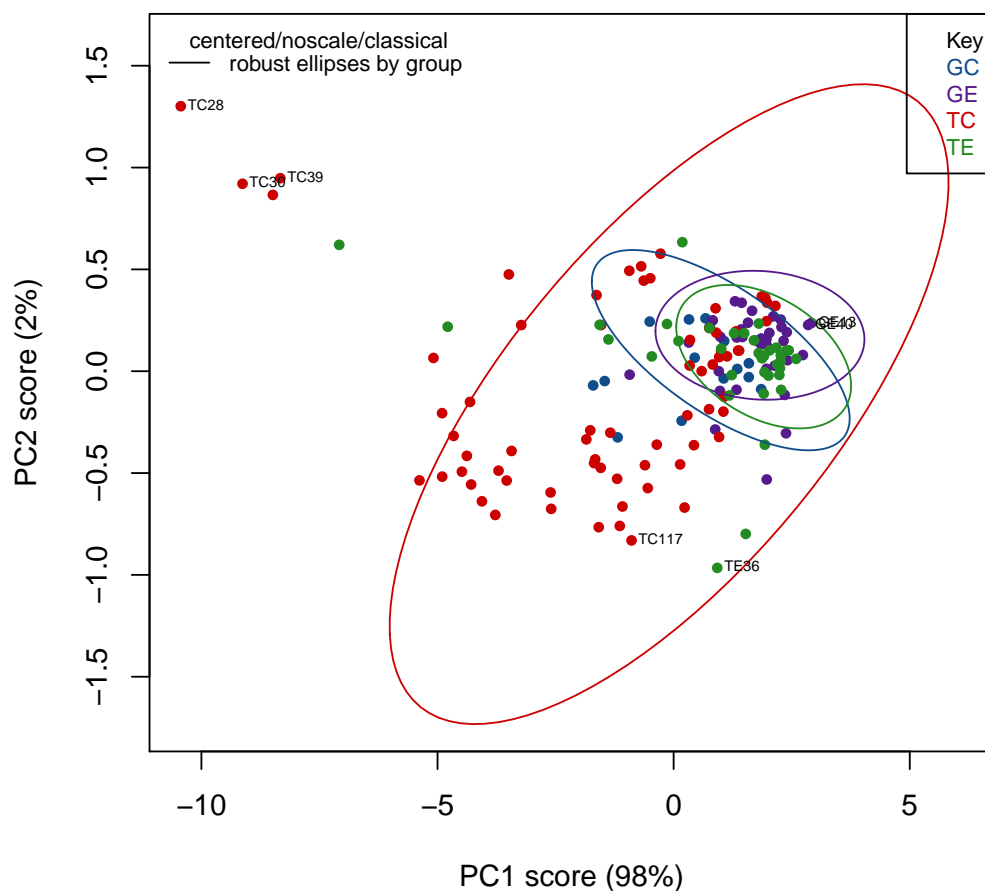

```
> grep("GE7", CuticleIR$names)
```

```
[1] 49
```

See the discussion in the next section for more details on using grep effectively.

These sample plots display the IR spectra in two ways that may be upsetting to some readers: First, the x-axis is "backwards", because the underlying spectra were originally saved with an ascending frequency axis (which is not always the case). This is readily fixed by supplying the `xlim` argument in the desired order, e.g. `xlim = c(3000, 2750)` in the previous example. Second, the vertical scale in these examples is absorbance. When using IR for structural elucidation, the vertical axis is typically %T, with the peaks pointing downward. You don't have that choice in `ChemoSpec` because the absorbance mode is the appropriate one for chemometrics. Get used to it.

2.2.2 Identifying & Removing Problematic Samples

In the process of plotting and inspecting your spectra, you may find some spectra/samples that have problems. Perhaps they have instrumental artifacts. Or maybe you have decided to eliminate one subgroup of samples from your data

set to see how the results differ. To remove a particular sample, or samples meeting a certain criteria, you use the removeSample function. This function uses a grepping process based on its rem.sam argument, so you must be careful due to the greediness of grep. Let's imagine that sample TC138 has artifacts and needs to be removed. The command would be:

```
> noTC138 <- removeSample(CuticleIR, rem.sam = c("TC138"))
> sumSpectra(noTC138)

 Kelly's Complete IR Data Set, Summer 2009

        There are 156 spectra in this set.
        The y-axis unit is Absorbance.

        The frequency scale runs from 501.4261 to 3596.768 Wavenumbers
        There are 1242 frequency (x-axis) data points.
        The frequency resolution is 1.9286 Wavenumbers/point.

        This data set is not continuous along the frequency axis.
        Here are the data chunks:

   beg.freq end.freq     size beg.indx end.indx
1  501.4261 1797.420 1295.994        1      673
2 2501.3450 3596.768 1095.423      674     1242

        The spectra are divided into 4 groups:

  group no.        color symbol alt.sym
1    GC  16 dodgerblue4      2       b
2    GE  35     purple4     15       d
3    TC  68        red3      1       a
4    TE  37 forestgreen      3       c

*** Note: this data is an S3 object of class 'Spectra'
```

The sumSpectra command is optional, but confirms that there are now one fewer spectra in the set. You could also re-grep for the sample name to verify that it is not found. If you wanted to remove sample TE2, and you tried to remove it this way, you would end up removing 10 spectra, as the string "TE2" occurs in quite a few places. You can check this in advance with the grep function itself:

```
> TE2 <- grep("TE2", CuticleIR$names)
> CuticleIR$names[TE2] # gives the name(s) that contain "TE2"

 [1] "TE2"  "TE20" "TE21" "TE22" "TE23" "TE24" "TE25" "TE26" "TE27" "TE28"
[11] "TE29"

> TE2 # gives the corresponding indicies

 [1] 131 132 133 134 135 136 137 138 139 140 141
```

This is what is meant by "grep is greedy". In this situation, you have three choices:

1. You could manually remove the problem samples (> str(CuticleIR) would give you an idea of how to do that; see also below under Hierarchical Cluster Analysis).

2. removeSample also accepts indices of samples, so you could grep as above, note the index of the sample you actually want to remove, and use that in rem.sam.

3. If you know a bit about grep, you can pass a more sophisticated search pattern to rem.sam. For instance, in the example above where we wanted to remove TE2, but "caught" some other sample names, you could change the search pattern as follows, which tells grep to find a string that ends in TE2 (so it could still begin with xyzTE2 and be found):

```
> newTE2 <- grep("TE2\\>",  CuticleIR$names)
> CuticleIR$names[newTE2] # gives the name(s) found

[1] "TE2"

> newTE2 # gives the corresponding indices

[1] 131
```

You can also take advantage of this greediness. For instance, using `rem.sam = c("TC")` would remove all the samples in this category, regardless of any other characters in the string, as long as T and C are next to each other (this goes back to planning your experiments, and how clever you were in naming your samples).

2.2.3   Identifying & Removing Regions of No Interest

Many spectra will have regions that should be removed. It may be an uninformative, interfering peak like the water peak in ^{1}H NMR, or the CO_2 peak in IR. Or, there may be regions of the spectra that simply don't have much information — they contribute a noisy baseline and not much else. An example would be the region from about 1,800 or 1,900 cm^{-1} to about 2,500 cm^{-1} in IR, a region where there are typically no peaks except for the CO_2 stretch, and rarely (be careful!) alkyne stretches.

Finding these regions might be pretty simple, a matter of inspection coupled with your knowledge of spectroscopy. Another approach is to use the function `specSurvey` to examine the entire set of spectra. This function computes the standard deviation of the intensities at a particular frequency across the data set, and plots this against frequency. Regions where there is not much variation in the intensity will show up as unvarying baseline, and these regions are candidates for removal. Example 5 demonstrates the process.

Compare this to the figure produced by `check4Gaps` (Example 2); you'll see that the gap region is rendered by `specSurvey` by connecting the endpoints of each data chunk, producing what appears to be a region of little interest. All other areas have varying standard deviations, with the OH peak dominating, and no regions seem uninteresting, so let's keep them for now. However, if you wanted to remove certain peaks, the function `removeFreq` is the tool to use. In the `CuticleIR` data set, the water peak (above about 3,100 cm^{-1}) varies quite a bit, and may be just an artifact. We could remove as follows, ending with `sumSpectra` to verify what you wanted to happen, happened. Note that there are fewer frequency points now.

```
> noOH <- removeFreq(CuticleIR, rem.freq = CuticleIR$freq > 3100)
> sumSpectra(noOH)

 Kelly's Complete IR Data Set, Summer 2009

        There are 157 spectra in this set.
        The y-axis unit is Absorbance.

        The frequency scale runs from 501.4261 to 3099.199 Wavenumbers
        There are 984 frequency (x-axis) data points.
        The frequency resolution is 1.9286 Wavenumbers/point.

        This data set is not continuous along the frequency axis.
        Here are the data chunks:

   beg.freq end.freq      size beg.indx end.indx
1   501.4261 1797.420 1295.994        1      673
2  2501.3450 3099.199  597.854      674      984

        The spectra are divided into 4 groups:

  group no.        color symbol alt.sym
1    GC  16 dodgerblue4      2       b
2    GE  35      purple4     15       d
3    TC  69         red3      1       a
```
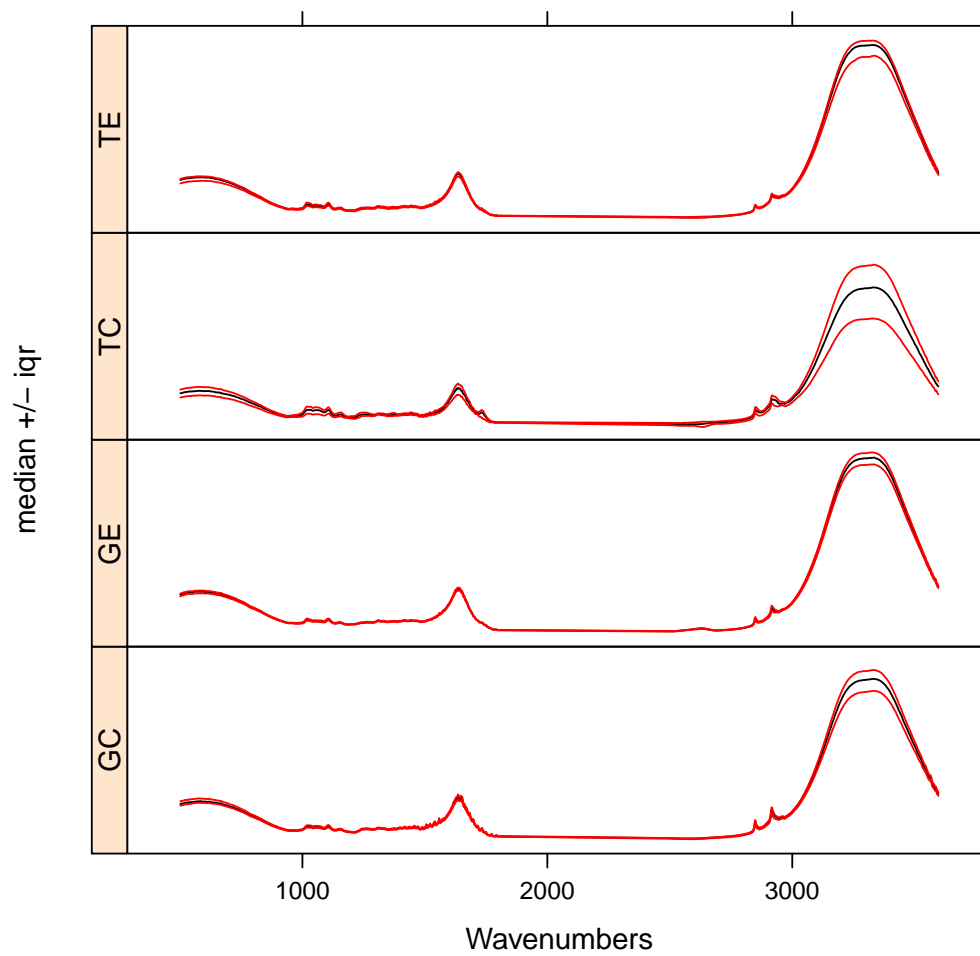
Example 5: Checking for Regions of No Interest

```
> specSurvey(CuticleIR, method = "iqr", title = "Cuticle IR Spectra")
```

Cuticle IR Spectra

Example 6: Verifying removeFreq Worked as Desired

```
> plotSpectra(noOH, title = "Cuticle IR Spectra, No Hydroxyl Peak?",
+ which = c(10, 40,  100,  140 , 150),
+ yrange = c(0, 5), offset = 0.8, lab.pos = 2000, xlim = c(500, 3600))
```

Cuticle IR Spectra, No Hydroxyl Peak?


```
4    TE  37 forestgreen     3      c
```
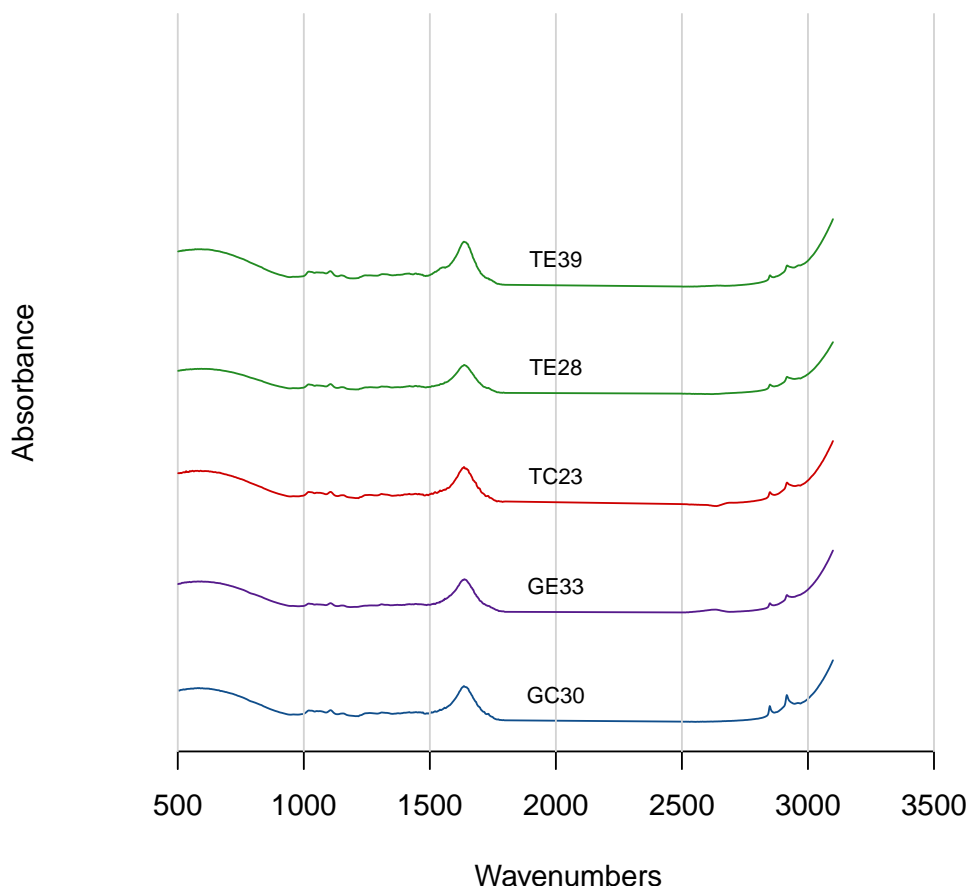
```
*** Note: this data is an S3 object of class 'Spectra'
```

You could also plot the new spectra if desired, to make absolutely certain you removed what you wanted. In this case, we can make the point more strongly by plotting with `xlim` corresponding to the original data range, not that of the modified spectra, as shown in Example 6. We indeed did accomplish what we set out to do, though we might consider trimming off more frequencies, perhaps down to 3,000 cm^{-1}. You could also run `check4Gaps` as another check.

2.3   Data Pre-Processing Options

There are a number of data pre-processing options available for your consideration. The main choices are whether to normalize the data, whether to bin the data, and whether to scale the data. Data scaling is handled by the PCA routines, see Section 2.5. Normalization is handled by the `normSpectra` function. Usually one normalizes data in which the sample preparation procedure may lead to differences in concentration, such as body fluids that might have been diluted during handling, or that vary due to the physiological state of the organism studied. The `CuticleIR` data set is taken using an

ATR device in which the leaves are pinned against the IR analyzer, and no dilution is possible, so normalization probably isn't really appropriate. Currently, there is only one means of normalizing, and that is to divide each point (frequency) in a spectrum by the sum of all points in that spectrum. Other means of normalizing can be readily added if they affect all points in the same way. Normalization is accomplished by:

```
> normCIR <- normSpectra(CuticleIR)
```

But remember, this doesn't make sense for this data set. The literature contains a number of useful discussions about normalization issues.[2, 4–7]

Another type of pre-processing that you may wish to consider is binning or bucketing, in which groups of frequencies are collapsed into one frequency value, and the corresponding intensities are summed. There are two reasons for doing this. One is to compact the data, but the algorithms in R are quite fast, and data sets of the size of CuticleIR don't slow it down much. The other reason is to compensate for shifts in very narrow peaks from sample to sample. This is typically done in ^{1}H NMR because changes in dilution, ionic strength, or pH can cause slight shifts. Spectra with broad, rolling peaks won't have this problem (UV-Vis for example). The function binBuck is your friend:

Compare the results here with the sumSpectra of the full data set (Section 2.2. In particular note that the frequency resolution has gone down due to the binning process. ChemoSpec uses the simplest of binning algorithms: after perhaps dropping a few points (with a warning) to make your data set divisible by the specified bin.ratio, data points are replaced by the average frequency and the sum of the grouped intensities. Depending upon the fine structure in your data and the bin.ratio this might cause important peaks to be split between different bins. There are more sophisticated binning algorithms in the literature that try to address this, but none are currently implemented in ChemoSpec.[8, 9]

2.4   Hierarchical Cluster Analysis

Hierarchical cluster analysis (HCA from now on) is a clustering method (surprise!) in which "distances" between samples are calculated and displayed in a dendrogram (a tree-like structure; these are also used in evolution and systematics where they are called cladograms). The details behind HCA can be readily found elsewhere (Chapter 6 of [2] is a good choice). With ChemoSpec you have access to any of the methods available in R function hclust; for more details, type ?hclust at the console. To demonstrate, let's first remove some samples from CuticleIR so the resulting plot is a bit less cluttered. Then we'll carry out the HCA itself. The sample removal will be done manually; this process and the results are shown in Example 7.

The result is a dendrogram. The vertical scale represents the numerical distance between samples. For this data set, there is no obvious clustering by group. Note that the function hcaScores does the same kind of analysis using the results of PCA, rather than the raw spectra. It is discussed in the next section.

2.5   Principal Components Analysis

Principal components analysis (PCA from now on) is the real workhorse of exploratory data analysis. It makes no assumptions about group membership, but clustering (possibly in high dimensions) of the resulting sample scores can be very helpful in understanding your data. The theory and practice of PCA is covered well elsewhere (Chapter 3 of [2] is an excellent choice). Here, we'll concentrate on using the PCA methods in ChemoSpec. Briefly however, you can think of PCA as determining the minimum number of components necessary to describe a data set, in effect, removing noise. Think of a typical spectrum: some regions are clearly just noise. Further, a typical spectroscopic peak spans quite a few frequency units as the peak goes up, tops out, and then returns to baseline. Any one of the points in a particular peak describe much the same thing, namely the intensity of the peak. Plus, each frequency within a given peak envelope is correlated to every other frequency in the envelope (they rise and fall in unison as the peak changes size from sample to sample). PCA can look "past" all the noise and correlation in the data set, and boil the entire data set down to essentials. Unfortunately, the principal components that are uncovered in the process don't correspond to anything concrete, usually. Again, you may wish to consult a more detailed treatment!

Table 1 gives an overview of the options available in ChemoSpec, and the relevant functions.

There's quite a bit of choice here; let's work through an example and illustrate, or at least mention, the options as we go. Keep in mind that it's up to you to decide how to analyze your data. Most people try various options, and follow the ones that lead to the most insight. But the decision is yours!

Example 7: Manual Removal of Samples Followed by HCA

```
> keep <- seq(1, 157, 6) # select every 6th spectrum
> someCIR <- CuticleIR # make a copy of the original data
> someCIR$names <- someCIR$names[keep] # modify each piece manually
> someCIR$colors <- someCIR$colors[keep]
> someCIR$groups <- someCIR$groups[keep]
> someCIR$sym <- someCIR$sym[keep]
> someCIR$alt.sym <- someCIR$alt.sym[keep]
> someCIR$data <- someCIR$data[keep,] # samples in rows, freq in columns
> chkSpectra(someCIR, confirm = TRUE) # make sure we didn't screw it up!
You must be awesome: These spectra look just dandy!
> hcaSpectra(someCIR, title = "Every 6th Sample of CuticleIR")
```

Every 6th Sample of CuticleIR: HCA Analysis
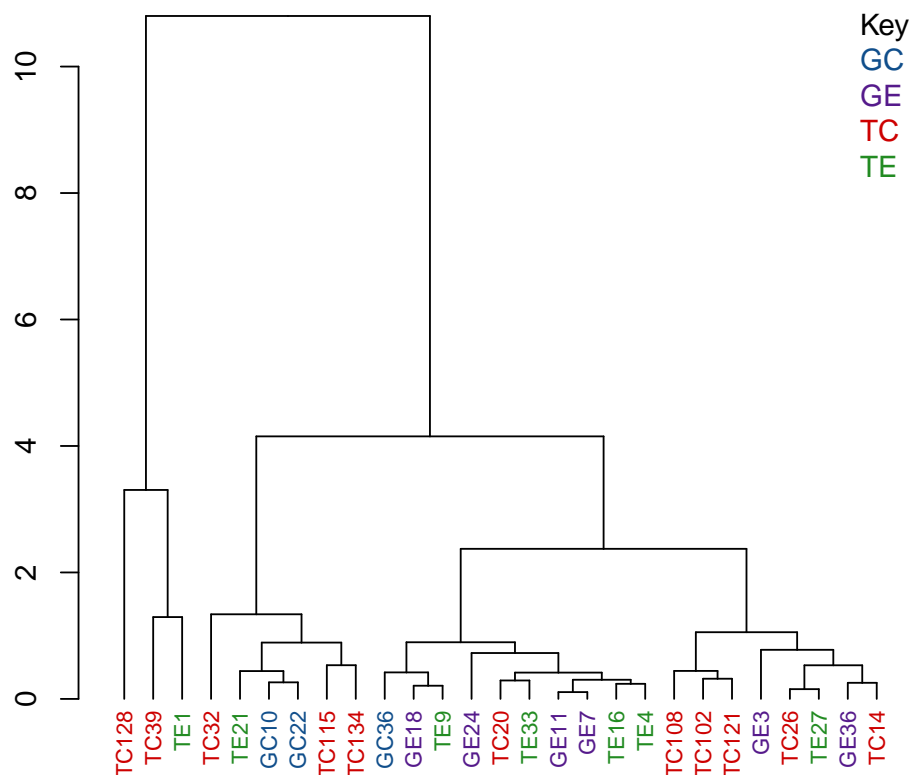
clustering method: complete   distance method: euclidean

Table 1: Principal Components Analysis Options & Functions

PCA options	scaling options	function
classical PCA	no scaling, autoscaling, Pareto scaling	`classPCA`
robust PCA	no scaling, median absolute deviation	`robPCA`
Diagnostics		
OD plots		`pcaDiag`
SD plots		`pcaDiag`
Choosing the correct no. of PCs		
scree plot		`plotScree`
bootstrap analysis (classical PCA only)		`pcaBoot`
Score plots	plotting options	
2D plots	robust or classical confidence ellipses	`plotScores`
3D plots		
—static 3D plots		`plotScores3D`
—interactive 3D plots		`plotScoresRGL`
—interactive multivariate plots		`plotScoresG`
Loading plots		
loadings vs frequencies		`plotLoadings`
loadings vs other loadings		`plot2Loadings`
s-plot (correlation vs covariance)		`sPlotSpectra`
Other		
HCA of PCA scores		`hcaScores`
ANOVA-PCA		`aovPCA`

The first step is to carry out the PCA. You have two main options, either classical methods, or robust methods. Classical methods use all the data you provide to compute the scores and loadings. Robust methods focus on the core or heart of the data, which means that some samples may be downweighted. This difference is important, and the results from the two methods may be quite different, depending upon your the nature of your data. The differences arise because PCA methods (both classical and robust) attempt to find the components that explain as much of the variance in the data set as possible. If you have a sample that is genuinely corrupted, for instance due to sample handling, its spectral profile may be very different from all other samples, and it can legitimately be called an outlier. In classical PCA, this one sample will contribute strongly to the variance of the entire data set, and the PCA scores will reflect that (it is sometimes said that scores and loadings follow the outliers). With robust PCA, samples with rather different characteristics do not have as great an influence, because robust measures of variance, such as the median absolute deviation, are used.

Note that as of `ChemoSpec` 1.46, neither `classPCA` nor `robPCA` carry out any normalization by samples. You need to decide if you want to normalize the samples, and if so, use `normSpectra`.

Besides choosing to use classical or robust methods, you also need to choose a scaling method. For classical PCA, your choices are no scaling, autoscaling, or Pareto scaling. In classical analysis, if you don't scale the data, large peaks contribute more strongly to the results. If you autoscale, then each peak contributes equally to the results (including noise "peaks"). Pareto scaling is a compromise between these two. For robust PCA, you can choose not to scale, or you can scale according to the median absolute deviation. Median absolute deviation is a means of downweighting more extreme peaks. The literature has plenty of recommendations about scaling options appropriate for the type of measurement (instrument) as well as the nature of the biological data set.[2, 4–7, 10]

There is not enough space here to illustrate all possible combinations of options; Example 8 and Example 9 show the use and results of classical and robust PCA without scaling, followed by plotting of the first two PCs (we'll discuss plotting options momentarily). You can see from these plots that the robust and classical methods have produced rather different results, not only in the overall appearance of the plots, but in the amount of variance explained by each PC.

Since we've plotted the scores to see the results, let's mention a few features of `plotScores` which produces a 2D plot of the results (we'll deal with 3D options later). Note that an annotation is provided in the upper left corner of the plot that describes the history of this analysis, so you don't lose track of what you are viewing. The `tol` argument controls what fraction of points are labeled with the sample name. This is a means of identifying potential outliers. The `ellipse` argument determines if and how the ellipses are drawn (the 95% confidence interval is used).

You can choose `"none"` for no ellipses, `"cls"` for classically computed confidence ellipses, `"rob"` for robustly computed ellipses, or `"both"` if you want to directly compare the two. Note that the use of classical and robust here has nothing to do with the PCA algorithm — it's the same idea however, but applied to the 2D array of scores produced by PCA. Points outside the ellipses are more likely candidates for outlier status.

Plots such as shown in Examples 8 and 9 can give you an idea of potential outliers, but `ChemoSpec` includes more sophisticated approaches. The function `pcaDiag` can produce two types of plots that can be helpful (Examples 10 and 11). The meaning and interpretation of these plots is discussed in more detail in Varmuza and Filzmoser, Chapter 3.[2] NOTE: There is some sort of problem with the OD plot as you can plainly see. I'll be working on it!

Depending upon your data, and your interpretation of the results, you may decide that some samples should be discarded, in which case you can use `removeSample` as previously described, then repeat the PCA analysis. The next step for most people is to determine the number of PCs needed to describe the data. This is usually done with a scree plot as shown in Example 12. In this case, it's clear that most of the variance can be described by one PC! We'll pursue that more in a bit.

If you are using classical PCA, you can also get a sense of the number of PCs needed via a bootstrap method, as shown in Example 13. Note that this method is iterative and takes a bit of time. Comparing these results to the scree plot, you'll see that the bootstrap method suggests that 2 PCs would not always be enough to reach the 95% level, while the scree plot suggests that 2 PCs is sufficient.

Now let's turn to viewing scores in 3D. There are currently 3 options in `ChemoSpec`: plotting using `lattice` graphics, which produces a static plot that you have to adjust manually, and two interactive plots, one based on package `rgl` and one based upon package `rggobi` which in turn uses the program `GGobi`. Probably the best place to start is with `plotScoresRGL`. It it well suited to exploring your data, and can be printed out in high quality. However, the nature of the GL graphics device means that the title and the legend move with the data, so this may not give a hardcopy suitable for publications. This interactive plot cannot be invoked in this document, but here are the necessary commands:

Example 8: How to Carry Out Classical PCA

```
> class <- classPCA(CuticleIR, choice = "noscale")
> plotScores(CuticleIR, title = "Cuticle IR Spectra", class,
+ pcs = c(1,2), ellipse = "rob", tol = 0.01)
```
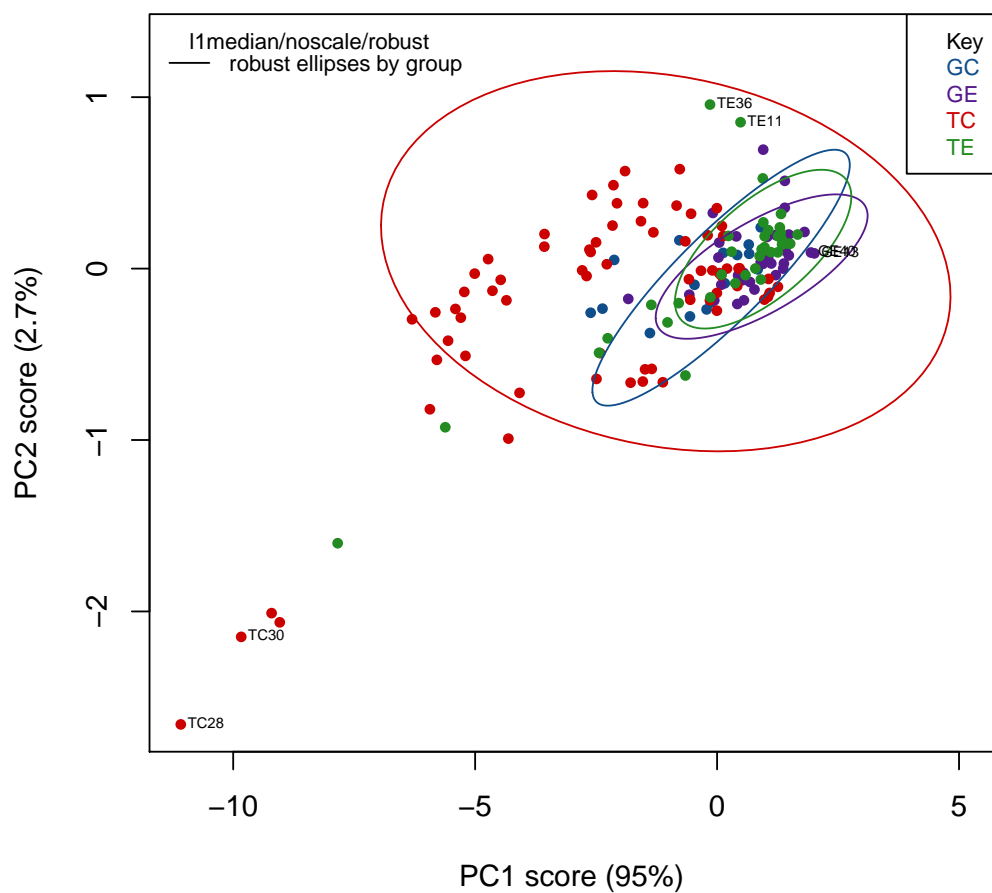
Cuticle IR Spectra: PCA Score Plot
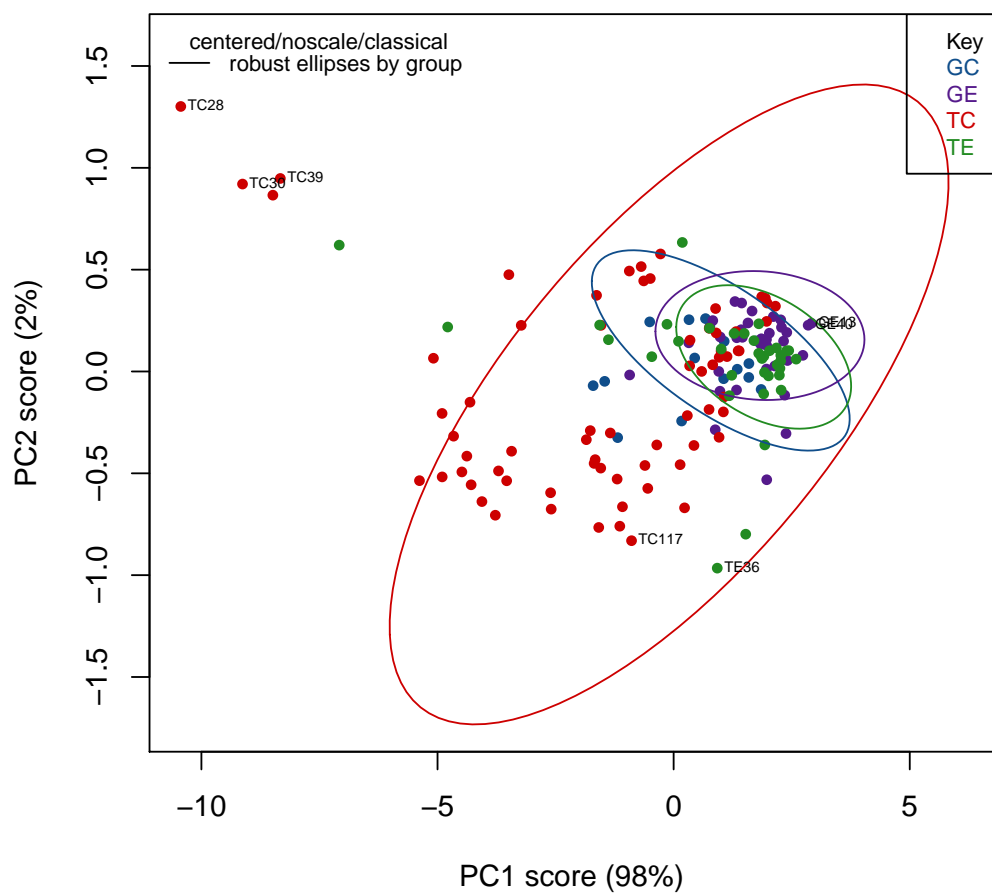
Example 9: How to Carry Out Robust PCA
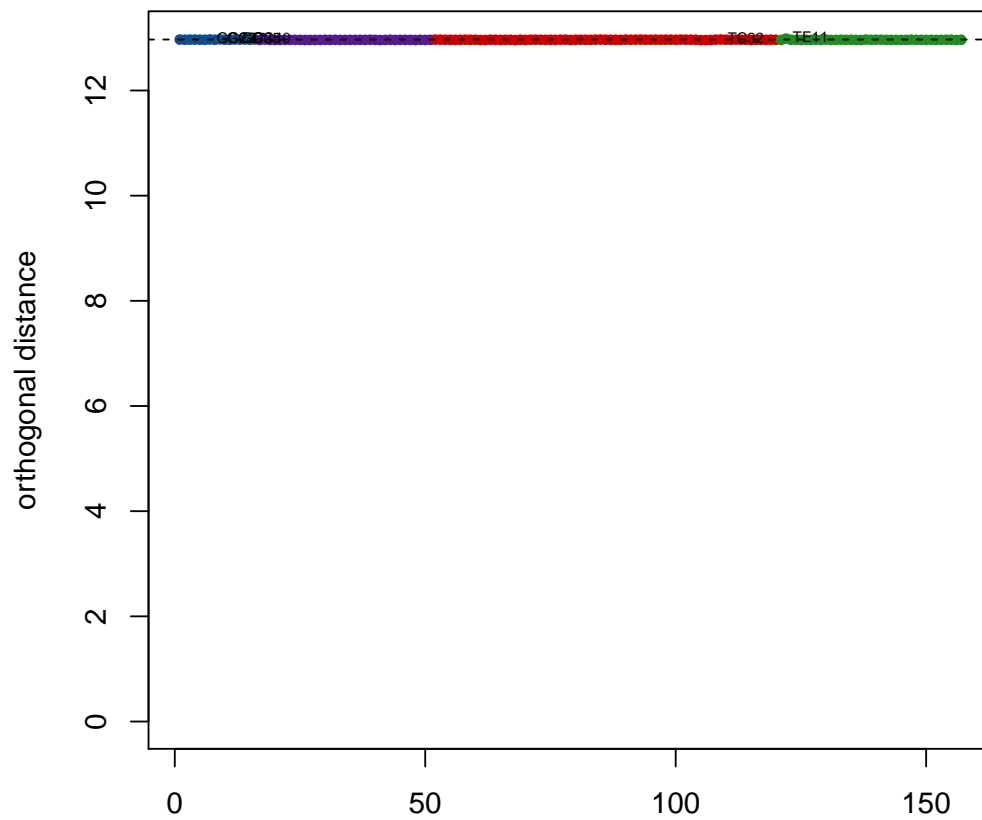
```
> robust <- robPCA(CuticleIR, choice = "noscale")
> plotScores(CuticleIR, title = "Cuticle IR Spectra", robust,
+ pcs = c(1,2), ellipse = "rob", tol = 0.01)
```

Cuticle IR Spectra: PCA Score Plot

## Example 10: Diagnostics: Orthogonal Distances

```
> diagnostics <- pcaDiag(CuticleIR, class, pcs = 2, plot = "OD")
```

Possible PCA Outliers based on Orthogonal Distance

Kelly's Complete IR Data Set, Summer 2009
centered/noscale/classical 2 PCs
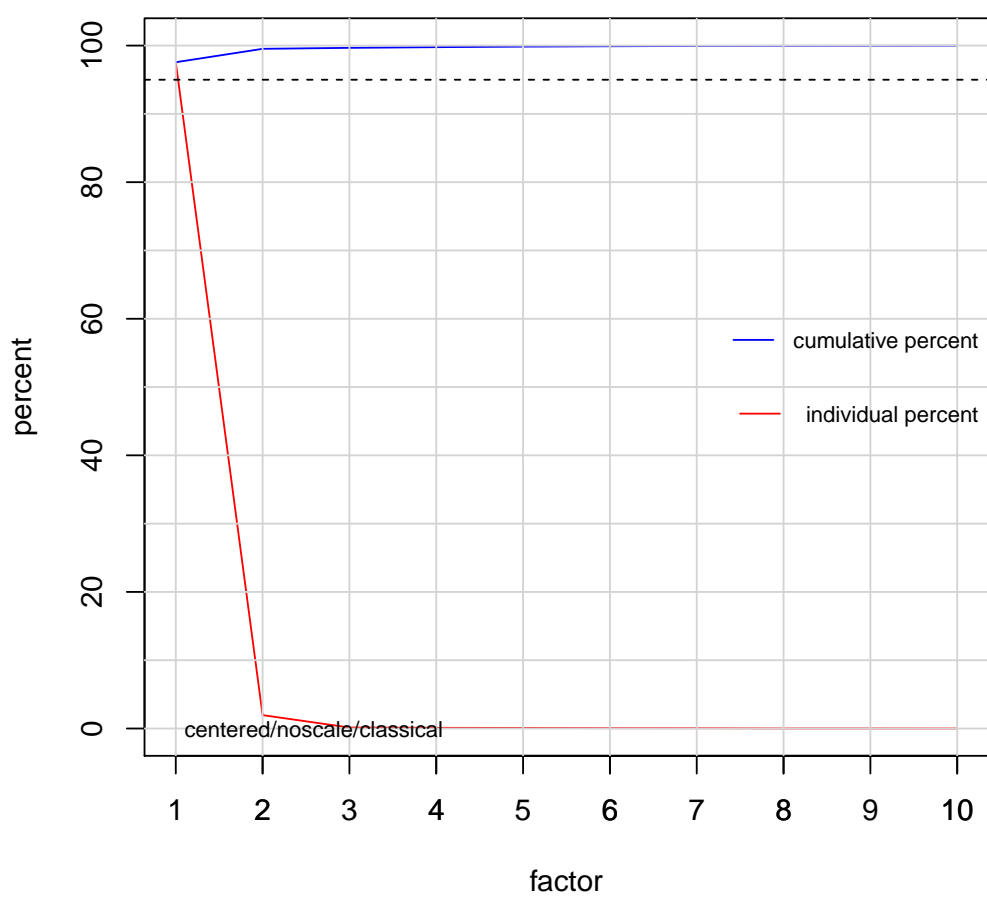
Example 11: Diagnostics: Score Distances

```
> diagnostics <- pcaDiag(CuticleIR, class, pcs = 2, plot = "SD")
```

Possible PCA Outliers based on Score Distance

Kelly's Complete IR Data Set, Summer 2009
centered/noscale/classical 2 PCs
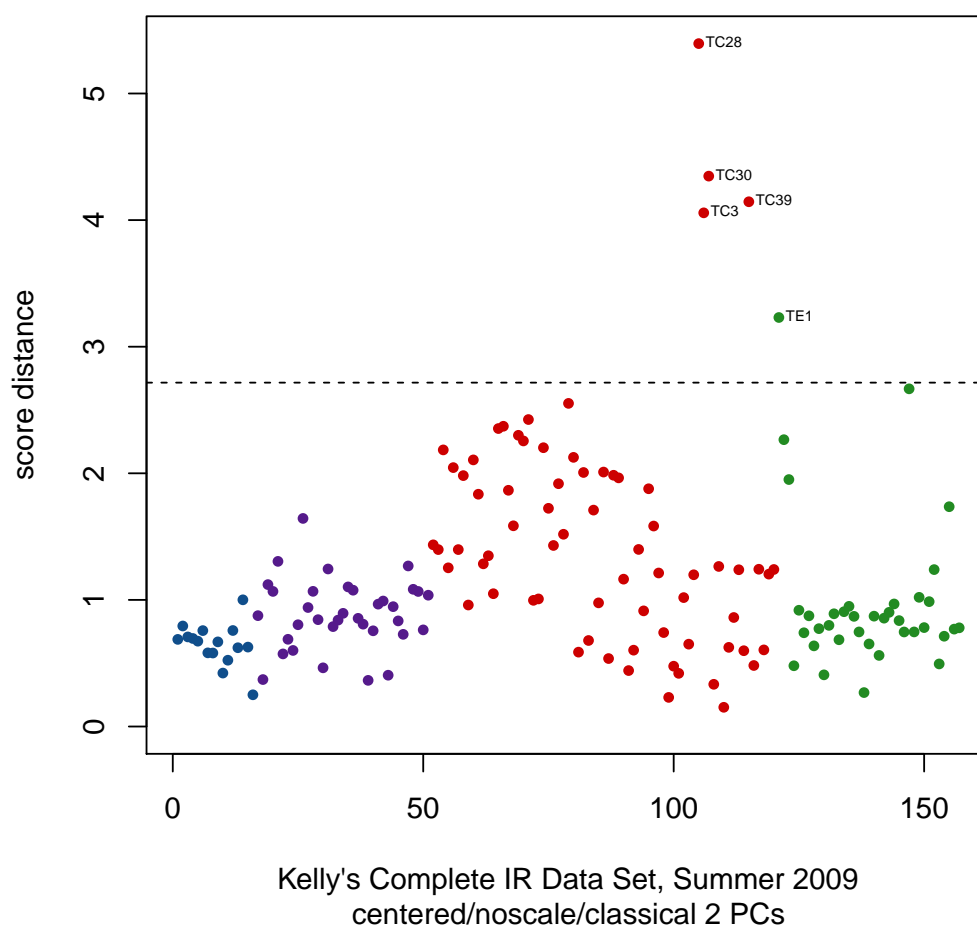
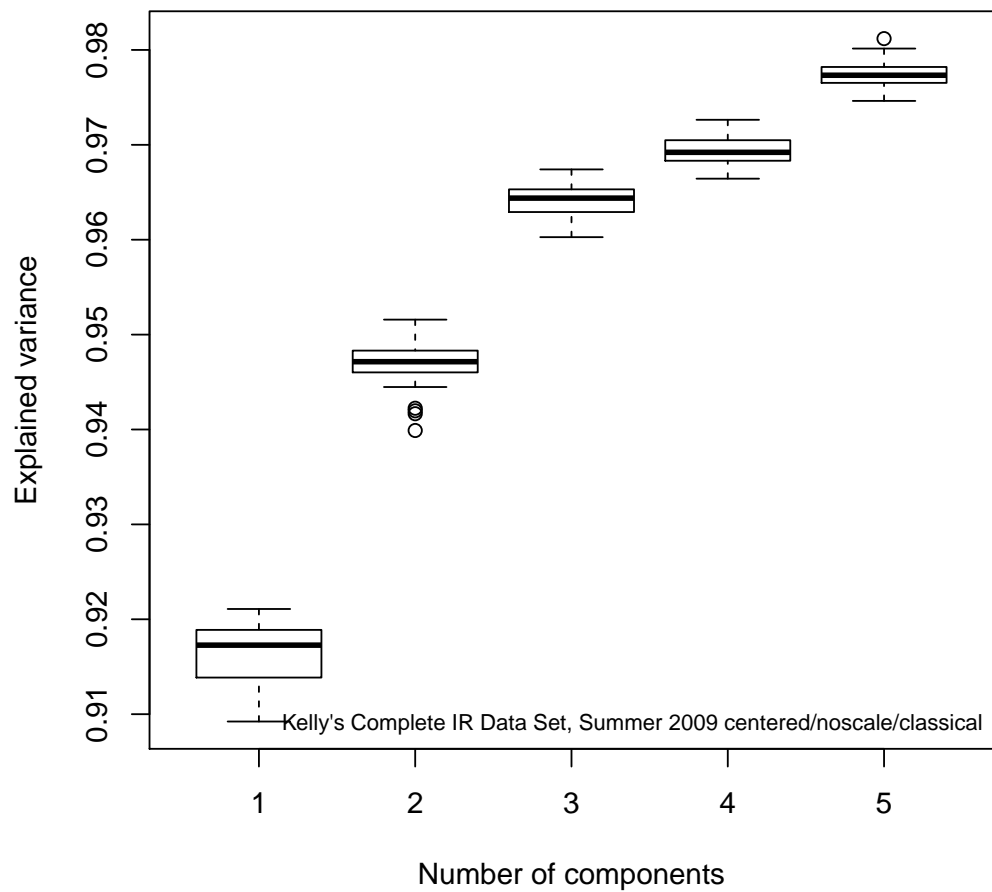**Example 12: Creating a Scree Plot**

```
> plotScree(class, title = "Cuticle IR Spectra")
```
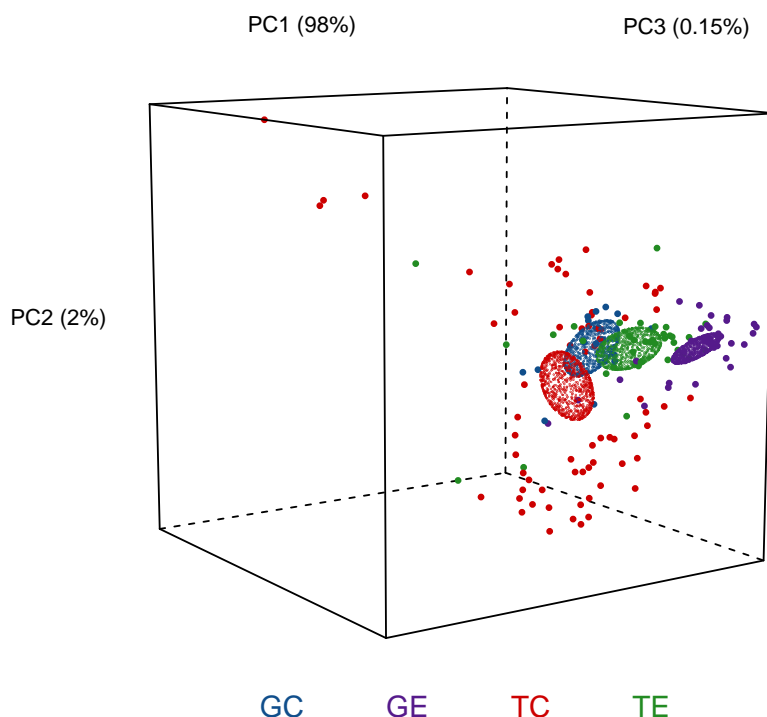
Cuticle IR Spectra: Scree Plot

Example 13: Conducting Bootstrap Analysis for No. of PCs

```
> out <- pcaBoot(CuticleIR, pcs = 5, choice = "noscale")
```

Bootstrap Analysis of Number of PCs

Kelly's Complete IR Data Set, Summer 2009 centered/noscale/classical

Number of components

Example 14: One Way to Plot Scores in 3D

```
> plotScores3D(CuticleIR, class, title = "Cuticle IR Spectra")
```

Cuticle IR Spectra: PCA Score Plot

PC1 (98%)   PC3 (0.15%)

PC2 (2%)

GC   GE   TC   TE

Kelly's Complete IR Data Set, Summer 2009

```
> plotScoresRGL(CuticleIR, class, title = "Cuticle IR Spectra",
+ leg.pos = "A", t.pos = "B") # not run - it's interactive!
```

For full details, of course take a look at the manual page, `?plotScoresRGL`. If you want a similar and probably more publication-worthy plot, you can use `plotScores3D` as shown in Example 14.

Finally, you can install the program `GGobi` and the package `rggobi` and use `plotScoresG` to produce an different interactive plot. This is actually the most powerful analysis tool, as `GGobi` is not restricted to 3 dimensions, and can use projection pursuit methods to find interesting views of your data. With an additional package, `DescribeDisplay`, you can create very nice plots of your data. Note that at this writing (October 10, 2011), `GGobi` has been described as on "life support" and it seems there might be a new program in development to replace it. However, you can find a working version at software.rc.fas.harvard.edu/mirrors/R/ (and related mirrors) thanks to Simon Urbanek. Note that the communication between `plotScoresG` and `GGobi` does not quite work as described earlier under the discussion of the color schemes; this will not be fixed until the future of `GGobi` is certain.

```
> plotScoresG(CuticleIR, class) # not run - it's interactive!
```

In addition to the scores, PCA also produces loadings which tell you how each variable (frequencies in spectral applications) affect the scores. Examining these loadings can be critical to interpreting your results. Example 15 gives an example. You can see that the hydroxyl peak above 3,000 cm^{-1} has a large effect on PC 1, and it is this peak that is responsible for

Example 15: Creating a Loading Plot

```
> plotLoadings(CuticleIR, class, title = "Cuticle IR Spectra",
+ loads = c(1, 2), ref = 1)
```

Cuticle IR Spectra: Loadings Plot
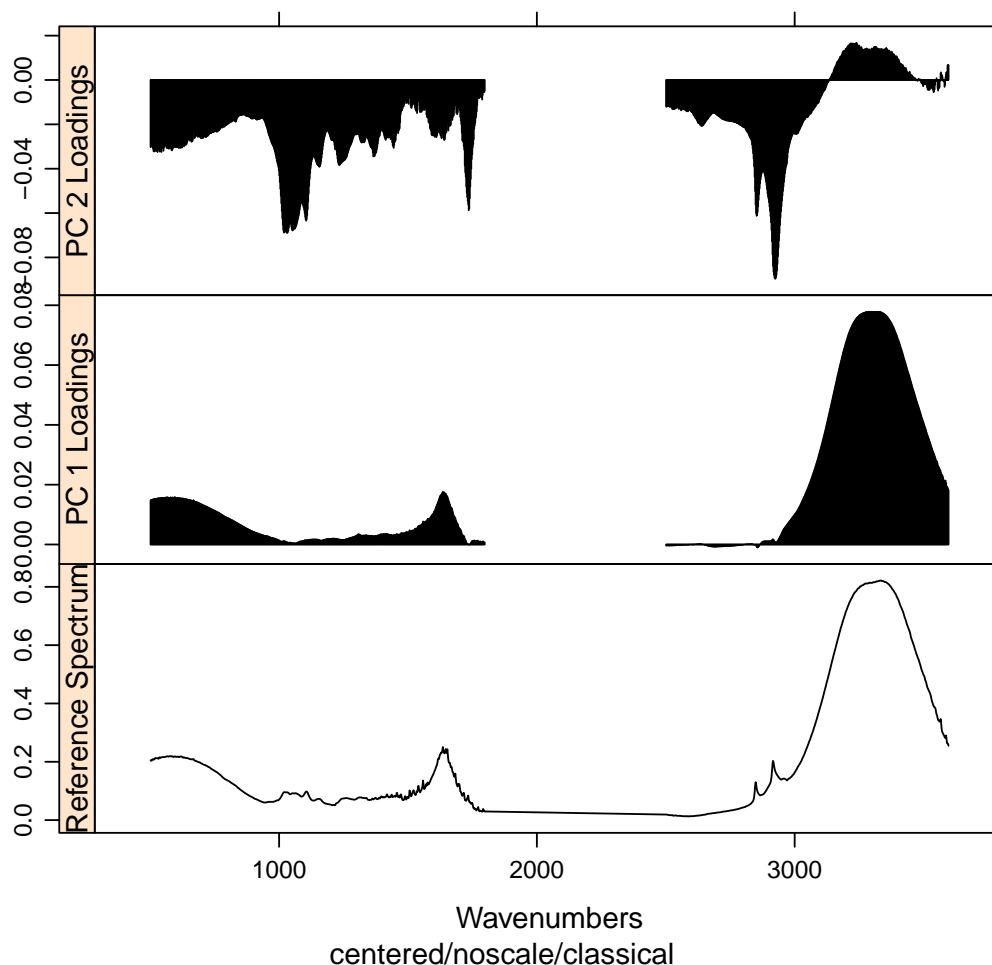
PC 1 explaining over 90% of the variance (mentioned earlier). It would be of interest to eliminate this peak, and repeat the PCA, but analysis of specific data is not our goal here.

You can also plot one loading against another, using function `plot2Loadings` (Example 16). This is typically not too useful for spectroscopic data, since many of the variables are correlated (as they are parts of the same peak, hence the serpentine lines in the figure). The most extreme points on the plot, however, can give you an idea of which peaks (frequencies) serve to differentiate a pair of PCs, and hence, drive your data clustering.

However, a potentially more useful approach is to use an s-plot to determine which variables have the greatest influence. A standard loadings plot (`plotLoadings`) shows you which frequency ranges contribute to which principal components, but the plot allows the vertical axis to be free. Unless you look at the y axis scale, you get the impression that the loadings for principal component 1 etc. all contribute equally. The function `sPlotSpectra` plots the correlation of each frequency variable with a particular score against the covariance of that frequency variable with the same score. The result is an s-shaped plot with the most influential frequency variables in the upper right hand quadrant. An example is shown in Example 17. This method was reported in Wiklund *et. al.*[11]

Finally, you can blend the ideas of PCA and HCA. Since PCA eliminates the noise in a data set (after you have selected the important PCs), you can carry out HCA on the PCA scores, since the scores represent the cleaned up data. The result using the `CuticleIR` data set are not much better than doing HCA on the raw spectra, so we won't illustrate it,
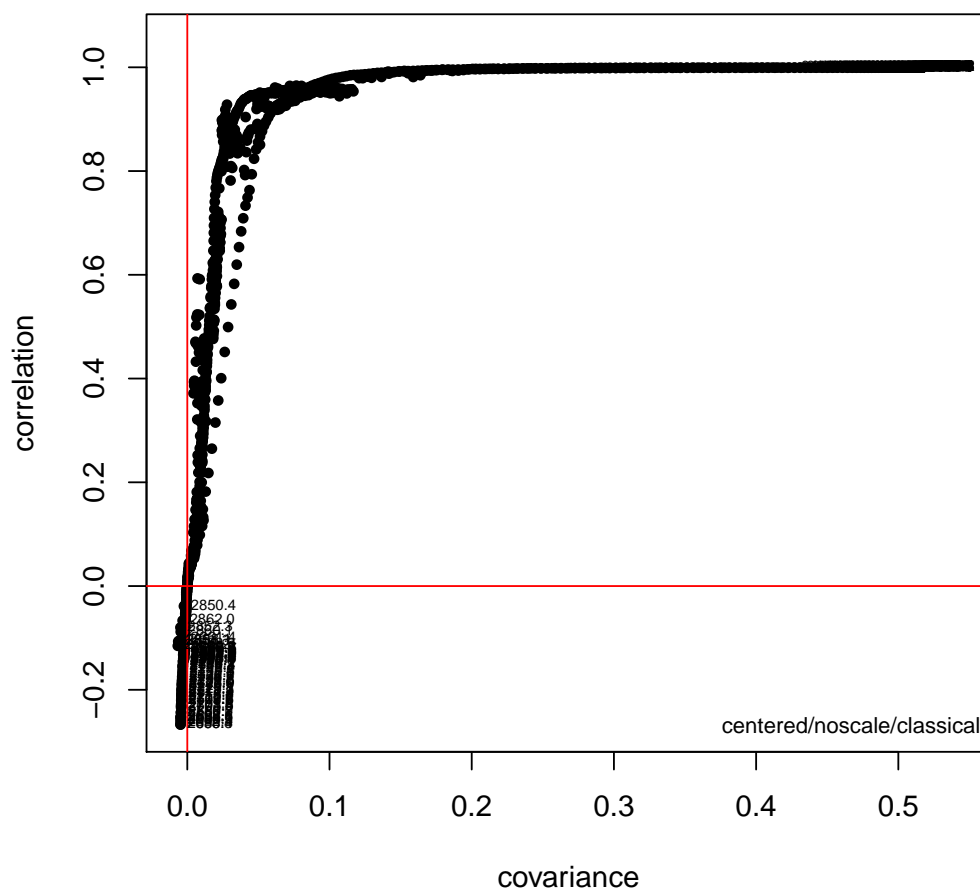
Example 16: Plotting One Loading vs. Another

```
> plot2Loadings(CuticleIR, class, title = "Cuticle IR Spectra",
+ loads = c(1, 2), tol = 0.002)
```
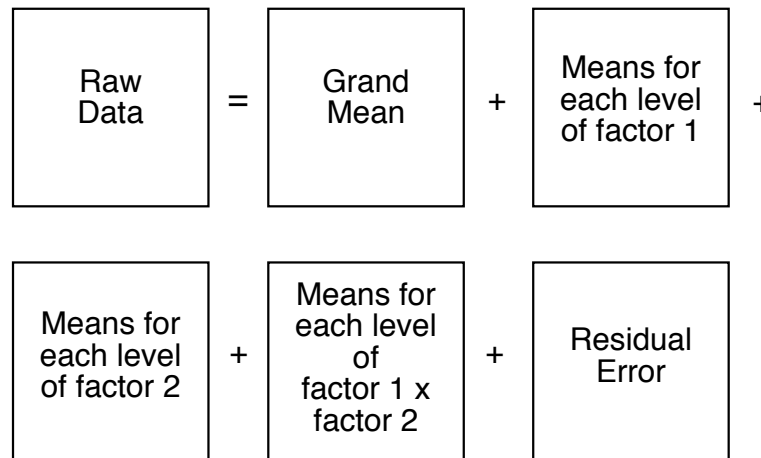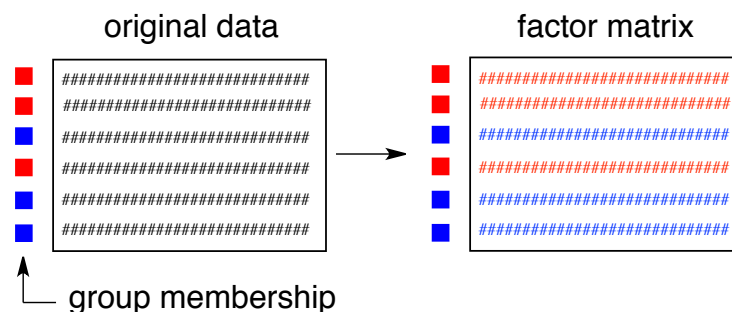
Cuticle IR Spectra: Covariance of Loadings

PC1 (98%) loadings
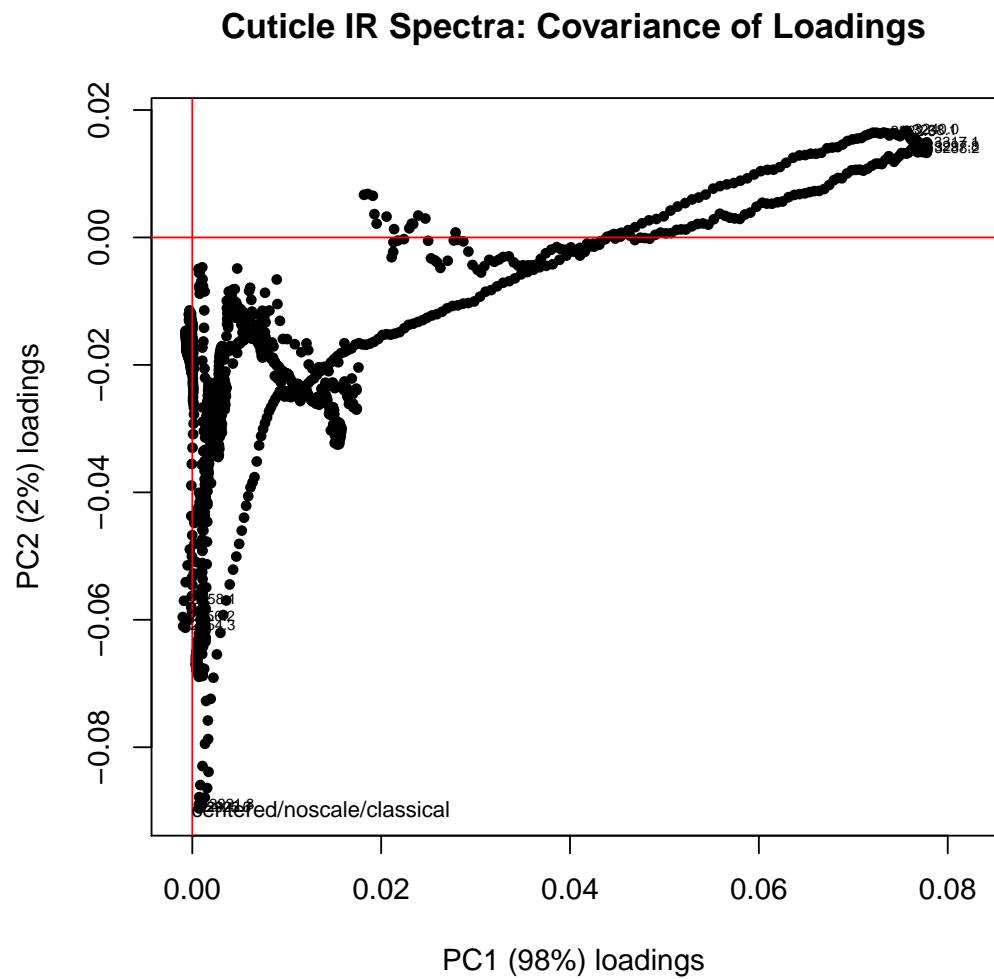
Example 17: s-Plot to Identify Influential Frequencies

```
> spt <- sPlotSpectra(CuticleIR, class, title = "Cuticle IR Spectra", pc = 1)
```

Cuticle IR Spectra: s–Plot

2

Example 18: aovPCA breaks the data into a series of submatrices

Example 19: Submatrices are composed of rows which are averages of each factor level

but the command would be:

```
> hcaScores(CuticleIR,  class, scores = c(1:5), title = "Cuticle IR Spectra",
+ method = "complete")
```

2.6   ANOVA-PCA

Harrington *et. al.*[12] (and a few others[13]) have demonstrated a method which combines traditional ANOVA with PCA. Standard PCA is blind to class membership, though one generally colors the points in a score plot using the known class membership. ANOVA-PCA uses the class membership to divide the original centered data matrix into submatrices. Each submatrix corresponds to a particular factor, and the rows of the submatrix have been replaced by the average spectrum of each level of the factor. The original data set is thought of as a sum of these submatrices plus residual error. The residual error is added back to each submatrix and then PCA is performed. This is conceptually illustrated in Examples **??** and 19.

ANOVA-PCA has been implemented in ChemoSpec via the functions aovPCA, aovPCAscores and aovPCAloadings. The method is shown in Example 20. The idea here is that if a factor is significant, there will be separation along PC1 in a plot of PC1 vs PC2. That's not the case in this example, by the way.

Example 20: aovPCA on the CuticleIR Data Set

```
> # Split the existing "groups" element of the Spectra object
> # into two new factors with two levels each
> n.groups <-list(genotype = c("G", "T"), treatment = c("C", "E"))
> NewIR <- splitSpectraGroups(CuticleIR, n.groups)
> # run aovPCA & plot the first score plot
> mats <-aovPCA(NewIR, fac = c("genotype", "treatment"))
> apca1 <- aovPCAscores(NewIR, mats, plot = 1, ellipse = "rob")
```


genotype: PCA Score Plot
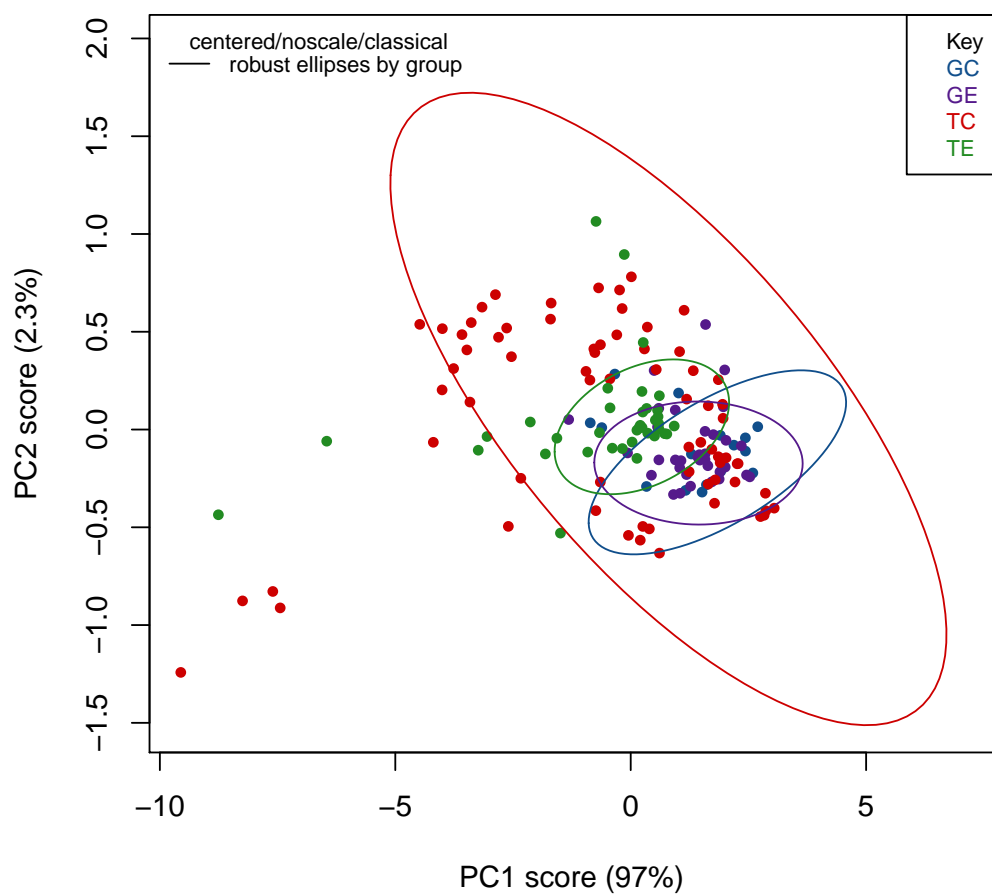
2.7   Model-Based Clustering Using mclust

PCA and HCA are techniques which are unsupervised and have no underlying model. HCA computes distances between pairs of spectra and agglomeratively groups these in an iterative fashion until the dendrogram is complete. PCA seeks out components that maximize the variance. While in PCA one often (and ChemoSpec does) displays the samples coded by their group membership, this information is not actually used in PCA; any apparent correspondence between the sample group classification and the clusters found is accidental in terms of the computation, but of course, this is what one hopes to find!

mclust is a model-based clustering package that takes a different approach.[14, 15]. mclust assumes that there are groups within your data set, and that those groups are multivariate normally distributed. Using an iterative approach, mclust samples various possible groupings within your data set, and uses a Bayesian Information Criterion (BIC) to determine which of the various groupings it finds best fits the data distribution. mclust looks for groups that follow certain constraints, for instance, one constraint is that all the groups found must have a spherical distribution of data points, while another allows for ellipsoidal distributions. See the paper by Fraley and Raftery[15] for more details. The basic idea however is that mclust goes looking for groups in your data set, and then you can compare the groupings it finds with the groupings you know to be true. ChemoSpec contains several functions that interface with and extend mclust functions. Let's demonstrate using a subset of the CuticleIR data set (we'll use only the T genotypes to make it easier to see what's going on, and we'll change the color scheme too):

```
> data(CuticleIR)
> G.only <- removeSample(CuticleIR, rem.sam = c("TE", "TC"))
> G.only <- conColScheme(G.only, new = c("red", "blue"))
> sumSpectra(G.only)

 Kelly's Complete IR Data Set, Summer 2009


        There are 51 spectra in this set.
        The y-axis unit is Absorbance.


        The frequency scale runs from 501.4261 to 3596.768 Wavenumbers
        There are 1242 frequency (x-axis) data points.
        The frequency resolution is 1.9286 Wavenumbers/point.


        This data set is not continuous along the frequency axis.
        Here are the data chunks:

   beg.freq end.freq     size beg.indx end.indx
1   501.4261 1797.420 1295.994        1      673
2 2501.3450 3596.768 1095.423      674     1242


        The spectra are divided into 2 groups:

  group no. color symbol alt.sym
1    GC  16   red      2       b
2    GE  35  blue     15       d


*** Note: this data is an S3 object of class 'Spectra'
```

Now let's do a PCA and plot the scores using plotScores. The code and results are seen in Example 21, where you can see that the two groups overlap slightly, and there are some widely-scattered points in each group. Next, we'll let mclust have a go at it. Remember that mclust doesn't know about your group assignments, it just looks at the data. In this case, mclust first uses the BIC to determine which model best fits your data; these results are shown in Example 22. Next, Example 23 shows the 3 groups that mclust finds in the data which actually is composed of 2 groups. Visually comparing the score plot in Example 21 with the mclust results in Example 23, you'll see that most of the rogue points have been collected into their own group, and the majority of the the true group members appear to form their own clusters. We need not guess however, as mclust will map the true groups onto the groups it has found. Points in error are X-ed out. These results can be seen in Example 24. From this plot, you can see that mclust is reasonably correct about

Example 21: PCA on a Subset of CuticleIR

```
> G.only.pca <- classPCA(G.only)
> plotScores(G.only, G.only.pca, title = "Two Genotypes (Controls)",
+ ellipse = "rob")
```
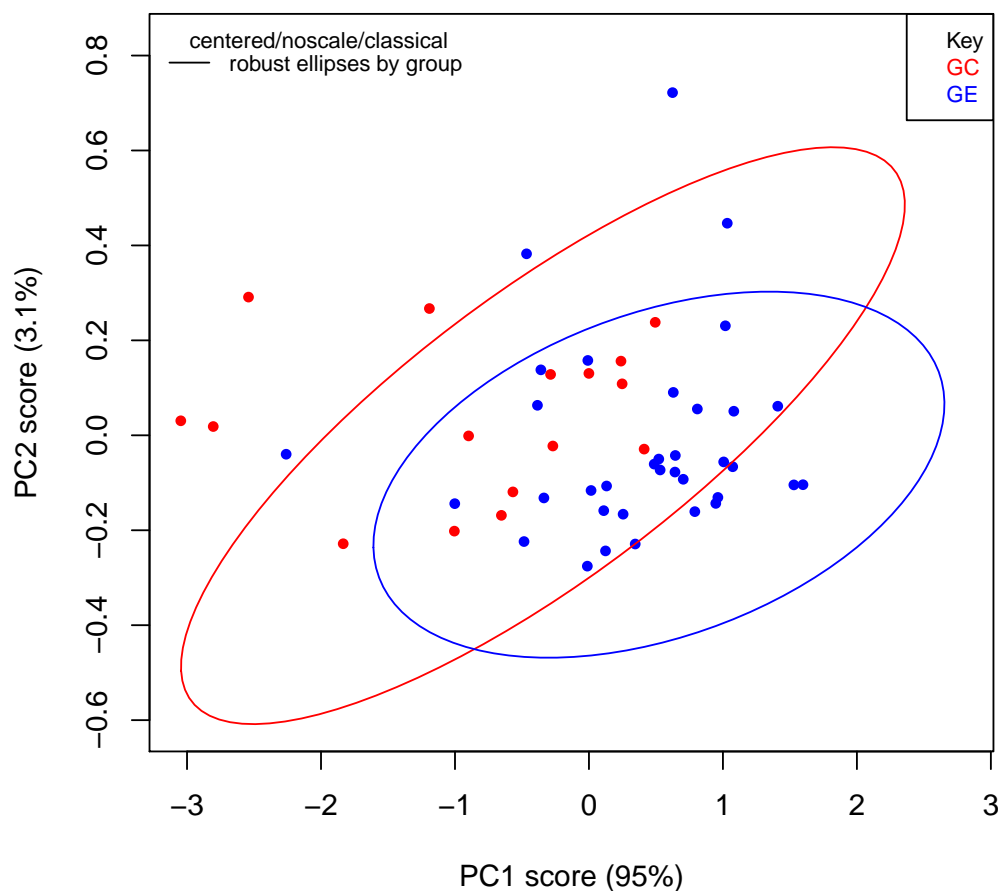

group membership; it appears to be taking a robust approach but in fact that is just an illusion in this case (sometimes outliers really drag the main body of good points around).

You can also do a similar analysis in 3D, using `mclust3dSpectra`. This function uses `mclust` to find the groups, but then uses non-`mclust` functions to draw confidence ellipses. This function uses `rgl` graphics so it cannot demonstrated here, but after doing the PCA as described in the examples, the commands would be:

```
> mclust3dSpectra(G.only, G.only.pca) # not run - it's interactive!
```

You have all the options here that you do with `plotScoresRGL`, namely, classical, robust or no ellipses, control of the ellipse details, and labeling of extreme points.
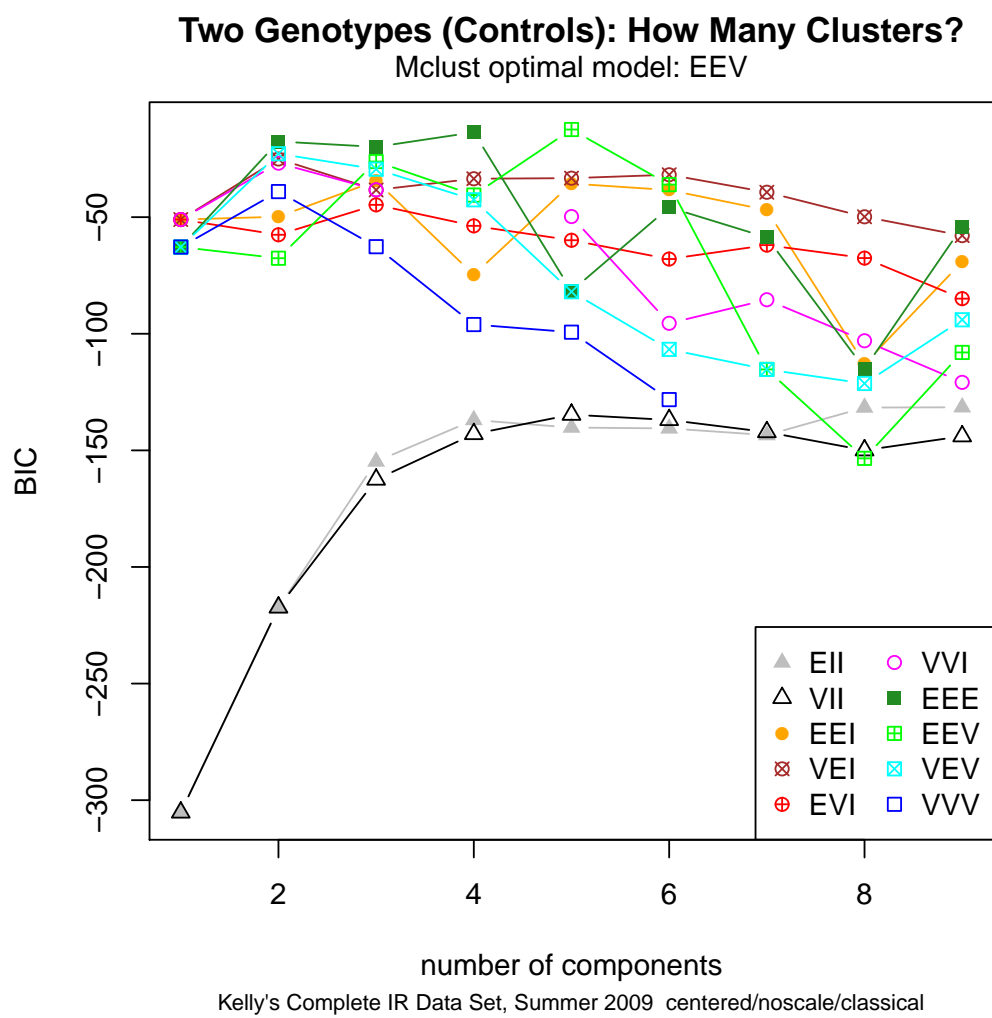
I hope you have enjoyed this tour of the features of `ChemoSpec`!

3 Functions That Are Not Discussed But Probably Should Be!

The help files of course do apply in the meantime. . .

Example 22: mclust Chooses an Optimal Model

```
> model <- mclustSpectra(G.only, G.only.pca, plot = "BIC",
+ title = "Two Genotypes (Controls)")
```

Two Genotypes (Controls): How Many Clusters?
Mclust optimal model: EEV

number of components
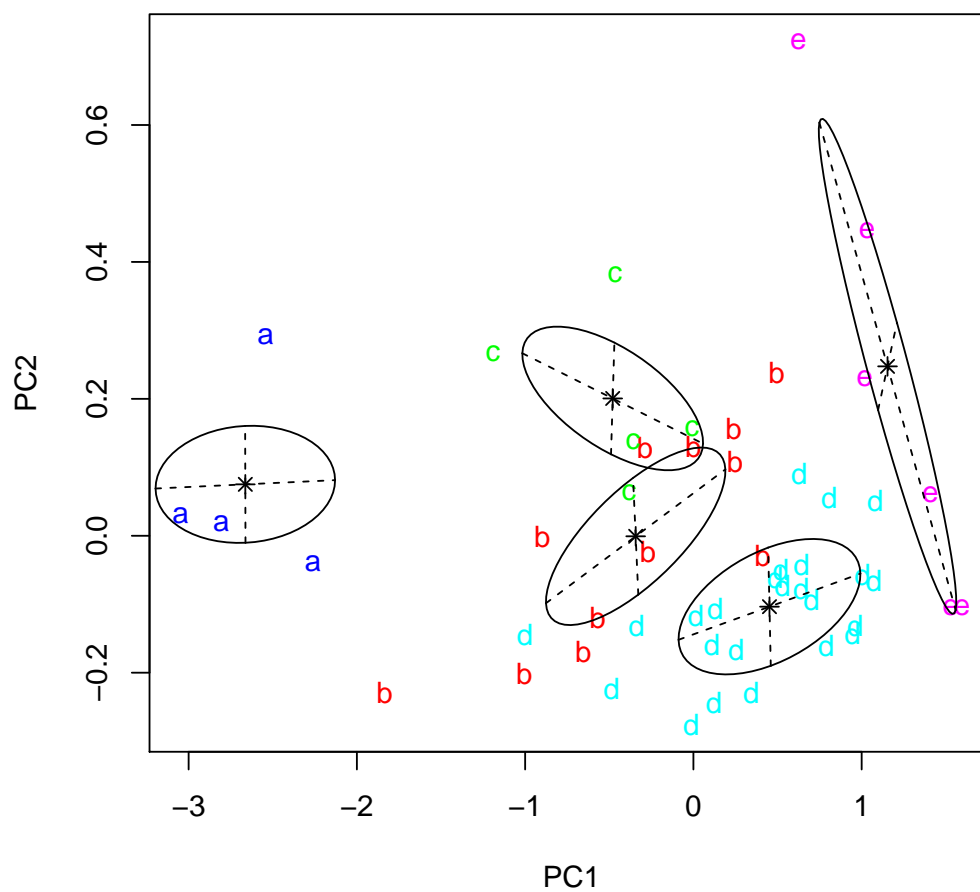Kelly's Complete IR Data Set, Summer 2009 centered/noscale/classical

Example 23: mclust's Thoughts on the Matter

```
> model <- mclustSpectra(G.only, G.only.pca, plot = "proj",
+ title = "Two Genotypes (Controls)")
```


Two Genotypes (Controls): Clusters Found by mclust
Mclust optimal model: EEV

Kelly's Complete IR Data Set, Summer 2009 centered/noscale/classical

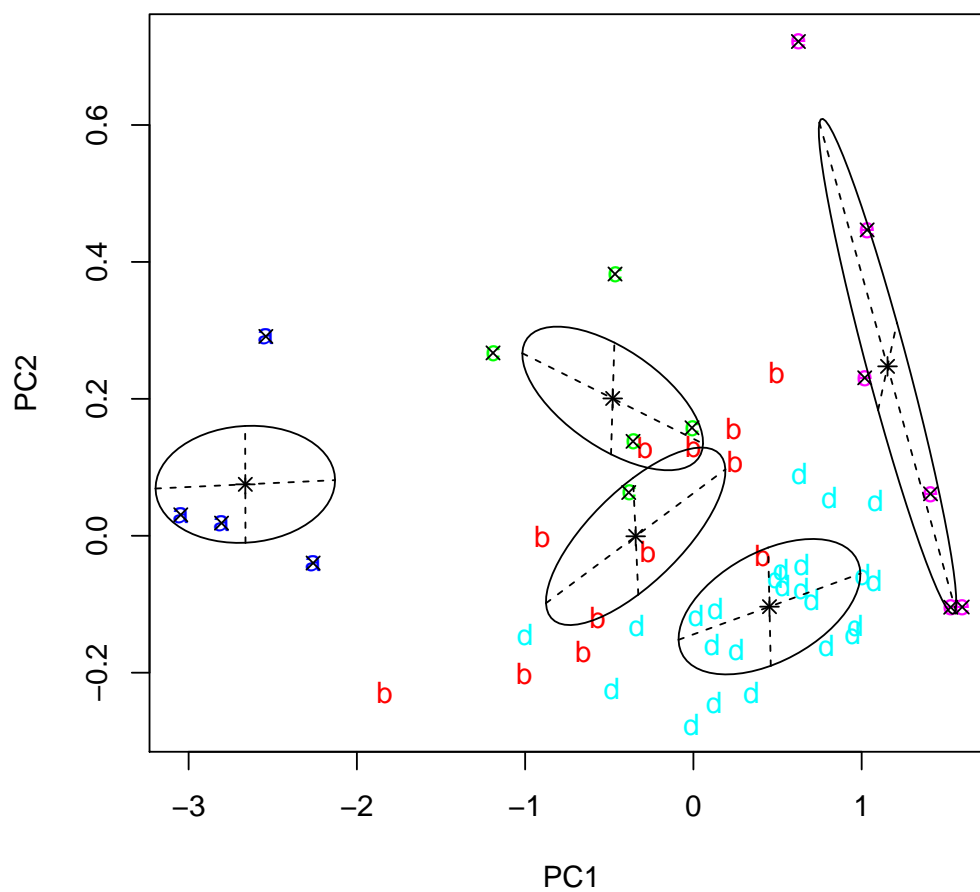**Example 24: Comparing mclust Results to the TRUTH**

```
> model <- mclustSpectra(G.only, G.only.pca, plot = "errors",
+ title = "Two Genotypes (Controls)", truth = G.only$groups)
```

Two Genotypes (Controls): Misclassifications?

Mclust optimal model: EEV

PC1

Kelly's Complete IR Data Set, Summer 2009 centered/noscale/classical

1. Demonstrate analysis of chromatograms

2. splitSpectraGroups

3. hypTestScores

4. hmapSpectra

4   Technical Background

`ChemoSpec` is written entirely in `R`, there is no compiled code. Hence, it should be platform independent (please let me know if you discover otherwise). `ChemoSpec` uses `S3` classes under the hood because frankly they were must faster to write. Converting to `S4` would not be terribly difficult. For the pros and cons of classes and object-oriented programming in `R`, see the help archives (search my name for one thread and some really interesting replies from the big dogs). `ChemoSpec` employs several different graphics packages - the choice was one of practicality. I find it pretty quick to write in base graphics, but `lattice` and `ggplot2` do certain things very well, so they were used where appropriate. Even `rgl` makes an appearance. In general, I tried to make all the graphics output look similar for consistency.

In understanding the operation of a package, it is useful to know how the functions relate to each other, i.e., which functions call each other. These relationships are readily visualized with a diagram like Example 25, generated by function `foodweb` in package `mvbutils`[16].

5   Acknowledgements

The development of `ChemoSpec` began while I was on sabbatical, and was aided greatly by an award of a Fisher Fellowship. These programs are coordinated by the Faculty Development Committee at DePauw, and I am very grateful to them as well as the individuals who originally created these programs. One of my student researchers, Kelly Summers, took the data included in `CuticleIR` in the summer of 2009 as part of a preliminary study. Prof. Dan Raftery at Purdue University provided an NMR data set (not included here) which was very helpful in troubleshooting functions. I am also grateful to Prof. Peter Filzmoser who answered a number of my questions related to the algorithms in his `chemometrics` package. Finally, Roberto Canteri of the Fundazione Bruno Kessler (Italy) brought some small bugs to my attention, and made some good suggestions for improving the underlying code. I am grateful to Roberto for assistance!

6   The Competition

Several other packages exist which do some of the same tasks as `ChemoSpec`, and do other things as well. `spectrino` is a GUI interface that runs only under the Windows OS[17]. It runs as a separate program in communication with `R`. It is oriented mainly toward processing and organizing spectral data prior to statistical analysis. `hyperspec` is a very nice package that appeared while I was developing `ChemoSpec`, and it does many of the same things as `ChemoSpec`, and a few more. It is written using `S4` classes which suggests that Claudia Beleites is a better computer scientist than me! `TIMP` is geared toward more sophisticated modeling of time-dependent spectral data sets.[18] Finally, the package `Metabonomic`[19] provides a GUI interface to spectral processing such as baseline correction, as well as a range of exploratory and supervised statistical methods. Note: my comments here are based on the latest versions I have explored; newer versions may have considerably more features. Check 'em out for yourself!

References

[1] B. A. Hanson, *ChemoSpec: Exploratory Chemometrics for Spectroscopy*, 2011. R package version 1.47-0.

[2] K. Varmuza and P. Filzmoser, *Introduction to Multivariate Statistical Analysis in Chemometrics*. CRC Press, 2009.

[3] D. S. Wishart, "Current progress in computational metabolomics," *Briefings in Bioinformatics*, vol. 8, no. 5, pp. 279–293, 2007.

Example 25: Map of Functions in ChemoSpec

```
> require(mvbutils)
> foodweb(where = "package:ChemoSpec", charlim = 30, cex = 0.75, lwd = 1)
```
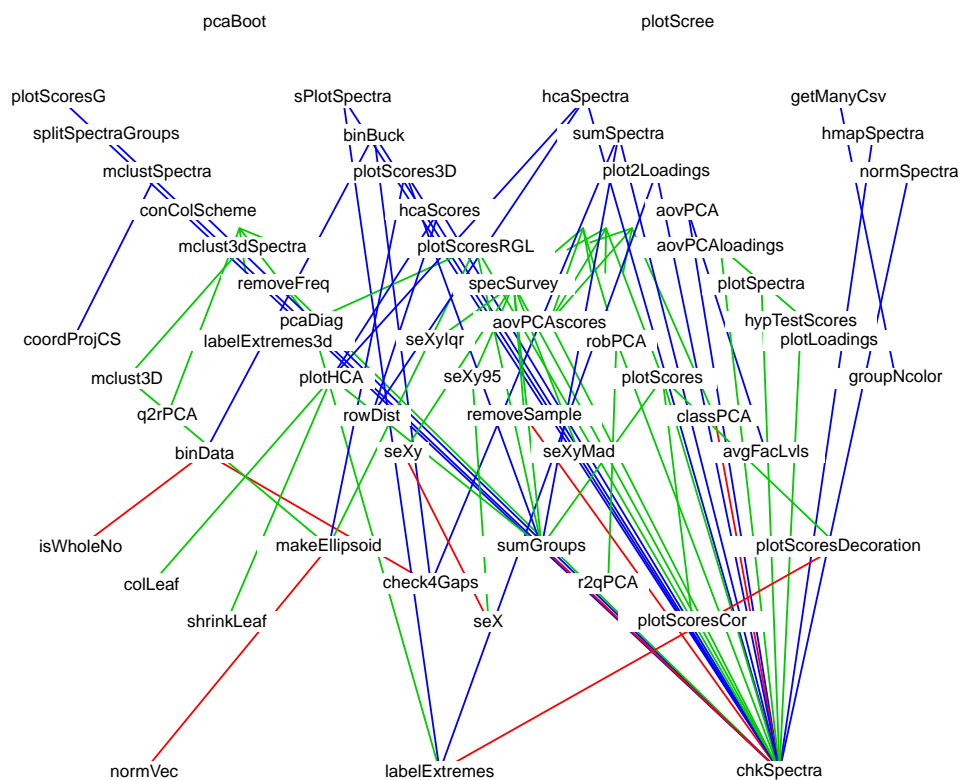
[4] A. Craig, O. Cloareo, E. Holmes, J. K. Nicholson, and J. C. Lindon, "Scaling and normalization effects in NMR spectroscopic metabonomic data sets," *Analytical Chemistry*, vol. 78, no. 7, pp. 2262–2267, 2006.

[5] R. Romano, M. T. Santini, and P. L. Indovina, "A time-domain algorithm for NMR spectral normalization," *Journal of Magnetic Resonance*, vol. 146, no. 1, pp. 89–99, 2000.

[6] R. A. van den Berg, H. C. J. Hoefsloot, J. A. Westerhuis, A. K. Smilde, and M. J. van der Werf, "Centering, scaling, and transformations: improving the biological information content of metabolomics data," *BMC Genomics*, vol. 7, p. 15, 2006.

[7] S. C. Zhang, C. Zheng, I. R. Lanza, K. S. Nair, D. Raftery, and O. Vitek, "Interdependence of signal processing and analysis of urine H-1 NMR spectra for metabolic profiling," *Analytical Chemistry*, vol. 81, no. 15, pp. 6080–6088, 2009.

[8] P. E. Anderson, N. V. Reo, N. J. DelRaso, T. E. Doom, and M. L. Raymer, "Gaussian binning: a new kernel-based method for processing NMR spectroscopic data for metabolomics," *Metabolomics*, vol. 4, no. 3, pp. 261–272, 2008.

[9] T. De Meyer, D. Sinnaeve, B. Van Gasse, E. Tsiporkova, E. R. Rietzschel, M. L. De Buyzere, T. C. Gillebert, S. Bekaert, J. C. Martins, and W. Van Criekinge, "NMR-based characterization of metabolic alterations in hypertension using an adaptive, intelligent binning algorithm," *Analytical Chemistry*, vol. 80, no. 10, pp. 3783–3790, 2008.

[10] T. K. Karakach, P. D. Wentzell, and J. A. Walter, "Characterization of the measurement error structure in 1D H-1 NMR data for metabolomics studies," *Analytica Chimica Acta*, vol. 636, no. 2, pp. 163–174, 2009.

[11] S. Wiklund, E. Johansson, L. Sjostrom, E. J. Mellerowicz, U. Edlund, J. P. Shockcor, J. Gottfries, T. Moritz, and J. Trygg, "Visualization of gc/tof-ms-based metabolomics data for identification of biochemically interesting compounds using opls class models," *Analytical Chemistry*, vol. 80, no. 1, pp. 115–122, 2008. PMID: 18027910.

[12] P. Harrington, N. Vieira, J. Espinoza, J. Nien, R. Romero, and A. Yergey, "Analysis of variance-principal component analysis: A soft tool for proteomic discovery," *ANALYTICA CHIMICA ACTA*, vol. 544, no. 1-2, pp. 118–127, 2005.

[13] R. C. Pinto, V. Bosc, H. Nocairi, A. S. Barros, and D. N. Rutledge, "Using ANOVA-PCA for discriminant analysis: Application to the study ofmid-infrared spectra of carraghenan gels as a function of concentration and temperature," *ANALYTICA CHIMICA ACTA*, vol. 629, no. 1-2, pp. 47–55, 2008.

[14] C. Fraley and A. Raftery, *mclust: Model-Based Clustering / Normal Mixture Modeling*, 2009. R package version 3.4.

[15] C. Fraley and A. E. Raftery, "Model-based clustering, discriminant analysis, and density estimation," *Journal of the American Statistical Association*, vol. 97, no. 458, pp. 611–631, 2002.

[16] M. V. Bravington, *mvbutils: Workspace organization, code and documentation editing, package prep and editing, etc.*, 2009. R package version 2.5.0.

[17] T. Krastev, "spectrino software: Spectra visualization and preparation for R," *Journal of Statistical Software*, vol. 18, no. 10, pp. 1–16, 2007.

[18] K. M. Mullen and I. H. M. van Stokkum, "TIMP: an R package for modeling multi-way spectroscopic measurements," *Journal of Statistical Software*, vol. 18, no. 3, 2007.

[19] J. L. Izquierdo, *Metabonomic: GUI for Metabonomic Analysis*, 2009. R package version 3.3.1.