Journal of Statistical Software

January 2007, Volume 18, Issue 10. http://www.jstatsoft.org/

spectrino Software: Spectra Visualization and Preparation for R

Teodor Krastev
SpecLabs

Abstract

spectrino is a spectra preparation software utility for the R language and environment for statistical computing. It is an operating-system specific tool, for use under Microsoft Windows, with specialized visualization, organization and preprocessing features for spectra. The software accepts spectral data from analytical instruments and then prepares a data structure to be introduced in R. **spectrino** has a rich set of features to create data structures and visually manipulate/compare spectra. The application is accessible by a library of functions from within R. These commands allow for the creation and manipulation of data structures in **spectrino** and the selective extraction of spectral data. Before exporting, the spectra are preprocessed according the requirements of consecutive discriminant analysis. This preprocessing is adjustable by a series of options.

Keywords: spectroscopy, spectrometry, analytical instruments, spectral data tank, visualization, R project, discriminant analysis.

1. Introduction

The increasing use of a variety of discriminant analysis (a.k.a. pattern recognition) techniques in spectroscopy and spectrometry is due to an increasing instrumental reproducibility, as well as to more widespread use of versatile statistical methods (PCA-MD, neural networks, etc.). The purpose of these discriminant techniques is to classify samples into well defined categories based on a training set of similar samples. A discriminant method can "learn" to recognize a spectrum to belong to a category by "training" it with spectra of the same group.

The popular open statistical software project R (R Development Core Team 2006) has hundreds of contributors from various fields of science and mathematics, and offers flexible packages for discriminant analysis: e.g., `prcomp` from **stats** (R Development Core Team 2006), **mda** (Hastie *et al.* 2006), **AMORE** (Castejón Limas *et al.* 2006), **mclust** (Fraley and Raftery

2006), etc. A fast-growing variety of analytical instruments produce large amounts of spectral data. It seems natural to have some universal intermediator, which imports the spectral data, organizes it in a structure appropriate for discriminant analysis, performs preprocessing and finally exports the spectra into R (or allows the data to be imported from within R). Our suggestion for such a universal intermediator is **spectrino**, with the addition of an ability for visual control and comparison of spectra. R–**spectrino** communication requires the **rcom** package. Both the **spectrino** and **rcom** packages are platform specific and can be run under Microsoft Windows operating systems only.

2. Program purpose

The **spectrino** application can be connected to R (using the **rcom** library) as an out-of-process COM server. A range of visual options and tools allows the user to visually control and compare spectra. The principal aim of the program is to organize spectra into a three-level structure for selective data extraction into R.

From the point of view of R, **spectrino** is a library of functions for creation/modification of spectral data structures and selective data extraction. The latter is preceded by tunable preprocessing, optimized for discriminant analysis. The main data-extraction functions can also be initiated from **spectrino**'s own menu.

A typical procedure involving **spectrino** would be:

- Exporting a set of spectra files in flat-text x-y format from the spectrometer software and placing them in one directory.
- Creating and filling in a spec-group data structure in **spectrino**. This process involves showing and comparing the loaded spectra, looking for obvious errors and adjusting preprocessing options. This is performed independent of R.
- Extraction of the spectral data from **spectrino**. This may be accomplished by including **spectrino** library functions in an R-script that also performs the discriminant analysis, or by extracting spec-groups in R variables (from R or from **spectrino**) and then using them in user R-script.

3. Graphical user interface

The graphical user interface (GUI) of the **spectrino** application is divided into three panels as shown in Figure 1. The application is a Windows executable and can be run without R. Its menu interface is described in detail in Appendix A.

3.1. Names panel and data organization

The main objective of **spectrino** is to give structured access to a collection of measured spectra, which we refer to as a *spectral data tank*. **spectrino** puts the data in a a three-level structure, facilitating discriminant analysis.

The lowest level of the structure (to the raw data) collects sets of data points into spectra. The primary purpose of **spectrino** is to work with mass-spectra. However, there is only one

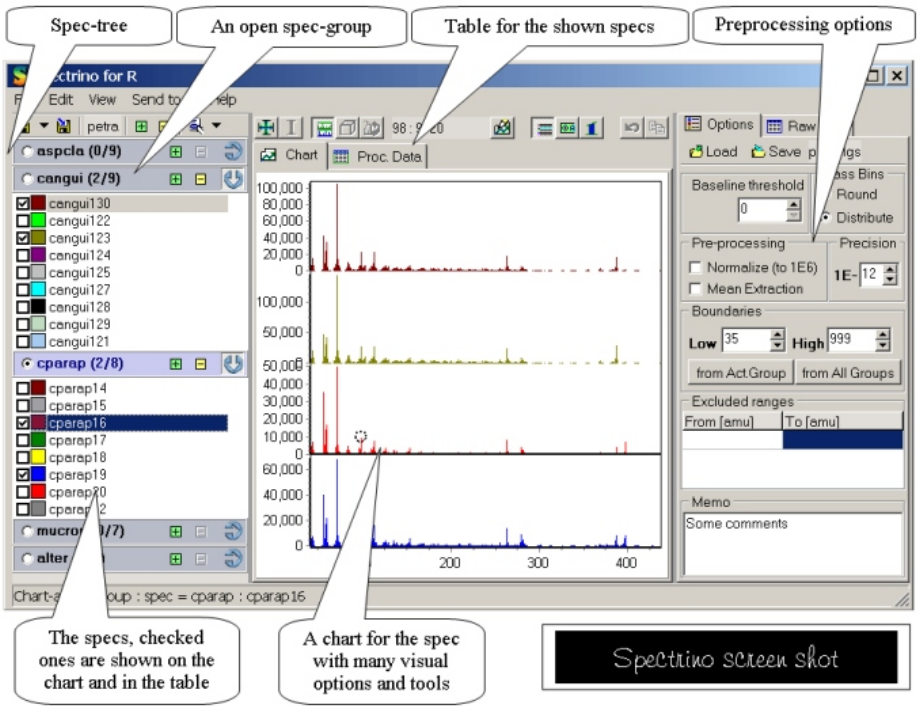

Figure 1: The graphical user interface of **spectrino**. From left to right are the names panel, the chart panel, and a panel for preprocessing options. Optionally, a communication panel can be shown on the left hand side.

relatively small restriction related to that - the reference X-scale uses integer numbers. The restriction can be overcome easily by converting the data, before input into **spectrino**, to appropriate (in regard to the instrument resolution) X-axis units of measurement (e.g. using Angstrom instead of nanometer for VIS spectra) or simply using natural channel numbers in case of a multi-channel detector. A spectrum can have a comment text attached to it (kept in the spec-group file).

In **spectrino**, a spectrum is either enabled or disabled, appearing in the spec-group list as checked/unchecked. Only enabled spectra are shown in the chart. All the commands for data extraction in R have an "OnlyChecked" booloean option which allows the user to extract only the enabled spectra. The state of the spectra can be changed from R as well. The main purpose of enabled/disabled state is to allow implementation of spectrum quality control - either visually or by calculations from within R. A secondary purpose can be to experiment with "what-if" exclusion of some spectra. This can be used to estimate how sensitive your algorithm to a random exclusion of spectra is.

The second level in the structured representation of data in **spectrino** is a group of spectra (or *spec-group*) which correspond to one classification group or category or class. Typically, spectra of one group represent measurements of the same material or type of materials. One of the groups is marked *active*. The active group is also identifiable in R, and can be selectively analyzed. Which group is active can be changed using the visual interface (in particular, using a radio button in the title bar) or via an R command.

The third level of data representation is a tree representing the whole list of all the groups

loaded, which we term a *spec-tree*. Typically, this is one statistical "experiment" in discriminant analysis. Only one spec-tree can be loaded at a time. The spec-tree file contains the preprocessing options applied; the user can choose to load only the list of groups, so that alternative preprocessing options may be applied.

The left panel of Figure 1 shows that it is visually easy to follow this three-tier structure. The remainder of this section is a manual devoted to the explanation of chart and preprocessing panels of Figure 1 in detail.

3.2. Chart panel

The main purpose of chart panel is visualization of selected (enabled) spectra for quality control and comparison.

Left mouse button can be used to zoom part of the chart and right one - to scroll (move) visible part of it.

Buttons

The following buttons are present on the chart panel.

- **Unzoom** button, undoes the zoom (mouse zoom backwards will do the same).
- **AutoY**, rescales the Y-axis when zooming or scrolling, so as to fit the maximum of the visible part of the spectra.
- **Stacked** view, shows the spectra in a separate y-axes (stacked) or overlays them with a common y-axis.
- **Pseudo 3D**, shows the spectra in 3D depth.
- **Reorder** spectra, rotates the order of the spectra so the last spectrum on the back becomes the first (in the front). It works only with a common y-axis.
- **All checked**, shows all checked spectra from all the groups.
- **Active group**, shows the checked spectra from the active group only.
- **One spectrum**, shows only the active spectrum (the selected one from the active group).
- **Update**, updates Proc. Data table. Because of the low update speed, the updating of the table is manual.
- **Copy**, copies the table into the clipboard, including the X-axis values (left column), but without the column titles (top row).

3.3. Preprocessing

The preprocessing is applied before the spectra are exported to R as array.

Described here are options for preprocessing.

Mass bin modes

Typically, the discriminant techniques require each measurement (or spectrum) to be presented as a set of values of established variables. These variables are usually the intensities of some natural type of channels (or bins). In mass-spectrometry these are the mass numbers. In any type of multi-channel detector (e.g. CCD in optical spectroscopy), one channel is one detector cell. Thus, in case a calibration is needed to attach an intensity peak to a channel (as with the mass numbers) and in order to keep the same set of variables for the whole spec-tree, all the spectra must have a common X-scale. Fist, **spectrino** creates the reference scale (as a set of bins) using all the integers inside the boundaries and without "excluded ranges". If the scale of a spectrum does not match the reference scale, that spectrum is recalculated according to the reference one, using one of the "Mass Bin" modes:

In the "Round" bin mode, every mass peak is accumulated into the closest integer bin (e.g. 1124.3 mass peak goes to 1124 bin; 1128.8 to 1129, etc). This mode is applicable when there are no big deviations from the calibration (mass positions). If the calibration error is inferior to half of the bin size, that method will produce a better correction of the calibration.

In the "Distribute" bin mode, the intensity of every peak is distributed to the bins next to it. The portion of the intensity complementary to 100 percents is accumulated in the closest bin and the rest of it, in the farther bin. For example, a peak at position 1124.4 will be distributed, as follows: 60 percents of its intensity to bin 1124 and 40 percents - to bin 1125. If the calibration is off more than half of a bin or the calibration error is unknown, this would be the method of choice.

In the example given in Figure 2, the spectrum that is second from the top has been simulated to be Gaussian-distributed. The top spectrum is a Gaussian with an artificial de-calibration (with added noise of 0.4 sigma). The third spectrum has been corrected with the "Round" method, and the bottom spectrum has been corrected with the "Distribute" method. As it is quite obvious from the chart, the "Distribute" method gives a better correction (similarity to the gauss).

In the case where all spectra have a natural integer X-scale (as from a multi-channel detector), either one of the modes would produce the same result.

Low and high boundaries

The boundaries are required to limit the X-scale of exported spectra. These values can be set manually or calculated using the two buttons ("from Act.Group" and "from All Groups") from the active group or from the whole spec-tree. The calculated value for the low boundary is equal to the highest (rightmost) from the left boundaries of the spectra. Likewise, the high boundary is the lowest (leftmost) of the right boundaries.

Excluded ranges

Exclusion of some parts of spectra may be desirable. **spectrino** applies all the ranges given in the range list. Some reasons to exclude parts of the spectra could be:

- Recognizable contamination/buffer of the samples - if the contamination is presented only in some parts of the spectrum, these parts can be excluded.
- If for some reason parts of the detector channels are defective or work poorly.

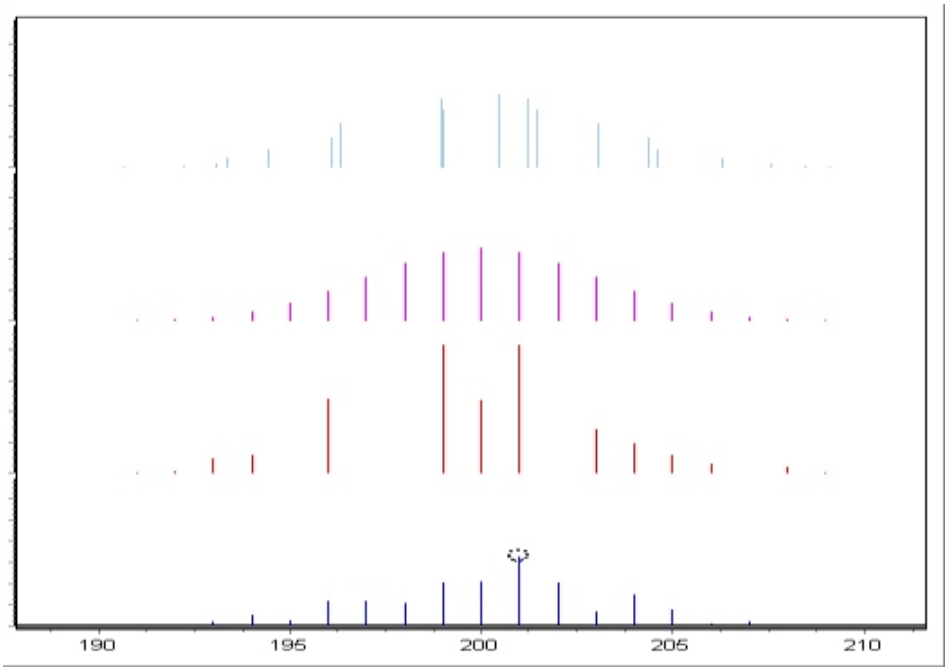

Figure 2: Mass bin modes comparison. The spectrum second from the top is Gaussian-distributed. The top spectrum is a Gaussian with an artificial de-calibration (with added noise of 0.4 sigma). The third spectrum has been corrected with the "Round" method, and the bottom spectrum has been corrected with the "Distribute" method.

- If in some spectra certain spectral lines are distinctively saturated, but the user still wishes to use these spectra.

Baseline correction

If "Baseline correction" is on, each intensity in the spectra is decreased by a baseline threshold value and then the negative values are replaced with zeros.

Normalization

The intensities of the peaks are multiplied by such a coefficient that the total of them is equal to 1,000,000. This option is most commonly used to compensate for the different overall intensities of the spectra (usually due to the varying long-term sensitivity of the instrument, or sample mass variation).

Mean extraction

The mean extraction option calculates the base-group's mean spectrum (average of all the spectra in the base group) and extracts this mean from every spectrum processed.

This option is applicable only when variation analysis is used (e.g. PCA). The purpose of the operation is to rescale the intensities, so that the predominant magnitude is the variation. This should increase the precision of any further calculations without changing the recognizable pattern.

Precision

This option defines the machine zero. It is used in calculations (e.g. to stop sequence of iterations).

Although the primary aim of **spectrino** is to be used with qualitative (discriminant) analysis, it is also usable for quantitative analysis. In the case, the concentrations should be encoded in the spec/group names.

4. Calling spectrino from R

Besides its visualization features, **spectrino** can be used independent of its graphical user interface as a spectra storage, organizer and preprocessing utility.

The concept of the commands is to reproduce the functionality of the GUI using only the R environment installed under Microsoft Windows. The user may start learning **spectrino** from its GUI and later move to performing the same analysis via the R commands in the list below. The commands can be classified as open/save file operations, getting/setting item state, reading names, counting items, deleting item and data extraction. This classification is valid on spec-tree level, where the items are the groups and on the group level where the items are spectra. The general commands are initializing and quitting **spectrino**; and reading/changing options.

4.1. Variables used in command definitions

The following **Variables** are used in the commands definitions:

- `Grp,Spc`: string or integer variable. String corresponds to the group/spec name, and integer to the index of it (by default - 0). Index 0 for `Grp` points to the active group and `Spc` index 0 is the selected spectrum. The names of groups or spectra are their filenames without path or extension (no space or special characters allowed). To avoid misinterpretation the names of group or spectrum cannot be numbers (even Windows lets you do it). In some cases `"*"` masks all the items.
- `GrpIdx, SpcIdx`: integer variable; symbol `"*"` masks all the items.
- `OnlyChecked, Checked, InclOpt, Visible`: logical variables; (`TRUE`/`FALSE`); the value by default is `FALSE`
- `*Filename`: string; if full path filename is required, forward slashes must be used e.g. *D:/Prime/Data/Test.txt*
- On the lest side of definitions - `i` is an integer, `b` is a boolean, `s` is a string, `v` is a vector, `m` is a matrix.

4.2. R commands

Initialization of **spectrino**

Check if R-object of **spectrino** exists, and if not, creates **spectrino** object in R, runs and initializes **spectrino** application.

The command is recommendable, but optional - it would be called, when any other command is executed and R-object of **spectrino** does not exist. The function returns `FALSE`, if failed.
The syntax of this command is

```
b <- spnNew()
```

Retrieve number of groups loaded

The command to retrieve the number of groups loaded is

```
i <- spnGetGrpCount()
```

Retrieve number of spectra in group

The command to retrieve the number of spectra in a group is

```
i <- spnGetSpcCount(OnlyChecked,Grp)
```

Some examples are

```
i1 <- spnGetSpcCount(TRUE,2) # the number of the checked spectra in second group
i1 <- spnGetSpcCount(FALSE,0) # the number of all spectra in the active group
```

Get the group name

The command to retrieve the name associated with a group having index `GrpIdx` is as follows.

```
s <- spnGetGrpName(GrpIdx)
```

If `GrpIdx="*"`, this command returns a comma-separated list of all groups.

Get the spectrum name

The command to retrieve the name associated with a spectrum having index `GrpIdx` is as follows.

```
s <- spnGetSpcName(Grp,SpcIdx)
```

If `SpcIdx="*"`, this command returns a comma-separated list of all spectra in the group.

Get reference X set of values

All the spectra in a spec-tree are assumed to have common X set of values, so in case of a spectrum in different X values, X sets are recalculated according to the preprocessing options. The command to return the vector of X values is as follows.

```
v <- spnGetRefer()
```

Return a spectrum

To obtain only the intensities (Y-values) of `Spc` spectrum from `Grp` group, similar to `spnGetGrp` and `spnGetTree`, a command as follows may be used.

```
v <- spnGetSpc(Grp,Spc)
```

Some examples are

```
v1 <- spnGetSpc(2,3) # or in an equivalent form
v1 <- spnGetSpc("waer","sea12") # where "waer" is the second group,
                                # and "sea12" - the third spectrum in it.
spnGetSpc(2,"*") # is equivalent to spnGetGrp(FALSE,2)
```

Get spectra from one group (matrix)

This retrieves spectra in **m** array from `Grp` group. Variables are by columns; measurements are by rows, for more convenient use with `prcomp` (principal component analysis) function.

```
m <- spnGetGrp(OnlyChecked,Grp)
```

Some examples are

```
m1 <- spnGetGrp(FALSE,3) # creates matrix with all the spectra in
spnGetGrp(TRUE,"*") # is equivalent to spnGetTree(TRUE)
```

Get spectra from all the groups (matrix)

This retrieves spectra in **m** array from the entire spec-tree, excluding the spectra from a group called "unknowns". That special name is supposed to be for testing purposes only, so the data from that group are not included in all-data-get command. The command is as follows

```
m <- spnGetTree(OnlyChecked)
```

Get the logical vector of the state of spectra

The command of getting the logical vector of the state of spectra is as follows

```
lv <- spnGetSpcChecked(Grp)
```

Set `Spc` *spectrum of* `Grp` *spec-group checkbox*

If `Spc="*"` then all of spectra in `Grp` group are set. If `Grp="*"` then all of spectra in all groups are set to Checked. The command is as follows

```
spnSetSpcChecked(Grp,Spc,Checked)
```

Some examples are

```
spnSetSpcChecked(2,5,TRUE) # check just one;
spnSetSpcChecked(2,"*",FALSE) # all the spectrum from second group to OFF;
spnSetSpcChecked("*","*",TRUE) # all the spectra in all groups to ON
```

Open `SFilename` *in* `Grp` *group*

In this command to open a `SFilename` in `Grp` group, the path of `SFilename` is ignored and the path of `Grp` group is used. So the spectrum file must be in group file directory.

The function returns the number of the spectra in that group after the adding, as `spnGetSpcCount(false,Grp` The command is as follows

```
i <- spnOpenSpc(Grp,SFilename)
```

An example is

```
i1 <- spnOpenSpc(2,"sassul") # loads in second group the file "sassul.txt"
```

Open a group from `GFilename` *file*

If `NewGrp` is true, then GFilename must not exist to be created. If `NewGrp` is false, then GFilename must exist to be opened. If directory is omitted, the spec-tree directory is assumed.

The function returns the number of groups after the adding, which is the index of added group, as `spnGetGrpCount()`. The command is as follows

```
i <- spnOpenGrp(GFilename,NewGrp)
```

An example is

```
s1 <- spnOpenGrp("gassul",FALSE) # loads a group from the file "gassul.gsp"
                                 # from the current spec-tree direcory.
```

Open a spec-tree from `TFilename`

`InclOpt` rules where the preprocessing options will be taken from.

If `InclOpt = 0` then no preprocessing will be is used; if `InclOpt = 1`, the last used options are applied; if `InclOpt = 2` the options are taken from `TFilename`.

The spec-tree file (*.str) always is saved with the preprocessing options. The function returns the number of groups in the new spec-tree. `i=spnGetGrpCount`. The command is

```
i <- spnOpenTree(TFilename,InclOpt)
```

Create a new spec-tree

To create new spec-tree you have to empty current one by `spnDelGrp("*")` and save the empty one under new name.

Save `Grp` *group as* `GFilename` *file*

The path of `GFilename` is ignored, because any group file must be in the same directory as the spectra in it. If `GFilename` is empty, then **spectrino** uses the proper name of the group (value by default). In that case the function returns the filename under which the file has been saved.

If `Grp="*"` then all groups are saved under their proper names (in that case `GFilename` is ignored, but some string must be present) and nothing gets back to R. The command is

```
s <- spnSaveGrp(Grp,GFilename)
```

Some examples are

```
spnSaveGrp(2,"") # save second group under its own name;
spnSaveGrp(2,"gassew") # rename second group to "gassew" and then save it;
spnSaveGrp("*","") # save all the groups under their names;
```

Save the spec-tree along with the preprocessing options

If `TFilename` is empty then **spectrino** uses the proper name of the spec-tree (value by default). The list of the groups contains the full filenames of the groups, except if all the groups and `TFilename` are in the same directory. In that case the directory part of group filenames is omitted and which allows easy transfer of the whole experiment to another location (with no directory reference inside). The function returns the filename under which the file has been saved. The command is

```
s <- spnSaveTree(TFilename)
```

Some examples are

```
spnSaveTree("") # save the spec-tree under its name
spnSaveTree("savenow") # rename the spec-tree and save it
```

Delete `Spc` *spectrum from* `Grp` *group*

If `Spc="*"` then delete all of the spectra in that group them. The function returns number of the spectra in that group after the deleting, as `spnGetSpcCount(false,Grp)`. The command is

```
i <- spnDelSpc(Grp,Spc)
```

Some examples are

```
spnDelSpc(3,"*") # delete all the spectra in third group
spnDelSpc(0,3) # delete third spectra in the active group
```

Delete `Grp` *group*

If `Grp="*"` then delete all of the spec-groups. The function returns number of groups after the deleting, as `spnGetGrpCount()` The command is

```
i <- spnDelGrp(Grp)
```

Some examples are

```
spnDelGrp(3) # delete third group from the tree
spcDelGrp("*") # empty the entire spec-tree
```

Set **spectrino** *to be visible or hidden*

If `Visible = TRUE`, **spectrino** is shown; else **spectrino** is hidden. This command gets back the current visibility. The command is

```
b <- spnSetVis(Visible)
```

Get/set active group

If `Grp=0` this command only gets the Active group index; otherwise this command sets one. The command is

```
i <- spnActGrp(Grp)
```

Set preprocessing option(s) and returns the full list of options

This command contains a comma separated list of options as they are in a spec-tree file -> Preprocess section. The correspondence to the visual preprocessing options should be obvious by their names. The command is

```
s <- spnSetPPOpt(opt)
```

The list `opt` can contain the following items

- Normalize=0/1
- MeanExtract=0/1
- LowLimit=<integer>
- HighLimit=<integer>
- Precision=<integer 1..10>
- Baseline=<integer>
- BaselineOn=0/1
- MassBins=0/1
- BaseGrp=<GroupName>

An example is

```
spnSetPPOpt("Normalize=1,MeanExtract=1,LowLimit=60")
```

spectrino *validation*

This command tests only the most common functions and modes. It should be used once, after the **spectrino** installation or an update. If the result from the function on the R console is "Validation confirmed", there is a very good chance that **spectrino** will work. Otherwise - the error message with a number appears. In the latter case, the user should contact http://www.spectrino.com/. The command is

```
spnValidation()
```

Quitting **spectrino**

This command frees R-objects created by **spectrino** and closes the application of **spectrino**. This is the proper way to close **spectrino**. Its call is

```
spnFree()
```

4.3. An example of the application of spectrino from R

The following example illustrates the use of **spectrino** from R. To execute the example, create the subdirectory *data* under the *(R-directory)/library/spectrino/exec* directory, and then unpack and copy there all the files from *example.zip*. The data are ms-spectra of some organic compounds.

```
# loading spectrino; NB rcom library is required
library("spectrino")

# opens spectrino application and creates spectrino object representing it
spnNew()

# opens spec-tree from "data" sub-directory of spectrino executable directory
spnOpenTree("%SpnPath%data/sptree.str",2)

# loading another spec-group
spnOpenGrp("%SpnPath%data/cam.sgr",FALSE)

# loading two more spectra
spnOpenSpc("cam","%SpnPath%data/cam03.txt")
spnOpenSpcn("cam","%SpnPath%data/cam12.txt")

# loading a matrix of all the specs into mr variable
mr <- spnGetTree(FALSE)

# summary of principal components analysis
```

```
summary(prcomp(mr))

# follow the grouping (classes) of the points,
# each of which represents a ms-spectrum
biplot(prcomp(mr))
```

5. Conclusion

The software product **spectrino** presented here is designed to organize spectroscopy data into a structure suitable for further use in discriminant analysis packages in R. Along with accomplishing this main purpose, the program offers rich and adjustable preprocessing option as well as options for the visual control and comparison of spectra. The interest (1.5 downloads per day only from spectrino.com) in **spectrino** is due to its position as an intermediator between instrumental spectrometry and statistics, as well as its versatility in spectra visualization.

The further development of **spectrino** will be in three directions. The first direction is towards the support of other spectra in addition to mass spectra, e.g. optical spectra, along with new visualization features and a wider set of preprocessing options. The second direction of development is to make **spectrino** more convenient for quantitative analysis, e.g., to allow transformation of a group list into a table with customized columns. The third direction of development is the implementation of customizable plug-ins, to allow calling R functions specialized for spectra processing from **spectrino**. A plug-in will offer the user parameterization of an R function inside **spectrino**, then execute the function in R and finally show the result in **spectrino**.

References

Castejón Limas M, Ordieres Meré JB, Vergara González EP, Martínez de Pisón Ascacibar FJ, Pernía Espinoza AV, Alba Elías F (2006). **AMORE***: A MORE Flexible Neural Network Package*. R package version 0.2-9, URL http://wiki.R-project.org/rwiki/doku.php?id=packages:cran:amore.

Fraley C, Raftery AE (2006). "Some Applications of Model-based Clustering in Chemistry." *R News*, **6**(3), 17–23.

Hastie T, Tibshirani R, Leisch F, Hornik K, Ripley BD (2006). **mda***: Mixture and Flexible Discriminant Analysis*. R package version 0.3-2.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

A. The menu of the spectrino graphical user interface

This appendix describes the menu of the **spectrino** GUI that operates as an executable under the Microsoft Windows operating system.

File - Create, open and save spec-tree or spec-group

- **New Spec-Tree**...
- **Open Spec-Tree with Options**...
- **Open Spec-Tree Only**...
- **Save Spec-tree As**...
- **Save Spec-tree incl. Groups**...
- **Open/New Spec-Group**...
- **Save Act. Spec-Group As**...
- **Add Spec to Act. Group**...
- **Exit**

Edit

- **Copy Image** Copies the image of the chart in the clipboard as a metafile image.
- **Generate Test Tree** Generates 9 spectra in three groups in one new tree for testing purposes.
- **Autosave On Close** (option) Saves the list of groups and every group in it when **spectrino** closes.

View

- **COM Script Panel** Shows the communication (**spectrino**<=>R) panel. A log of all **spectrino**-R communications is kept in the COM Script Panel; from there, the user can initiate all the **spectrino** functions available from R.

Send to R

Each of these menu commands requires a variable name to which the vector or matrix will be assigned in R. The matrices' indexes are: samples are by rows - first index; variables are by columns - second index (making it ready for "prcomp" - principal component analysis).

- **Send Whole Spec-Tree**... Sends a matrix of all the spectra from all the spec-groups in the current spec-tree.
- **Send All Checked Spectra from Tree**... Sends a matrix of all the checked spectra from all the spec-groups in the current spec-tree. If there is a group with "unknowns" name, that group is excluded.
- **Send Whole Act. Spec-Group**... Sends a matrix of all the spectra from the active spec-group.
- **Send Act Group Checked Spec**... Sends a matrix of all the checked spectra from the active spec-group.

- **Send Active Spec**... Sends a vector of the selected spectrum from the active spec-group.

Help

- **Spectrino Help**... Gives the table of contents of the help
- **Check for updates**
- **About**... Shows **spectrino** version number, copyright and website link http://www.spectrino.com/

Affiliation:

Teodor Krastev
SpecLabs
2385 Marin St, Brossard
J4Y 1K7, Quebec, Canada
E-mail: spectrino@sicyon.com
URL: http://www.spectrino.com/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 18, Issue 10 *Submitted:* 2006-10-01
January 2007 *Accepted:* 2007-01-10