Diplomarbeit

zum Thema

Multivariate, Interactive Visualization in R

ausgeführt am
Institut für Statistik und Wahrscheinlichkeitstheorie
der Technischen Universität Wien

unter der Anleitung von
O.Univ.-Prof. Dipl.-Ing. Dr.techn. Rudolf Dutter

durch

Gschwandtner Moritz
Matrikelnr.: 0125439
Wenhartgasse 34/21, 1210 Wien

Wien, am 15. 3. 2009 Moritz Gschwandtner

Abstract

This thesis introduces two programs called "Spin & Brush" and "Multivariate Graphics", the purpose of which is the visualization of multivariate data. Since the programs are included in an R package called `mvgraph` , the thesis also deals with the issue of building R packages which interface C/C++ code. To make the installation of `mvgraph` as easy as possible, there is a section about it at the end of the thesis.

Contents

List of Figures

Conventions

This thesis contains not only many commands that can be executed on a command line interface, but also output generated by these commands as well as contents of files. To increase the readability we followed the following conventions:

Filenames, commands and package names which appear in the text are printed in `verbatim` style. Important commands which we wanted to emphasize are displayed in a green border:

```
This is an important command.
```

While `R` commands always start with a `>` (like on the `R` command line interface), commands that have to be executed on a Windows/Linux terminal always start with a `$` (like on the Linux `bash` shell).

Output created when executing commands is always printed in a red border:

```
This is computer output.
```

The contents of files are printed in a blue border:

```
This is the contents of a file.
```

Chapter 1

Introduction

Wide fields of statistics deal with visualization of multivariate data. Primary objective is the exploration of data structures as a first step in a statistical analysis process. The next steps often involve statistical methods which have to be chosen carefully, since different methods may lead to different results. Visual exploration does not only give a first summary of the data but may often be a useful help in making the decision which statistical methods to use.

This thesis introduces two R programs called "Spin & Brush" and "Multivariate Graphics". The first is a three-dimensional data viewing program which offers the possibility to rotate and zoom. Data points can be clustered by using different colors and are identified by name or selection. The second was made for datasets which include geographical coordinates. Data points are drawn as "multivariate icons" on a geographical background map (the locations of the icons are specified by their coordinates) which makes it possible to show the values of many variables at a time. Several multivariate icons such as stars, trees or castles are provided. A more detailed description of the programs can be found in the following sections.

Since the programs are written in C/C++ and are included in an R package called `mvgraph`, we will also give a short introduction about how to build an R package which interfaces C/C++ code. The main part of "Spin & Brush" is written in OpenGL, an API for highly optimized graphical applications. A section about OpenGL is also included in the thesis. Furthermore both programs use some Qt (see http://trolltech.com/) libraries which makes the installation of the package more difficult than an installation of a package which interfaces standard C code. Therefore we include a section dealing with installation details.

Furthermore there is a "Frequently Asked Questions" part at the end of the thesis, the main objective of which is troubleshooting.

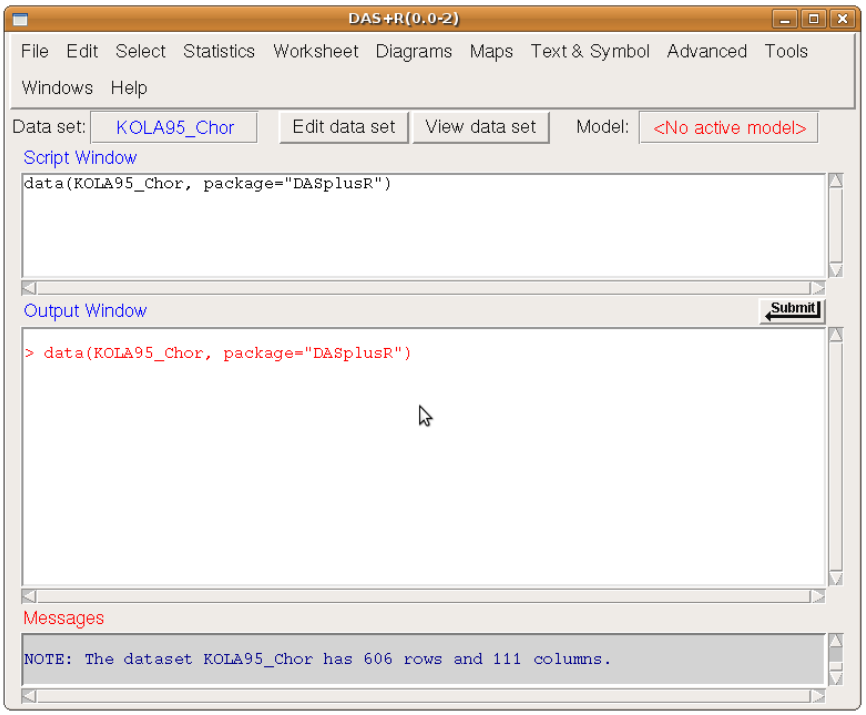

Figure 1.1: DASplusR Interface.

1.1 Motivation

O.Univ. Prof. Rudolf Dutter is currently working on an R package called `DASplusR`, a user interface, which facilitates the work with R for users who prefer mouse-clicking to the command line interface concept of R. Among many other useful programs "Spin & Brush" and "Multivariate Graphics" are also included in the UI of DASplusR.

Although DASplusR is an easy and comfortable way to access "Spin & Brush" and "Multivariate Graphics", the package `mvgraph` does not depend on DASplusR. Thus it is recommended but not necessary to use DASplusR.

For further information on DASplusR visit Prof. Dutters homepage at

http://www.statistik.tuwien.ac.at/StatDA/DASplusR/

Chapter 2

Spin & Brush

2.1 Introduction

Spin & Brush is an open source tool for visualization of three-dimensional data. The heart of the program is written in OpenGL, the rest is written in C++ and uses some Qt libraries (for further information on Qt visit http://trolltech.com/). Spin & Brush is included in the R package "mvgraph" and can be accessed by installing R (http://www.r-project.org/) and the package itself. If you encounter installation problems see Chapter 5.

2.2 Mathematical Background

2.2.1 Algebraic Operations

As mentioned before, main parts of "Spin & Brush" are written in OpenGL (Open Graphics Library) which is a cross-platform API for writing applications using 2D and 3D graphics. OpenGL is highly optimized for matrix operations such as translations or rotations which makes it very fast and powerful. Many modern computer and video games are written in OpenGL.

Rendering objects in OpenGL is restricted to so-called primitive objects like points, lines, triangles and squares. Everything else has to be constructed by combining these primitives (a cube is a combination of six squares, a sphere is a combination of many lines, and so on. Fortunately there are extensions to OpenGLs standard functions like GLU which facilitate things a little bit).

When rendering an object like a single point in OpenGL there are two ways of positioning it at the right place. The first is using global coordinates for every object which can be very cumbersome in the long run. The preferable way is manipulating the coordinate system first (which means, for example, translating the system) and then rendering the object at the origin. This second approach has many advantages over the first one: Consider a situation where the programmer wants to render 100

spheres at different positions. When using the second way it is sufficient to write a single routine which renders a sphere at the origin and translating the coordinate system between the function calls. This piece of code is much more reusable and elegant than providing global coordinates for every sphere.

Since this is a main concept of OpenGL, it may help to give a short example to make things clearer. Suppose we want to draw two circles - the first one positioned at (1, 1), the second one at (-2, 2) - in a two-dimensional space. The whole procedure consists of six steps and is illustrated in Figure 2.1:

- Translating the coordinate system by (1, 1)
- Drawing a circle at the new origin
- Resetting the coordinate system to its original position
- Translating the coordinate system by (-2, 2)
- Drawing a circle at the new origin
- Resetting the coordinate system to its original position

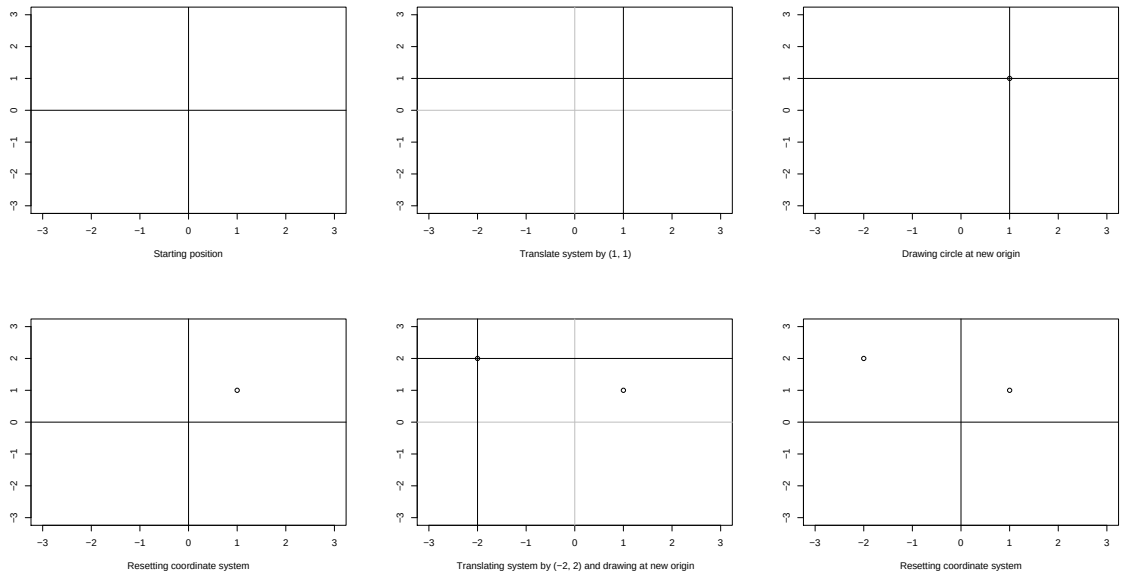

Figure 2.1: Example of OpenGL's concept of manipulating the coordinate system and drawing objects at the origin.

As one can see, OpenGL has much in common with algebraic and geometric operations like translations, scalings and rotations. Let us therefore take a closer look at these

three main algebraic operations in $\mathbb{R}^3$. In the following sections all matrices are given in the standard basis notation.

Translations The translation $\zeta_a, \mathbf{a} \in \mathbb{R}^3$ is defined by

$$\zeta_a : \mathbb{R}^3 \to \mathbb{R}^3 : \mathbf{x} \mapsto \mathbf{x} + \mathbf{a}$$

Except for the trivial translation ζ_0, translations are affine but not linear, since

$$\zeta_a(\mathbf{0}) = \mathbf{0} + \mathbf{a} = \mathbf{a} \neq \mathbf{0} \quad \mathbf{a} \neq \mathbf{0}$$

Thus a translation in $\mathbb{R}^3$ cannot be specified by a 3×3 matrix, which is not good as far as computation speed is concerned. We will return to this issue later.

Scalings The scaling $\sigma_\alpha, \alpha \in \mathbb{R}$ is defined by

$$\sigma_\alpha : \mathbb{R}^3 \to \mathbb{R}^3 : \mathbf{x} \mapsto \alpha.\mathbf{x}$$

This is already a specific scaling since all coordinates are multiplied by the same value. A more general version is the following:

$$\sigma_{\alpha,\beta,\gamma} : \mathbb{R}^3 \to \mathbb{R}^3 : \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} \alpha.x_1 \\ \beta.x_2 \\ \gamma.x_3 \end{pmatrix}$$

Here different dimensions are multiplied by different values. It is evident that $\sigma_\alpha = \sigma_{\alpha,\alpha,\alpha}$. Scalings are linear functions and can therefore be specified by a matrix. The scale matrix $\Sigma_{\alpha,\beta,\gamma}$ is given by

$$\Sigma_{\alpha,\beta,\gamma} = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix}$$

and scaling is done by a simple matrix-vector multiplication

$$\Sigma_{\alpha,\beta,\gamma}.\mathbf{x} = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix} . \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \alpha.x_1 \\ \beta.x_2 \\ \gamma.x_3 \end{pmatrix}$$

Rotations Rotations in $\mathbb{R}^3$ can be characterized by a rotation axis and a rotation angle φ. Since rotations are linear they can be specified by a rotation matrix Γ_φ, which could look like this:

$$\Gamma_\varphi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} \quad 0 \leq \varphi \leq \pi$$

In this case the X-axis is the rotation axis and the rotation angle is φ; both can easily be obtained from the rotation matrix Γ_φ. Of course the rotation axis is not restricted to one of the three coordinate system axes but can be any axis defined by a straight line through the origin. In the general case the rotation matrix may look more complex but it can be shown that every rotation matrix is of the given (simple) form when an adequate basis is chosen.

Note that the order of the operations is important, since it makes a difference if a scaling is done before a rotation or the other way round. This is also reflected by the fact that matrix multiplications are not commutative.

2.2.2 A New Dimension

When OpenGL performs one of the described operations, it internally calculates matrix products. This is done in the following way: Every point $(x, y, z)^\top$ that is rendered is plotted at position $M.(x, y, z)^\top$ where M is the socalled modelview matrix. When OpenGL is started, the modelview matrix is initialized to the identity matrix, which means that each point is rendered at its actual coordinates. For example: When a rotation is done, the modelview matrix is first multiplied by a rotation matrix and all points are rendered again. A great advantage of this approach is that the modelview matrix can be "saved" at each state and then be loaded again when it is needed. This corresponds to the "resetting of the coordinate system" which was described in the example.

We still face the problem that translations are not linear and can therefore not be specified by a translation matrix. This problem is solved by OpenGL with a little trick. Let us first take a look at the following function:

$$\rho : \mathbb{R}^3 \to \mathbb{R}^4 : \mathbf{x} \mapsto (\mathbf{x}, 1)^\top$$

where $\mathbf{x}$ is a vector in $\mathbb{R}^3$ (for example $\rho((1, 2, 3)^\top) = (1, 2, 3, 1)^\top$, ...). As we are now operating in $\mathbb{R}^4$, we are able to specify a translation ζ_a by a vector $\mathbf{a} = (x_0, y_0, z_0)^\top$ with the following 4×4 translation matrix G_a

$$G_a = \begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which leads to the matrix vector multiplication

$$\begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + x_0 \\ y + y_0 \\ z + y_0 \\ 1 \end{pmatrix}$$

Suppose the linear function represented by the translation matrix is named g_a, it is evident that

$$g_a(\rho(\mathbf{x})) = \rho(\zeta_a(\mathbf{x})) \quad \forall \mathbf{x} \in \mathbb{R}^3$$

which is exactly what we want. The basis of this technique is the following theorem:

Theorem 2.2.1 *Let $\alpha : \mathbb{R}^3 \to \mathbb{R}^3$ be an affine function. Then there is exactly one linear function $g : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^3 \times \mathbb{R}$ which satisfies*

$$g((\mathbf{x}, 1)^\top) = (\alpha(\mathbf{x}), 1)^\top \quad \forall \mathbf{x} \in \mathbb{R}^3$$

This is already a special version of the theorem which fits our conditions. A more general version (which includes semi-affine and semi-linear functions) is described in Havlicek (2003b).

In words the theorem allows us to embed our three dimensional space into a four dimensional space and get rid of matrix multiplication issues without running into trouble. Scalings $\sigma_{\alpha,\beta,\gamma}$ and rotations γ_φ remain so to say untouched as they are linear and their corresponding matrices simply become

$$\Sigma_{\alpha,\beta,\gamma} = \begin{pmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

and

$$\Gamma_\varphi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

when taking the matrices from our previous examples.

2.3 Starting the Program

If the installation of R and the package `mvgraph` was successful, we will get the following output when typing

```
> library(mvgraph)              # loads the package
> ?spin                         # displays help file
```

on the R command line:

```
spin            package:mvgraph            R Documentation

Spin and Brush

Description:
     Visualize 3d data

Usage:
     spin(x, y=0, z=0, groups=NULL)

Arguments:
       x: Matrix or data.frame containing data. When x is
          one-dimensional, y and z have to be supplied.

       y: Optional vector argument, when x is a vector

       z: Optional vector argument, when x is a vector

  groups: Group factor; has to be between 0 and 6

Details:
     Spin and Brush is a three-dimensional data viewing
     program which offers the possibility to rotate and
     zoom. Data points can be clustered by using different
     colors and identified by name or selection.

Value:
     spin returns a data.frame consisting of the
     three-dimensional data and an additional group factor
     depending on the brushed data points.

Examples:
             #data(swiss)
             #spin(swiss)
```

There are three possible ways to start the program "Spin & Brush". First of all (and this is also the recommended approach) the input argument x may be a data.frame with at least three columns, and arguments y and z are omitted. When using the R

command `data()` on an existing dataset, `R` usually creates a data.frame object in the current workspace. Thus this first approach can easily be tested by typing

```
> data(swiss)                    # loads the dataset
> spin(swiss)                    # starts the program
```

on the R command line (the dataset swiss is available on every R installation). If everything went fine, the variable selection dialog should appear (see Figure 2.2).

The second is pretty much the same as the first one with the only exception that the input argument x is a matrix with at least three columns. This can be achieved, for instance, by typing the following R commands:

```
> a <- rnorm(100)                # creates a vector
> b <- rnorm(100)                # creates a vector
> c <- rnorm(100)                # creates a vector
> myMatrix <- cbind(a, b, c)     # creates a matrix
> spin(myMatrix)                 # starts the program
```

The last possibility to start "Spin & Brush" is to use three vectors of the same length as input arguments x, y, z. This is done, for example, by creating the vectors a, b and c in the same manner as before and then typing

```
> spin(a, b, c)
```

When scanning the helpfile of "Spin & Brush" we see a fourth possible input argument called `groups` having the default value NULL. When a default value is given, it is not necessary to input the argument. For more information on the argument `groups` see Subsection 2.4.3.

2.3.1 Variable Selection Dialog

Each of the three ways will lead to the variable selection dialog, the point where the three variables to be visualized are chosen. How many variables appear in the left list

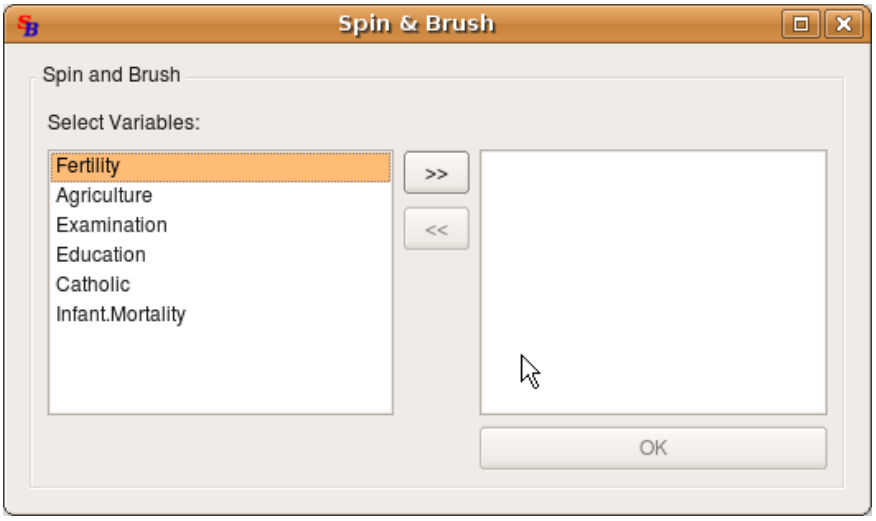

Figure 2.2: Spin & Brush's Variable Selection Dialog.

depends on the input arguments. To choose a variable simply select it by clicking it in the left list and then clicking the >> button. The variable name will move to the right list then, which means that the selection was successful. In the same way an already selected variable can be removed by clicking on it in the right list and then clicking the << button. The variable name will appear in the left list again, which means that it was successfully deselected.

Since "Spin & Brush" is a visualization tool for three dimensions, three variables have to be selected to enable the OK button, which leads to the

2.3.2 Item Selection Dialog

"Spin & Brush" provides six different symbols for visualizing data points, three sphere objects (a small one called dot and two bigger ones), two oktahedron objects and a cross object. One can select an item by clicking on the corresponding widget which results in a zoomed view with a dark background.

When using a dataset with many data points it is recommended to choose the cross object, since it is least computation intensive, while the octahedron and the sphere objects are more complex and should only be chosen when using small datasets.

After selecting an item and clicking the OK button the main program starts.

2.4 Functionality

The main window of the program consists of different parts. In the middle of the window there is the so-called view window where the data points are displayed. On the left there are some buttons that offer users the possibility to interact with the view

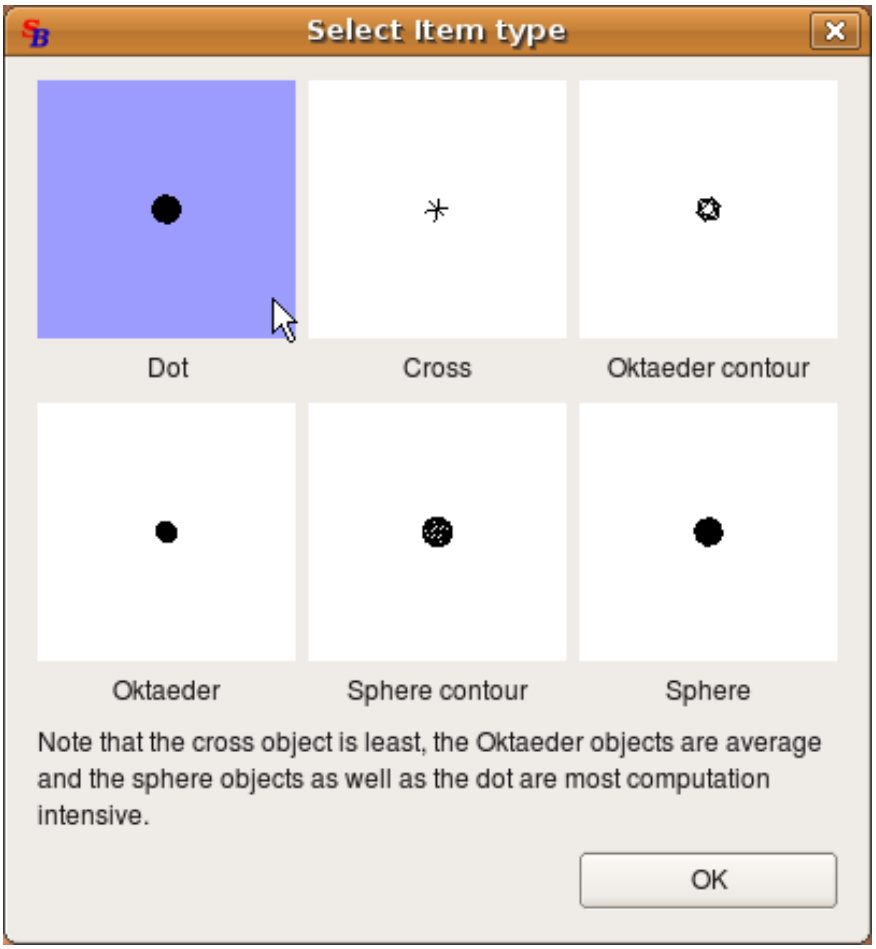

Figure 2.3: Spin & Brush's Item Selection Dialog.

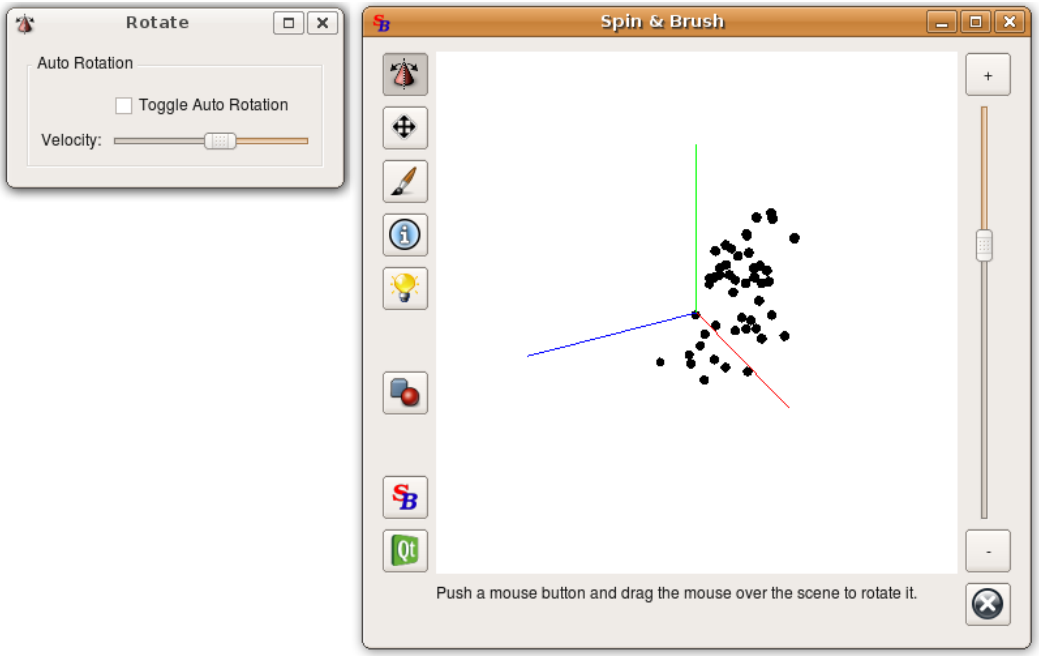

Figure 2.4: Spin & Brush's Rotation Mode.

window. On the right there are the zoom slider and the corresponding zoom buttons. Finally, at the bottom of the window there are a statusbar where status messages are displayed and the exit button which closes the window and returns control to the R interface. "Spin & Brush" provides six functions that offer the possibility to interact with the displayed scene. The following sections describe the different functions in detail.

2.4.1 Rotate

When the rotate button is checked the program is in rotation mode, which means that the user can rotate the scene by clicking the view window and dragging the mouse while pressing a mouse button. Different rotation behaviour is obtained depending on whether the left or the right mouse button is pressed. This offers the possibility to explore the data from different points of view.

When the program is in rotation mode the rotation window is visible where the user can enable auto-rotation by checking the corresponding checkbox (see Figure 2.4). The rotation speed of the auto-rotation can be specified with the speed slider. Note that rotating big scenes is computation intensive; therefore high speed rotations are not possible on some systems.

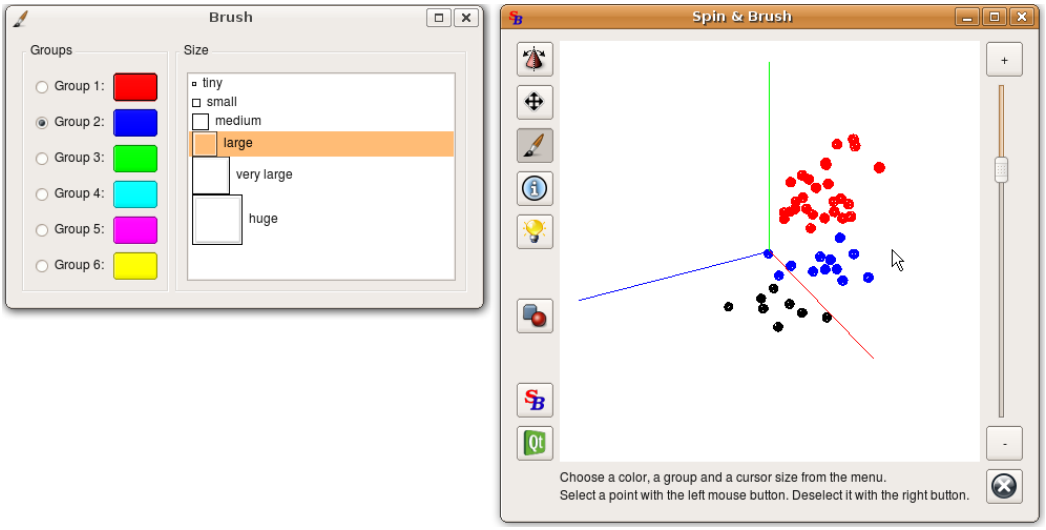

Figure 2.5: Spin & Brush's Brush Mode.

2.4.2 Move

Pressing the move button when the program is, say, in rotation mode will automatically deselect the rotate button and set the program in motion mode. The reason for this is that activating both rotation and motion mode at the same time is not possible. Dragging the mouse over the scene while the motion mode is active will result in moving the scene in the dragging direction. Thus, together with the rotation and zoom functionality, the scene can be viewed from all possible points of view.

2.4.3 Brush

Brushing is one of the main functions of the program. When the brush mode is active the brushing window, which is divided into two areas, is visible (see Figure 2.5). On the left side there are six group buttons with corresponding colors which can be chosen by clicking one of the color buttons. To set a group active one can either select one of the radio buttons or one of the color buttons.

On the right side there is a list with different cursor icons, any one of which can be selected. Then moving the mouse over the view window will transform the cursor icon into the selected one from the list. Now data points can be "brushed" by clicking them with the left mouse button. Their color will turn into the chosen color from the menu and internally the data point now is linked to the corresponding group. This can be undone by rightclicking the point, which turns its color into black again.

Brushing enables the user to cluster data points together. When exiting the program, R will return a factor variable which corresponds to the groups. Data points which have not been brushed get a value of 0, while brushed data points get a value

equal to the group number. This feature makes it possible to further process the data using the factor variable (e.g. plot by groups, etc.).

A grouping variable with values between 0 and 6 may also be used as the described input argument `groups`. When the program is started with such a grouping variable, data points are grouped by default and displayed in the corresponding colors. A good example for using a grouping variable as input is the following:

```
# load data
> data(iris)

# create index vectors
> s <- iris$Species == "setosa"
> ve <- iris$Species == "versicolor"
> vi <- iris$Species == "virginica"

# initialize and set grouping variable
> g <- rep(0, length(s))
> g[s] <- 1
> g[ve] <- 2
> g[vi] <- 3

# convert to factor
> g <- as.factor(g)

# start the program
> spin(iris, groups = g)
```

2.4.4 Info

Since data points are not labelled, it will often be desirable to identify them. This can be achieved in two ways of which the first is called info mode. Clicking a data point in the view window while the program is in info mode will open a dialog window with information about the clicked data point, containing name and coordinate values. When the clicked area of the view window contains several points, information about all of them is provided by the info dialog.

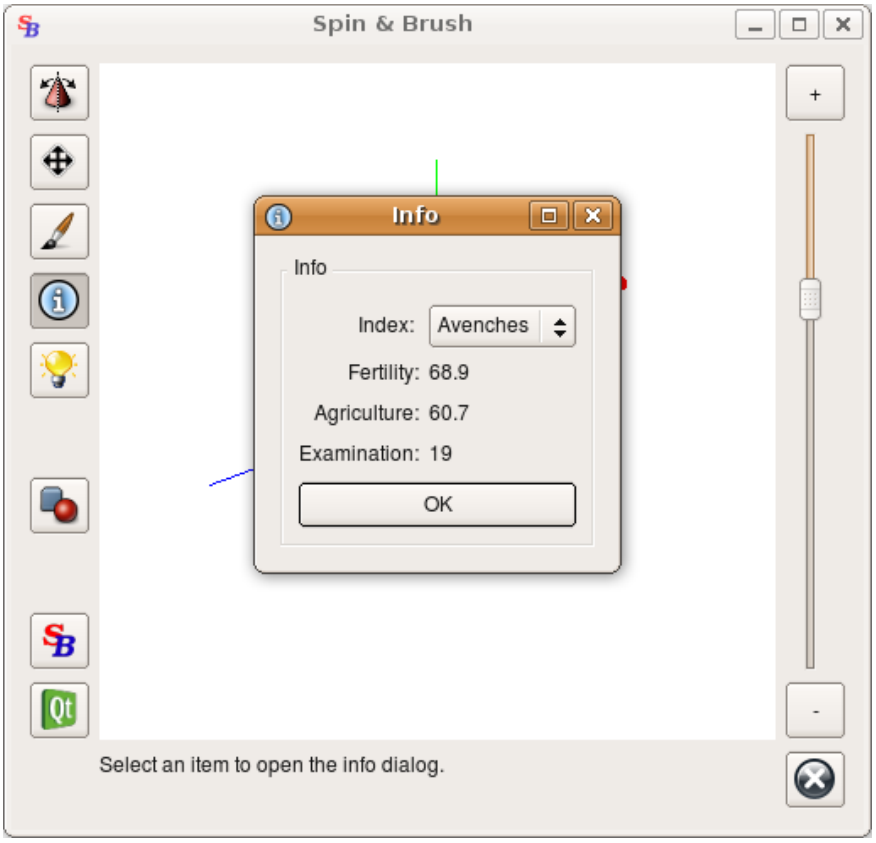

Figure 2.6: Spin & Brush's Info Mode gives detailed information to the user.

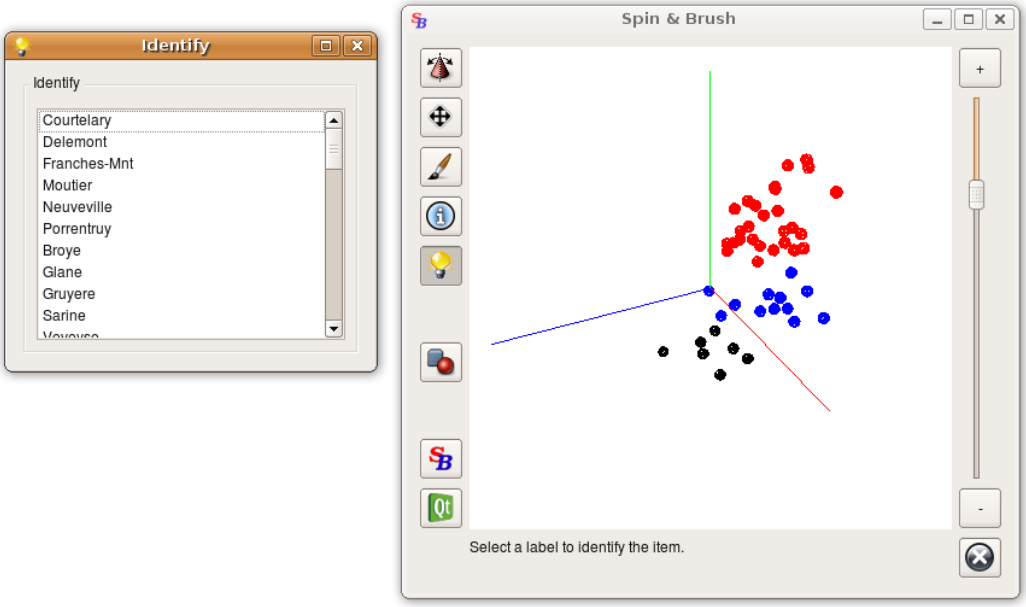

Figure 2.7: Spin & Brush's Identify Mode allows identification of data points by labels.

2.4.5 Identify

Data points can also be identified by name, which is done by activating the identify mode. This makes a window visible which contains a list of the data points labels. Selecting a label from the list will result in the blinking of the corresponding data point (see Figure 2.7).

Note that (string) labels are only available when the input argument of the program is a data.frame. Otherwise data points will be labelled with numbers.

2.4.6 Zoom

In addition to rotation and motion modes "Spin & Brush" also offers zooming functionality which extends the possibilities of viewing data. Zooming can be achieved by using the mouse wheel over the view window, pressing one of the zoom buttons or dragging the zoom slider on the right side of the window.

2.5 Memory Function

It is often desirable to reproduce results after exiting the program without doing things like brushing or choosing variables over and over again. To facilitate things, "Spin & Brush" provides a memory function which "remembers" the following settings when restarting the program with an already used dataset.

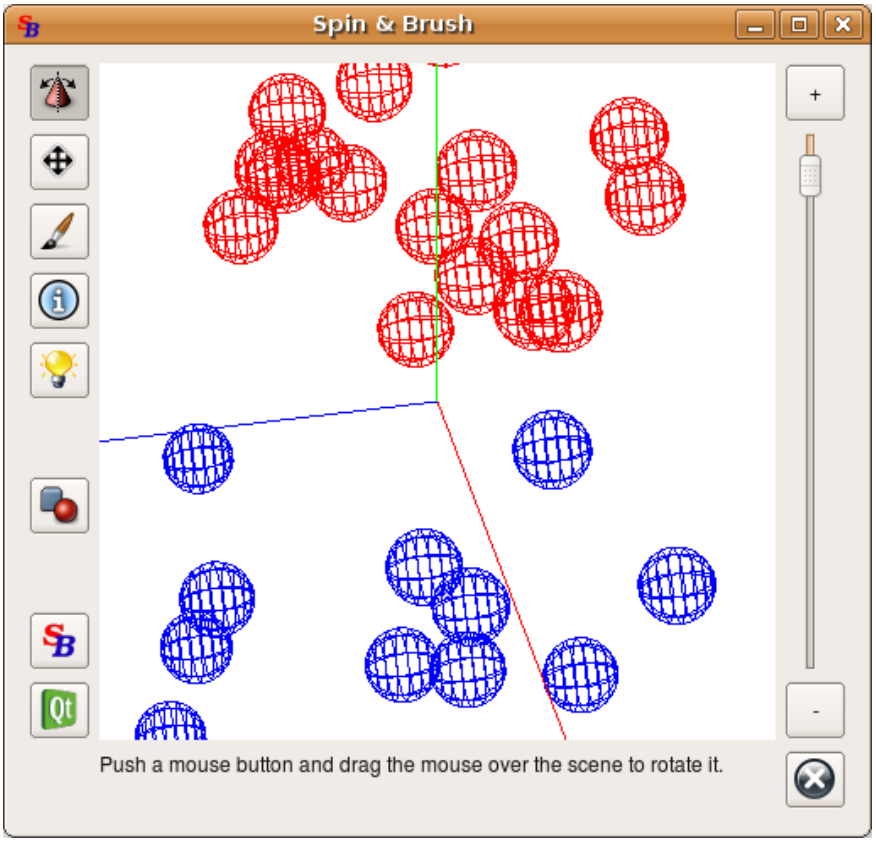

Figure 2.8: Zooming into a scene.

Variables: Especially in the case of big datasets it can be cumbersome to select and search for the same three variables again and again.

Groups: Brushed groups are remembered by the memory function.

Colors: The chosen colors of the groups are saved.

Saving these settings is achieved by creating an R list object `dataset_sb` in an environment called `.SSenv` after exiting the program. For example: When using the dataset `swiss`, an object called `swiss_sb` is created, which can be viewed by typing

```
> get("swiss_sb", env=.SSenv)
```

The output may look somewhat like this:

```
$groups

 [1] 2 2 2 2 2 2 1 1 0 2 1 1 1 1 1 1 2 0 0 1 1 1 1 1 1 1 1
[28] 1 2 1 1 1 1 1 1 1 1 1 2 0 2 0 2 2 0 1 0
Levels: 0 1 2

$selected
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE

$colors
 [1] 255   0   0   0   0 255   0 255   0   0 255 255 255   0
[15] 255 255 255   0
```

When running the program, it first looks for a memory object that corresponds to the dataset and then uses the saved settings if there is such an object.

Note that the memory function cannot save two different settings for the same dataset. A workaround for this problem would be to copy the dataset to another object with a different name. Then also the names of the created memory objects differ and nothing is overwritten.

When the input argument `groups` is specified, the group settings in the memory object are overwritten.

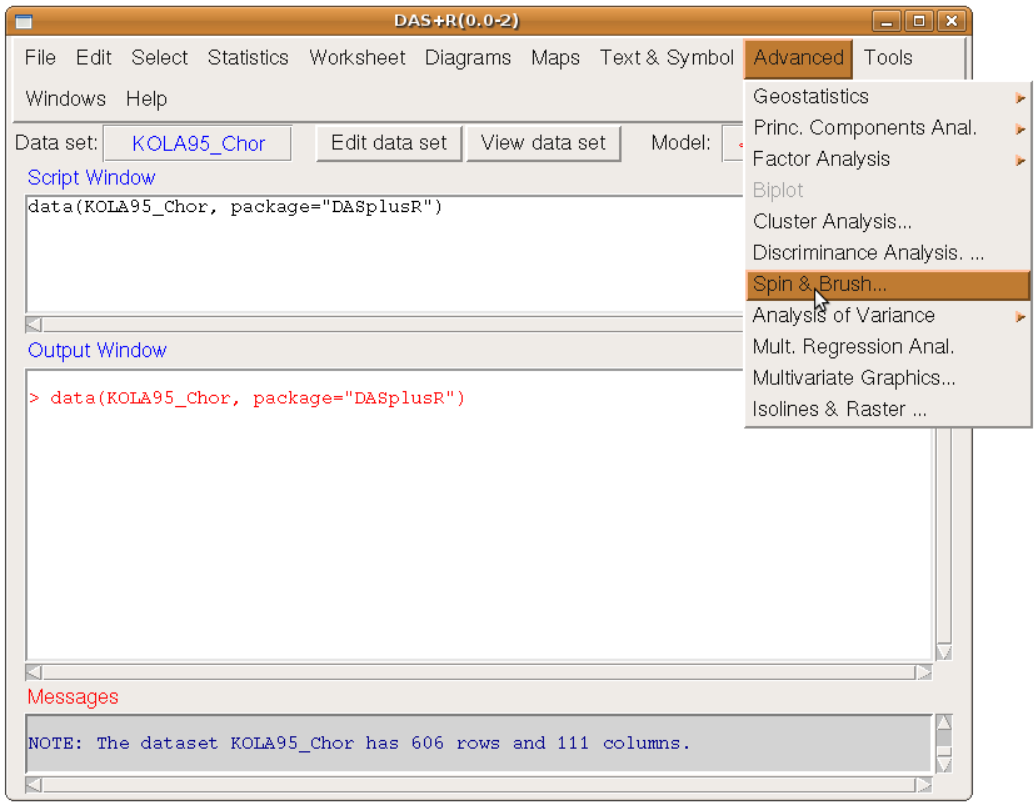

Figure 2.9: Spin & Brush - DASplusR Integration.

2.6 DAS+R Integration

As mentioned before, "Spin & Brush" is also included in the DAS+R interface which can be started by installing and loading the package DASplusR. The program can then be accessed via the menus "Advanced" / "Spin & Brush".

When starting "Spin & Brush" in DAS+R using the menus, there is no possibility to define input arguments. Thus always the current active dateset, which can be specified in DAS+R, is used.

Chapter 3

Multivariate Graphics

3.1 Introduction

Multivariate Graphics is an easy and comfortable way to display multivariate data which usually is linked to geographical coordinates. Multivariate icons such as stars, segments, boxes, etc. (see Section 3.4) are drawn at the coordinate locations, whereas size and style of the icons are determined by a subset of variables. While standard map plots can handle only one variable, up to 10-15 variables can be displayed when using Multivariate Graphics. This is a great advantage as it is often desirable and useful to compare variables when looking for correlated structures.

A disadvantage of this approach is the size of the icons which often makes it necessary to select a subset of observations to get a clearer plot. In order to get good-looking icons it is also necessary to scale variables to $[0, 1]$, since the ranges of chosen variables may differ a lot (this is done automatically by the program). Often it is convenient to plot a background map together with multivariate icons to get an impression of the geographical area corresponding to the data.

A good example of data linked to geographical coordinates is the Kola Project which is described in the next section.

3.2 The Kola Project

The Kola Project is an environmental investigation project in Arctic Europe (see Figure 3.1), which was initiated in 1992. For five years over 600 soil samples were taken covering an area of 188.000 km^2. The result is an eco-geochemical atlas containing regional distribution maps for more than 50 chemical elements in several soil layers.

Primary aims (taken from http://www.ngu.no/Kola/aims.html) of the project were

- mapping the extent of contamination by inorganic elements
- mapping the content of radionuclides

- investigating the process of trace element cycling
- establishing a soil sample bank for future investigations

The datasets are available in an R package called `StatDA` which can be downloaded from CRAN (http://cran.r-project.org/). In the following sections datasets from the Kola Project are used for examples and images.

3.3 Starting the Program

If the installation of mvgraph and StatDA was successful, we can start the program by simply typing

```
> library(mvgraph)              # loads the package
> library(StatDA)               # loads the package
> data(moss)                    # loads data from the
                                # Kola Project

> multiGraph(moss)              # starts the program
```

on the R command line interface, since `multiGraph()` takes a data.frame as argument. The dialog window in Figure 3.2 should appear: At the top of the window there is a so-called tabwidget where the type of the multivariate icon can be chosen by clicking on the corresponding tab. On every tab there is a small preview of the icon in order to get an impression of the style.

Under the tabwidget there are two drop-down boxes where the coordinate variables can be selected. Note that selecting data variables not intended to be coordinates often leads to weird results.

Below there are two list boxes for data variable selection, which works in the same way as "Spin & Brush's" Variable Selection Dialog (see Section 2.3.1). A variable being in the right list means that it is selected, while one in the left list means that it is currently not selected.

Finally there are two additional options: "Display Labels" and "Add to existing plot". Checking the first will add labels (corresponding to the labels in the used dataset) to the symbols. Enabling the second option will cause the program to draw the results in the current graphics device of R (if there is any).

3.4 Multivariate Icons

In the following subsections the multivariate icons are described and an example picture is given. The example is done using the same data over and over again by

Figure 3.1: The Kola Project Area.

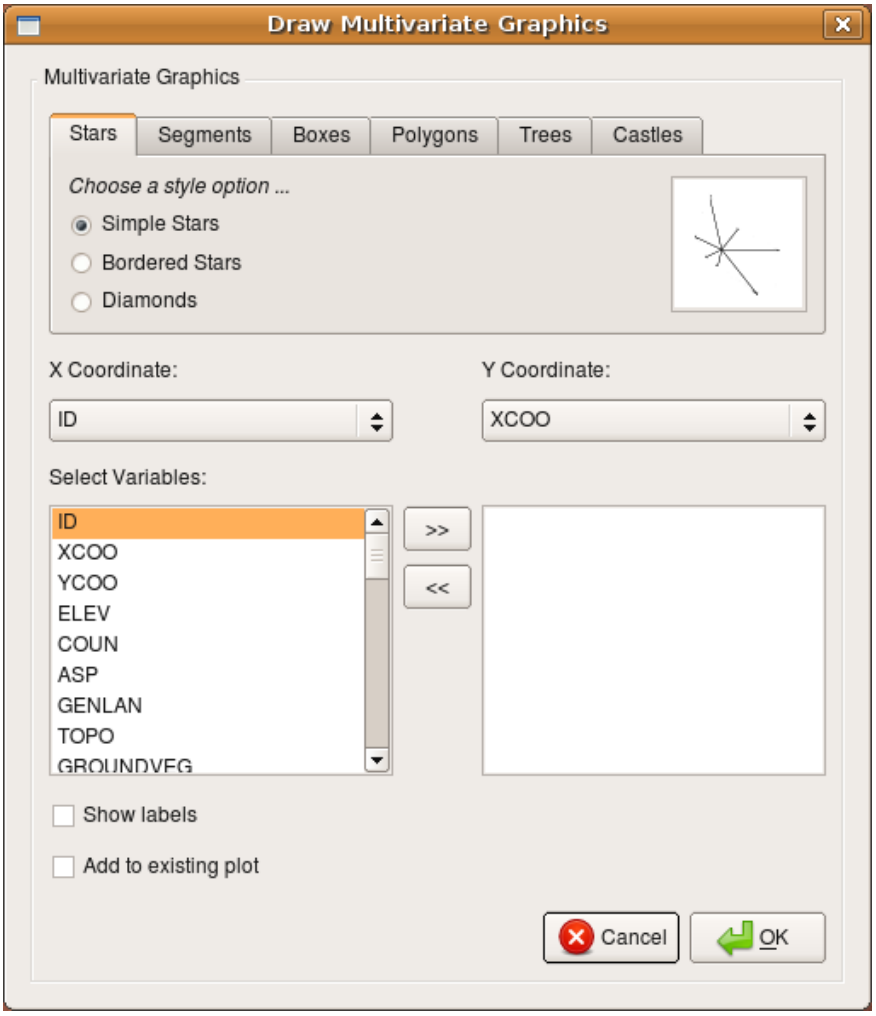

Figure 3.2: Multivariate Graphics Dialog.

executing the following code once

```
> library(mvgraph)              # loads the package
> library(StatDA)               # loads the package
> data(moss)                    # loads data from the
                                # Kola Project
> data(kola.background)         # loads background map
> myData <- moss[1:40,]         # takes subset
```

and starting the program several times with

```
> plot(moss$XCOO,moss$YCOO, frame.plot=FALSE,xaxt="n",
       yaxt="n",xlab="",ylab="",type="n")
> plotbg(map.col=c("gray","gray","gray","gray"),add.plot=T)
> multiGraph(myData)
```

each time selecting another icon.

3.4.1 Stars

Stars represent multivariate data by the length of their jags. Note that there are three different types of stars:

Simple stars are stars as a child would draw them consisting of jags only.

Bordered stars have an additional border around the jags.

Diamonds consist of the border only.

3.4.2 Segments

Segments look like pie charts, but every segment has the same interior angle, and the length of the radii differs. With the stars it is the length of their jags; with segments it is the length of their radii that describes the multivariate data. An advantage over the stars and all the other icons are the colors, which make it a bit easier to identify data correlation. Note that more than eight variables cannot be displayed by stars or segments.

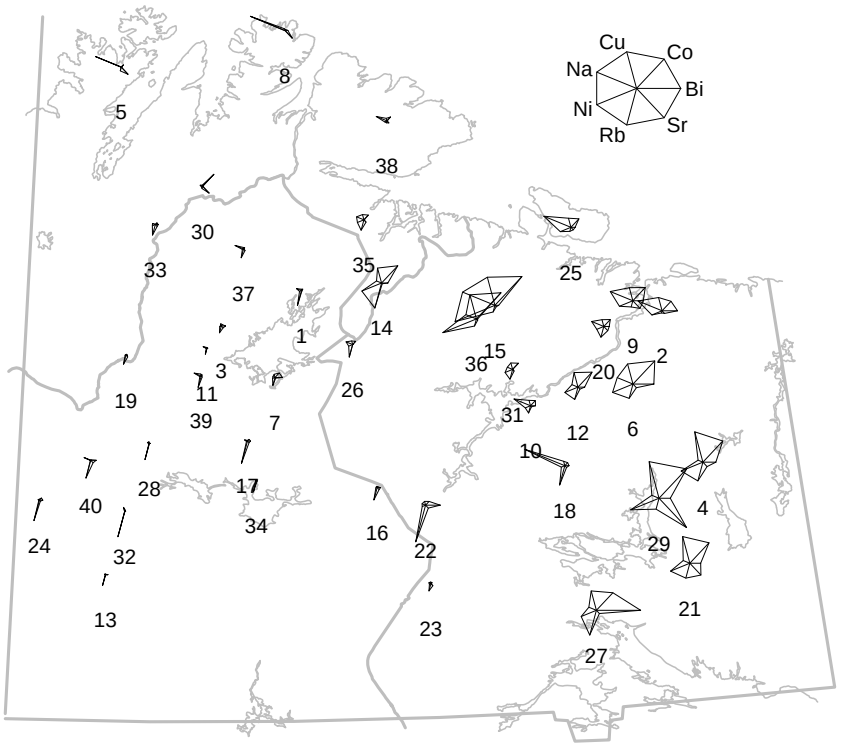

Figure 3.3: Multivariate Graphics: Stars.

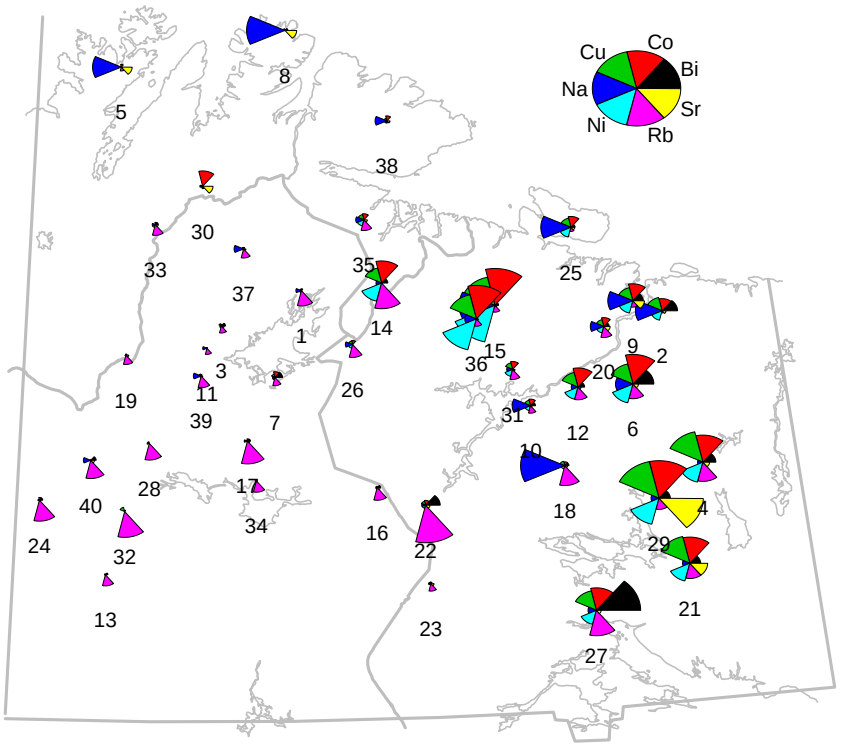

Figure 3.4: Multivariate Graphics: Segments.

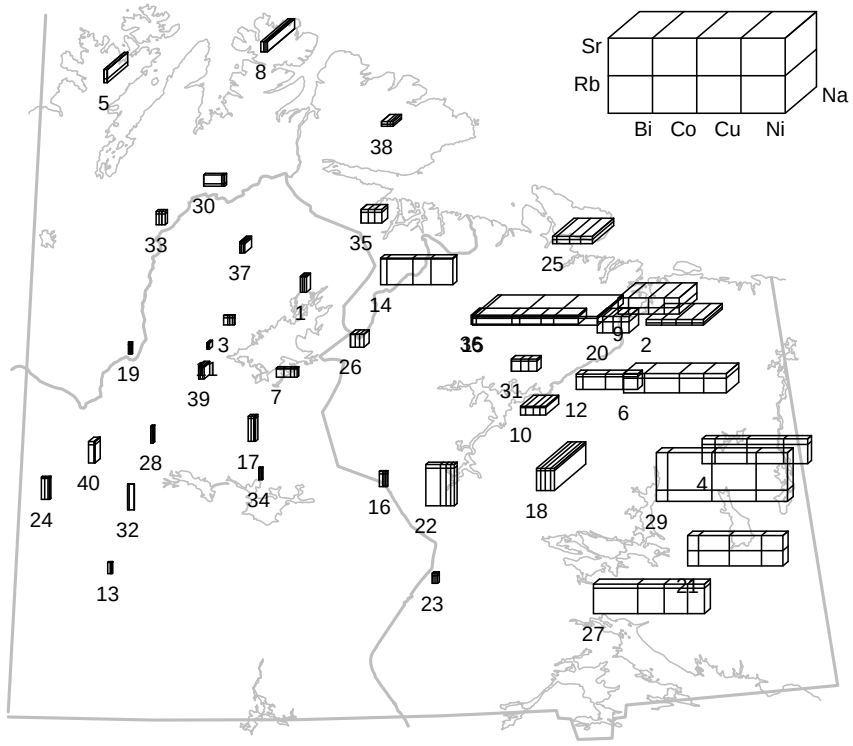

Figure 3.5: Multivariate Graphics: Boxes.

3.4.3 Boxes

Boxes are capable of displaying up to 15 variables at the same time and are therefore often used when a wide range of variables has to be displayed. The multivariate data is represented by the length, depth or height of a single box.

3.4.4 Polygons

The name "polygons" comes from their appearance which is similar to that of function graphs. The data variables are the values on the abscissa and the corresponding function values represent the data. It is recommended not to use more than five variables.

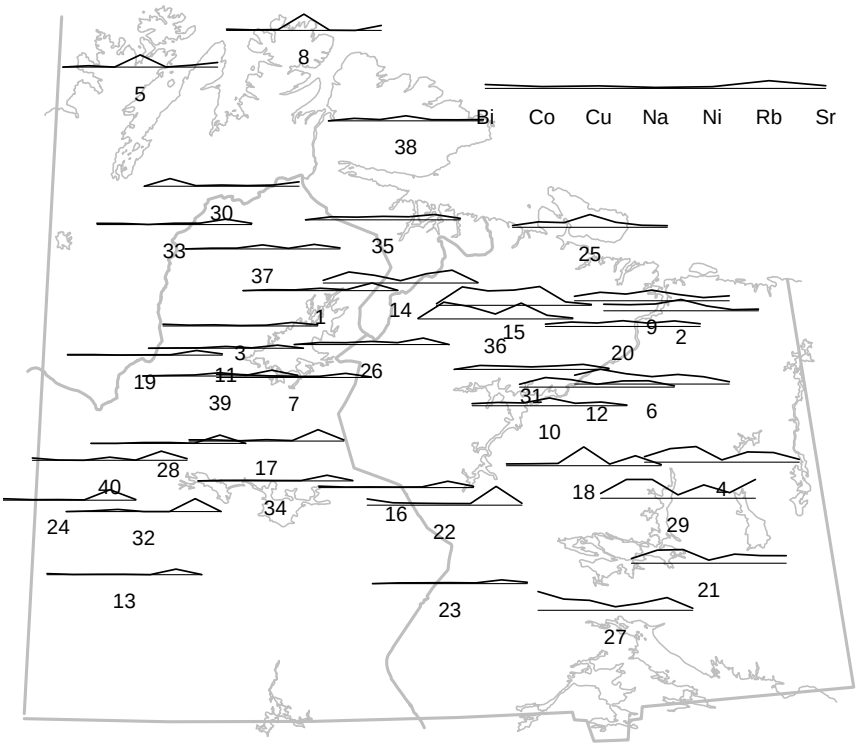

Figure 3.6: Multivariate Graphics: Polygons.

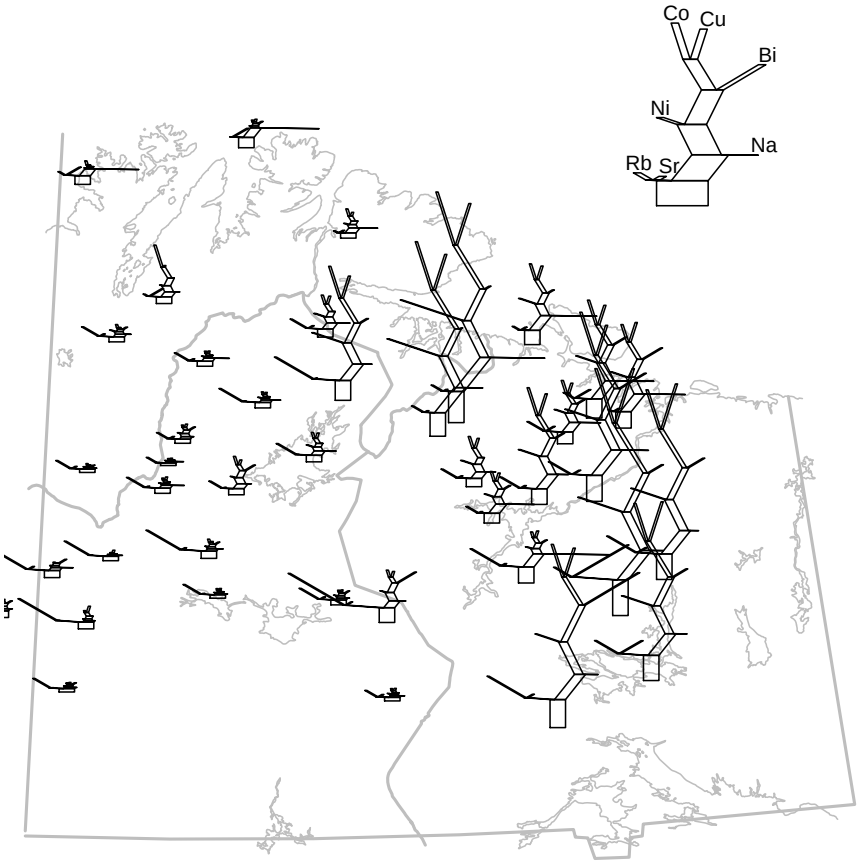

Figure 3.7: Multivariate Graphics: Trees.

3.4.5 Trees

Trees are nice looking icons which represent multivariate data by the length of their branches. Trees can easily display up to eight variables at a time.

3.4.6 Castles

Castles do not need much space and can therefore handle many variables. The data values are represented by the length of the rectangles.

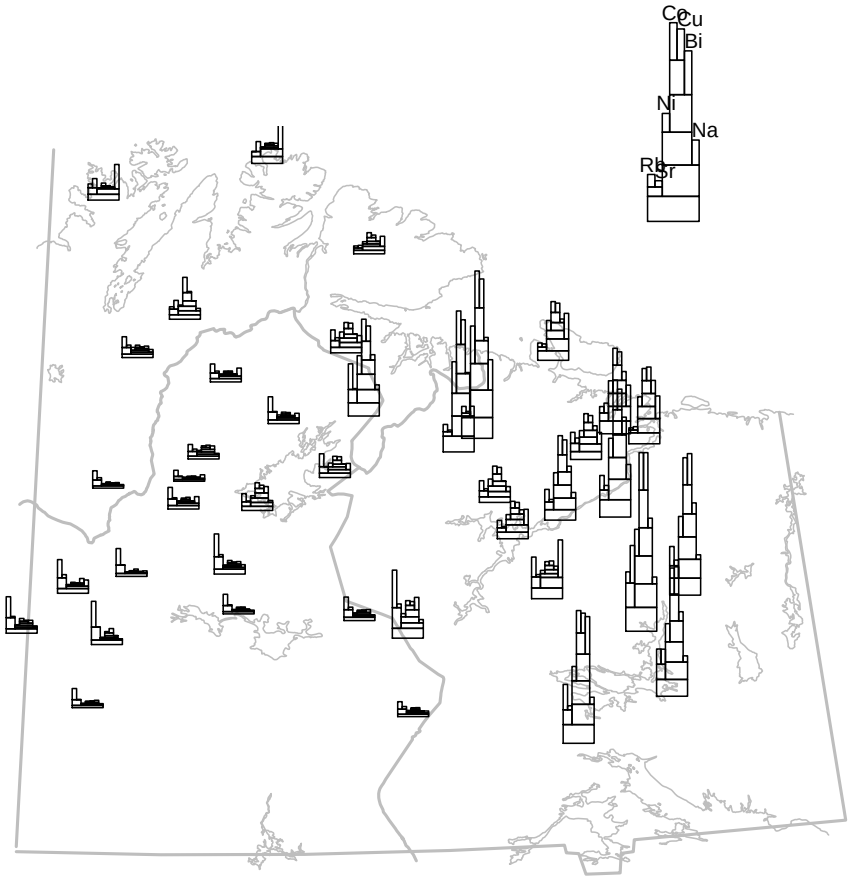

Figure 3.8: Multivariate Graphics: Castles.

3.5 Memory Function

Similar to "Spin & Brush" also "Multivariate Graphics" provides a memory function which facilitates work when repeating things. After once running the program, an R list object `dataset_mg` in an environment called `.SSenv` (the same as in "Spin & Brush") is saved, where `dataset` is the name of the used dataset (for example running `multiGraph(moss)` creates an object called `moss_mg`). When running the program with the same dataset again, the following things are stored in memory:

- Multivariate icon type
- Coordinate variables
- Data variables
- Label option
- Add to existing plot option

If there is a memory object `moss_mg`, it can be viewed by typing

```
> get("moss_mg", env=.SSenv)
```

which creates an output like this:

```
$index
[1] 1

$selected
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [9] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[21]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
[33]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
[45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

$xC
[1] 2

$yC
[1] 3
```

```
$simple
[1] TRUE

$bordered
[1] FALSE

$diamonds
[1] FALSE

$add
[1] FALSE

$labels
[1] FALSE
```

Note again that the memory function cannot save two different settings for the same dataset. A workaround for this problem would be to copy the dataset to another object with a different name. Then also the names of the created memory objects differ and nothing is overwritten.

3.6 DAS+R Integration

"Multivariate Graphics" is also included in the DAS+R interface and can be accessed over the menus "Advanced" / "Multivariate Graphics". Since no input arguments can be defined, the program is started using the current active dataset.

An advantage of the DAS+R approach is the support of geographical data, which may be provided by the DASData class. When using a DASData dataset and starting the program, it will automatically recognise the coordinate variables, if they have been specified once, and select them by default.

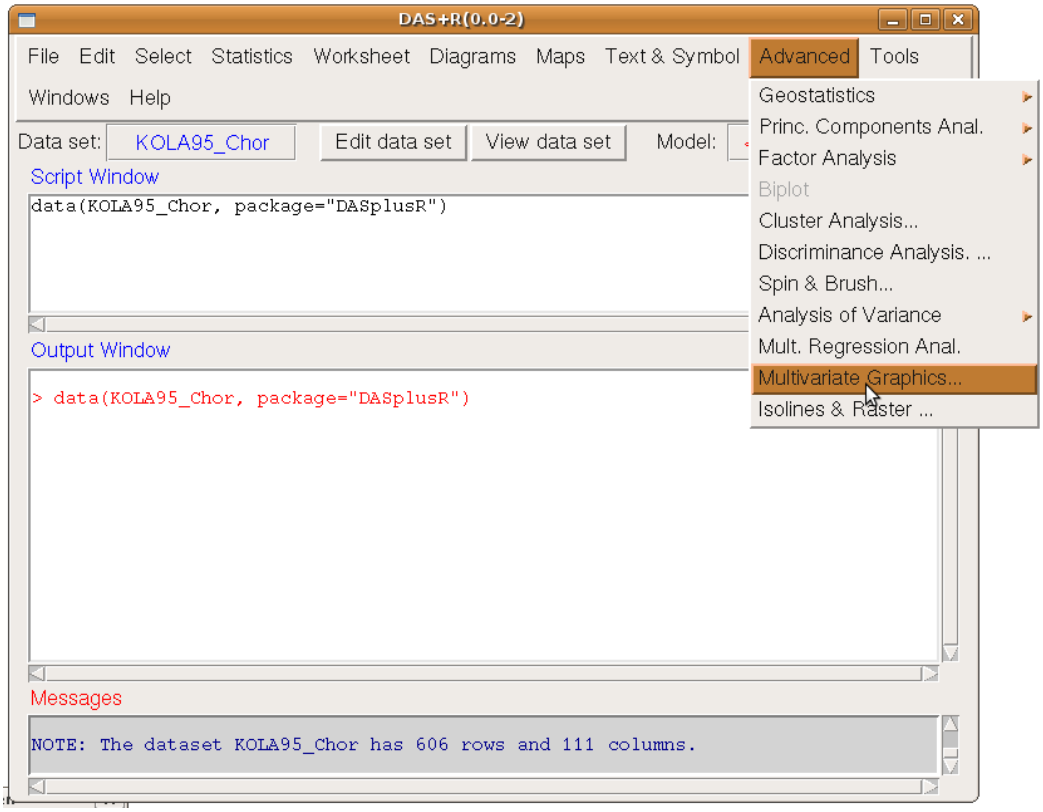

Figure 3.9: Multivariate Graphics - DASplusR Integration.

Chapter 4

Building R Packages

Building R packages is not as complicated as many users may think. R packages are a great way to extend the functionality of R and even include code from other languages like C or C++. Since `mvgraph` interfaces C++ code, let us take a closer look at this issue.

4.1 Package Structure

Few rules have to be considered when building up an R package. We will not go into detail since this is done by the manual Team (2008b), which can be downloaded from

http://cran.r-project.org/doc/manuals/R-exts.pdf,

but we will shortly explain how to build up a "minimal" package that interfaces C code. A minimal R package consists of a directory having the name of the package. This directory itself contains the following subdirectories:

R: The `R` folder contains R code, i.e. every R function which is intended to be accessed via the R command line interface after installing the package. It is not necessary to put every function in a single file; however, it is important to ensure that the files can be loaded into R via `source()` . For more information, type

```
> ?source
```

on the R command line. Files in the `R` folder are recommended to have the extension `.R` or `.r` .

man: Every function provided in the R folder has to be documented in a `.Rd` file which has to be put in the `man` folder; "man" stands for "manual". `.Rd` files have to follow a special syntax that is described in detail in Team (2008b). A typical .Rd file would look like this:

```
\name{a name}
\alias{an alias}
\title{a title}
\description{
  a description
}
\usage{
  function usage
}
\arguments{
  input arguments
}
\examples{
  some good examples
}
\keyword{some keywords}
```

When installing an R package, `.Rd` files are compiled by LaTeX which is the reason of the special syntax.

src: The `src` subdirectory is the most important when interfacing code from other languages, since it contains the source and header files for compiled code. When interfacing C/C++ code, it is recommended to use `.c/.cpp` as file extensions for the source files and `.h` for the header files. Furthermore, the `src` folder optionally contains the `Makevars` file, which contains preprocessor directives for the C/C++ compiler. This is described in detail in Chapter 5.

In addition, the package directory has to contain the so-called `DESCRIPTION` file, which provides general information about the package (version, maintainer, dependencies, etc.). The `mvgraph` `DESCRIPTION` file looks like this:

```
Package: mvgraph
Version: 1.0
```

```
Type: Package
Title: Multivariate, Interactive Visualization
Date: 2008-11-17
Author: Moritz Gschwandtner
Maintainer: Moritz Gschwandtner
            <moritz.gschwandtner@chello.at>
Depends: StatDA
Description: This package offers different possibilities to
             visualize multivariate data
License: GPL (>= 3)
```

There are some more optional subdirectories and files that may be included in the package directory; they are described in detail in Team (2008b).

4.2 Interfacing C/C++ Code

Interfacing C/C++ code from R can be done via the three interfaces `.C`, `.Call` or `.External`. In this section we will take a look at the `.C` interface, since we used it for the package `mvgraph`. Although the other interfaces are a bit more functional, the `.C` interface is slightly easier to handle and often suffices.

4.2.1 The .C Interface

The `.C` interface makes it possible to access C/C++ functions from within R. Suppose we have written a C/C++ function called `foo` (note that C++ functions to be accessed have to be included in an `extern "C" {}` statement). The next step is to compile it to a shared library object which can be loaded into R. If everything goes well, the function can then be accessed from R by typing

```
> .C("foo", ...)
```

where '...' are additional arguments corresponding to the C function parameters. It is important to know that all parameters of the C function have to be pointers, which means that all arguments are passed to the function by reference. Another important point is that every C function to be accessed via `.C` has to be of return type `void`, but all arguments passed to the function are returned to R when the execution of the function has finished. Up to 65 arguments can be passed using the `.C` interface, be

altered by the function and returned afterwards. A disadvantage of `.C` is that only basic R data types like `integer, logical, double,` or `character` can be passed. When it is necessary to pass more complex R objects like lists, data.frames, etc. to a C function, it is recommended to use `.Call` or `.External`. Note that e.g. a list may always be split up into its list entries, which is often a good workaround for this problem. The mapping between the mentioned R and C datatypes is the following:

R datatype	C/C++ datatype
`integer`	`int*`
`logical`	`int*`
`double`	`double*`
`character`	`char**`

This means that when for example passing a `logical` R object as argument to `.C` the corresponding parameter in the C function has to be of type `int*`.

4.2.2 Building a Shared Library

When we want R to access C Code via `.C`, we have to provide the code as a shared library which is typically a `.so` file on Linux and a `.dll` file on Windows. This is done by the `R CMD SHLIB` command. Suppose we have written a C source file called `foo.c` following all the rules we have discussed so far, we can compile it by typing

```
$ R CMD SHLIB foo.c
```

at a terminal which will create a shared library called `foo.so`. The name of the resulting library can be passed as additional parameter to `R CMD SHLIB` by using the `-o` directive. Typing

```
$ R CMD SHLIB -o mylib foo.c
```

will therefore create a file `mylib.so`. It is also possible to compile several source files into a single library, which is achieved by typing

```
$ R CMD SHLIB -o mylib foo1.c foo2.c foo3.c
```

Note that this process of building a shared library is done automatically when installing a source package via `R CMD INSTALL`. All source code provided in the `src` folder will be compiled to a shared object having the same name as the package.

It is often the case that one wants to interface C code which links against some other libraries. Then compiling will not work without providing information as to which libraries are needed and where these libraries can be found. This is typically done by the C preprocessor directives `-l` and `-L`. Since `R CMD INSTALL` does not take such flags as arguments, they have to be provided somewhere else. The right place for this is the `Makevars` file, which should be in the same directory as the source files (this is also the reason of putting `Makevars` in the `src` folder when building a whole package). All preprocessor directives known to the C/C++ compiler may be specified in the `Makevars` file, for example `-I`, `-L`, `-l`, `-Wall`, etc. When compiling via `R CMD SHLIB` works, there is a good chance that `Makevars` has been set up correctly. For further information on `Makevars` see Chapter 5.

4.2.3 Loading the Library

After compiling the source files, the resulting shared library has to be loaded into the R workspace. Suppose our file is named `mylib.so`: Then this can be achieved by typing

```
> dyn.load("mylib.so")
```

on the R command line. If R returns without warnings, the loading process was successful and the compiled C functions should now be accessible via `.C`. Whether an object is loaded or not can be tested by

```
> is.loaded("mylib.so")
```

which is sometimes useful. When providing a whole R package which interfaces C code, one usually wants the shared object (which was created by `R CMD INSTALL`)

to be loaded automatically when loading the package via `library()`. This can be achieved by writing a so-called `.First.lib` function, which is called automatically by R when the package is loaded. Since this is an R function, it should be placed in a file in the `R` folder (typically this file is named `zzz.R`). The `.First.lib` function of `mvgraph` looks like this:

```
.First.lib <- function(lib, pkg) {
    library.dynam("mvgraph", pkg, lib)
}
```

Note that `library.dynam()` is nearly the same as `dyn.load()`.

4.3 Installing the Package

The last step in the procedure is installing the package. Suppose the package folder is named `mypackage`. It is convenient first to check whether it is ready for installation by running

```
$ R CMD check mypackage
```

This performs several routine checks on the package (for example checking for a working LaTeX environment, checking the correct syntax, etc.) and gives information if something is wrong. If the check returns without error messages, the package can be installed via

```
$ R CMD INSTALL mypackage
```

and should be loadable by calling `library(mypackage)` in R. Another useful command in this context is `R CMD build` which builds a portable `.tar.gz` file.

4.4 An Example

To make things clearer, we will give a step-by-step example of now to build a simple R package which interfaces C code. This is only of demonstrative purpose and may be used as a template. For this section we assume that Linux and R are installed as well as the C compiler gcc.

The first step is to create a package structure as described in Section 4.1. This can easily be done by executing the commands

```
$ mkdir usefulpackage
$ cd usefulpackage
$ mkdir R
$ mkdir man
$ mkdir src
```

at a terminal. Of course, the name `usefulpackage` can be replaced by any other name.

Now it is time to write a C function which performs a simple multiplication of two integers. It is clear that this could be done in R by simply using the `*` operator, but as mentioned, this is only a simple example. We change to our `src` directory

```
$ cd src
```

open an editor and create the following file `mult.c`:

```
#include <stdlib.h>

// function declaration
void mult(int* a, int* b, int* ab);

// function definition
void mult(int* a, int* b, int* ab) {

  *ab = (*a)*(*b);
```

```
}
```

Since the C functions to be interfaced by R have no return type, we have to provide a variable which holds the result of the computation as parameter (in this case this is the variable `ab`). Note that all parameters of the function are pointers.

We now need a corresponding R function which interfaces the C function and has to be put in the `R` folder.

```
$ cd ../R
```

Again we open an editor and create a file called `mult.R`:

```
mult <- function(x, y) {

  # check input:
  if(is.null(x) || is.null(y))
    stop("Two input arguments have to be given!")

  # call to C function mult:
  result <- .C("mult", as.integer(x), as.integer(y),
               as.integer(0), PACKAGE="usefulpackage")

  # return third argument of the C function mult:
  result[[3]]
}
```

Note how the call to the C function is done: The first argument is the name of the function followed by all the arguments needed by the C function. The `as.integer` calls ensure that a correct data type conversion between R and C is done. Since the third parameter `ab` holds the result of the multiplication and is altered within the C function, we can simply input a 0. The optional `PACKAGE` argument specifies where to look for the symbol `mult`. The three parameters `a`, `b` and `ab` of the C function are

returned as a list object; we save it in a variable `result` and return only the third element, which actually is our product.

As mentioned earlier, when running `R CMD INSTALL`, all source code will later be compiled to a shared object called `usefulpackage.so`. We want this shared object to be loaded automatically when our package is loaded via `library(usefulpackage)`; so we create a R function `.First.lib` and put it in a file called `zzz.R`:

```
.First.lib <- function(lib, pkg) {
  library.dynam("usefulpackage", pkg, lib)
}
```

The next step is to write a documentation file for our function `mult.R`. We change to the `man` directory

```
$ cd ../man
```

and create a file `mult.Rd` which should look somewhat like this

```
\name{mult}
\alias{mult}
\title{A Simple Multiplication}
\description{
  Multiplication of two Integers
}
\usage{
  mult(x, y)
}
\arguments{
  \item{x}{An Integer}
  \item{y}{An Integer}
}
\examples{
  mult(2, 5)
```

```
}
\keyword{file}
```

The only thing that's left for a simple R package is the `DESCRIPTION` file, which has to be put in the package directory

```
$ cd ..
```

and could look like this:

```
Package: usefulpackage
Version: 1.0
Type: Package
Title: A Useful Package
Date: 2009-03-01
Author: Somebody
Maintainer: Somebody <some@body.at>
Description: This package provides some useful things.
License: GPL (>= 3)
```

That's it. Now it is time to check our package for possible errors, so we run

```
$ cd ..
$ R CMD check usefulpackage
```

and get the following output (on other systems the output may differ slightly):

```
* checking for working pdflatex ... OK
* using log directory
```

```
  '/home/moritz/Dokumente/usefulpackage.Rcheck'
* using R version 2.7.1 (2008-06-23)
* using session charset: UTF-8
* checking for file 'usefulpackage/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'usefulpackage' version '1.0'
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking whether package 'usefulpackage' can be
  installed ... OK
* checking package directory ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... OK
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated
  dependencies ... OK
* checking for unstated dependencies in R code ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
* checking R code for possible problems ... OK
* checking Rd files ... OK
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
* checking Rd \usage sections ... OK
* checking line endings in C/C++/Fortran
  sources/headers ... OK
* checking line endings in Makefiles ... OK
* checking for portable use of $BLAS_LIBS ... OK
* creating usefulpackage-Ex.R ... OK
* checking examples ... OK
* creating usefulpackage-manual.tex ... OK
* checking usefulpackage-manual.tex using pdflatex ... OK
```

Since there are no errors reported, we can install our package by typing

```
$ R CMD INSTALL usefulpackage
```

The last step is to start R and try out our multiplication

```
$ R
> library(usefulpackage)
> mult(-3, 4)
[1] -12
```

Chapter 5

Installation Instructions

As mentioned, `mvgraph` is an R package that interfaces C/C++ code which links against Qt libraries (for further information on Qt visit http://trolltech.com/). In order to make the package work properly, it is necessary to install Qt as well as the GNU C and C++ compilers `gcc` and `g++` on your system. In this chapter we assume that R and the compilers are already installed. I will give a step by step tutorial on how to install the package.

5.1 Package Dependencies

The package `mvgraph` depends on the package `StatDA` which in turn depends on several other packages like `akima`, `car`, `cluster`, ... The first step is to install all of these dependencies which will be done automatically when installing `StatDA`. This can be achieved by simply typing

```
> install.packages("StatDA")
```

on the R command line. A mirror list will appear asking you for a location near you. After selecting a mirrow, download and installation starts. If you don't have an internet connection you will have to get the packages from somewhere else and install them manually. For more information on installing R packages see Team (2008b). The package dependencies are shown in Figure 5.1.

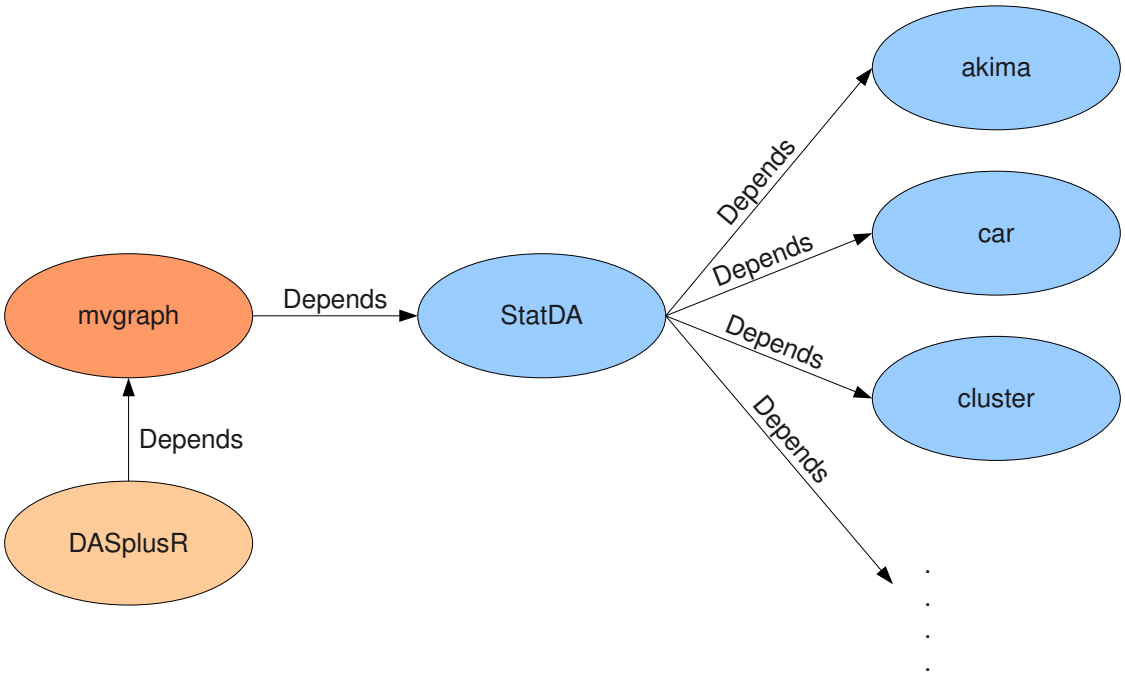

Figure 5.1: Installation - Package Dependencies.

5.2 Installation on Linux

5.2.1 Installing Qt

Installing Qt is the next step in the installation process. There are two ways this can be done:

- Installation of Qt with a package manager
- Installation of Qts sources

Although the first way is much faster and also easier, many Linux users prefer the second approach since it is more flexible. I will describe both of them.

Most Linux distributions such as Ubuntu or SUSE have package managers which make installation of new software fast and easy. On Ubuntu, the package manager is called `synaptic` and can be started with the command

```
$ sudo synaptic
```

Since we want to install new packages, we need to be superuser and therefore use the `sudo` command. The synaptic package manager allows us to search for strings. Search for `qt` and make sure that the following packages are installed,

```
libqt4-dev
libqt4-gui (libqtgui4)
libqt4-core (libqtcore4)
libqt4-opengl
libqt4-opengl-dev
```

which can be done by selecting the checkboxes in the first column (if they are not selected yet!). The result should be similar to Figure 5.2. The procedure on other Linux distributions is the same as above except that the package managers have different names (for example the manager on SUSE can be found in YAST).

The second way to install Qt is to install it from source. Visit http://trolltech.com/ and download the newest version of Qts sources (at the moment this is version 4.5) for Linux. Unpack the archive, open a shell and change to the unpacked directory. The following commands will install Qt on your system:

```
$ ./configure
$ make
$ make install
```

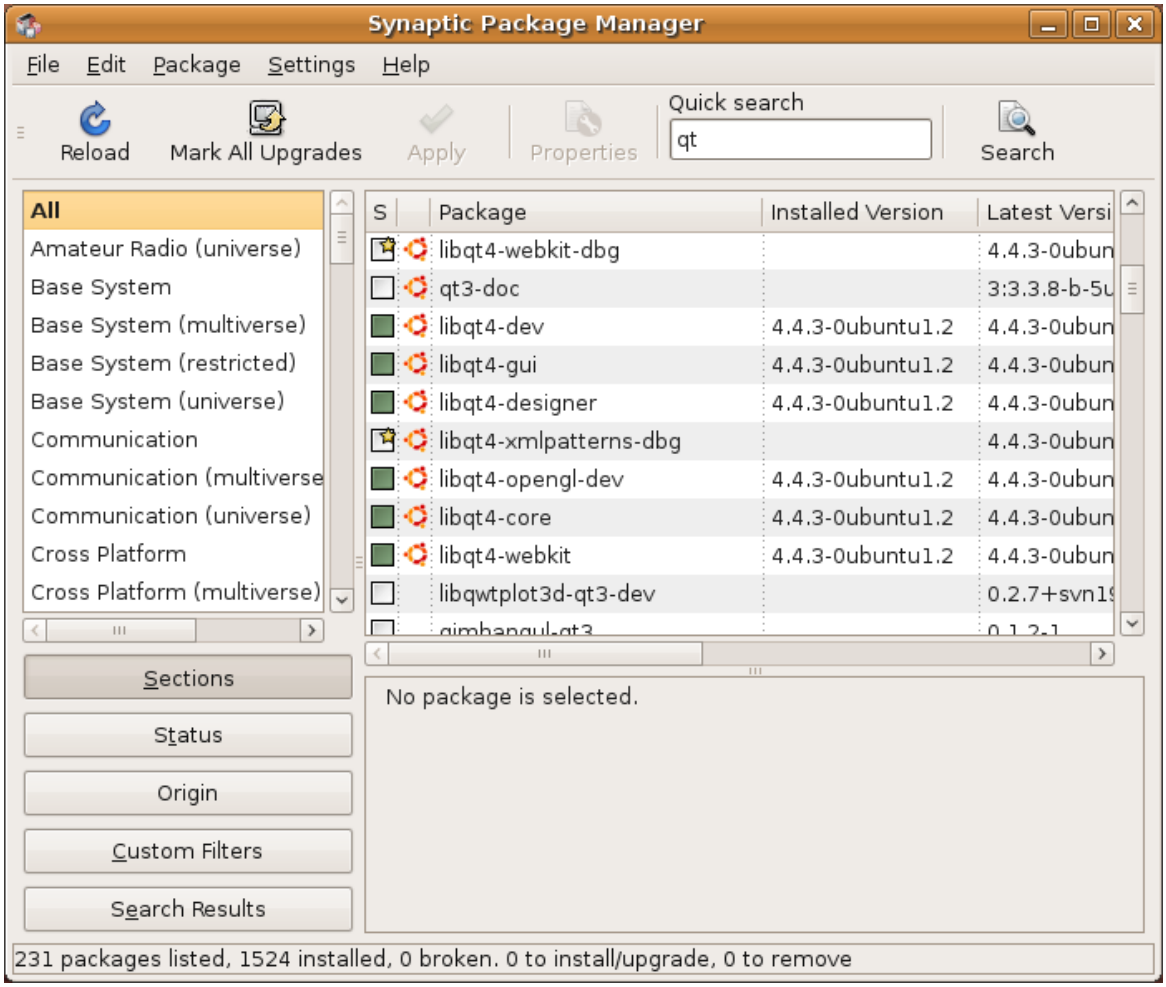

Figure 5.2: Installing Qt on Ubuntu.

Each of the commands will take some time; the whole installation may take up to two hours or more, even on fast systems. Thus it is recommended to install Qt with a package manager.

5.2.2 Configure

If `pkg-config` is installed on your system, we recommend you to make use of the `configure` file included in the package directory. The `configure` script tries to detect the locations of your Qt library and include files. It is called automatically by `R CMD INSTALL` and creates a `Makevars` file which is adapted to your system.

We recommend you to skip the next section and to go ahead with Section 5.2.4. If this approach does not work, you may want to remove the `configure` file from the package directory (otherwise the configure skript will always try to create a new `Makevars` file when running `R CMD INSTALL mvgraph`) and go ahead with Section 5.2.3.

5.2.3 Makevars

We need to find out where the Qt libraries and header files were installed to, since we want to adapt the `Makevars` file in the `src` folder of the `mvgraph` package. If you installed Qt with a package manager there is a good chance that Qts libraries were installed to

```
/usr/lib
```

and the include files to

```
/usr/include/qt4
```

Note that these paths may differ from system to system depending on your linux distribution and other settings. If you installed Qt from source, the files can most likely be found in

```
/usr/local/Trolltech/Qt-4.3.4/lib
```

and

```
/usr/local/Trolltech/Qt-4.3.4/include
```

where `4.3.4` is to be replaced with the correct version number.

A way to find the correct paths is to search for typical Qt files like `qlabel.h` (which is a header file and should lead to the correct include path) or `libQtCore.so` (which is a library and should lead to the correct library path). Make sure that you have found the correct paths, change to your `mvgraph/src` directory and open `Makevars` which should look like this:

```
DEFINES = -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED
          -DQT_OPENGL_LIB
PKG_CPPFLAGS = -I/usr/share/qt4/mkspecs/linux-g++
               -I/usr/include/qt4/QtCore
               -I/usr/include/qt4/QtGui
               -I/usr/include/qt4 -I/usr/include/qt4/QtOpenGL
               -I.
PKG_CFLAGS = -pipe -W -D_REENTRANT $(DEFINES)
PKG_CXXFLAGS = -pipe -W -D_REENTRANT $(DEFINES)
PKG_LIBS = $(SUBLIBS)  -L/usr/lib -lQtGui -lQtCore -lpthread
           -lQtOpenGL -lGLU
```

Replace every occurrence of `/usr/include/qt4` with the correct include path and `/usr/lib` (which actually occurs just once) with the correct library path.

5.2.4 Installing mvgraph

We can now change to `mvgraph`s parent directory and run

```
$ R CMD check mvgraph
```

which performs some routine checks on the package. If there is an error concerning the line

```
* checking whether package 'mvgraph' can be installed ...
```

it is most likely that the paths in the `Makevars` file are wrong. Or else, there could be a problem with the C/C++ compilers `gcc` and `g++`. When running `R CMD check`, a directory `mvgraph.Rcheck` is created containing a file called `00install.out`. Installation problems are logged and saved to this file. If no problems are encountered, we can install the package by typing

```
$ R CMD INSTALL mvgraph
```

Congratulations!

5.3 Installation on Windows

Installing R packages on Windows is slightly more complicated, since some programs that are standard components on every modern Linux distribution have to be installed first. This includes MinGW (which stands for "Minimalist GNU for Windows and contains the compilers `gcc` and `g++`), Perl and optionally LaTeX and the HTML Workshop, which are both needed for compiling documentation files. Furthermore some Unix tools have to be installed in order to make things work. Another issue that always plays a role when installing R source packages on Windows is setting the `PATH` system variable correctly, so all programs can be found during the installation routine.

In this section we assume that `R` is already installed in `C:\R-2.8.2` and Perl is installed in `C:\Perl`. The next step is installing Qt.

5.3.1 Installing Qt

Visit http://trolltech.com/ and download the newest version of Qts open source installer for windows. Trolltech offers a package which includes Qt and MinGW. This file is typically called `qt-win-opensource-version-mingw.exe`, where `version` stands for the current version number. After downloading and executing the file, installation starts (see Figure 5.3). The installer will guide you through the installation process. You should remember the locations where Qt and MinGW are installed, since these paths are important when setting the `PATH` environment variable and adapting the `Makevars.win` file. From now on we assume that Qt has been installed to `C:\Qt\4.4.3` (see Figure 5.4) and MinGW to `C:\MinGW`.

5.3.2 Installing Unix Tools

Since `R CMD check` makes use of commands like `sh`, it is necessary to provide these programs on windows. There are several projects for this purpose, but we recommend to visit http://unxutils.sourceforge.net/ and download the newest version. From now on we assume that the utilities are installed in `C:\Unxutils`.

Figure 5.3: Installing Qt on Windows.

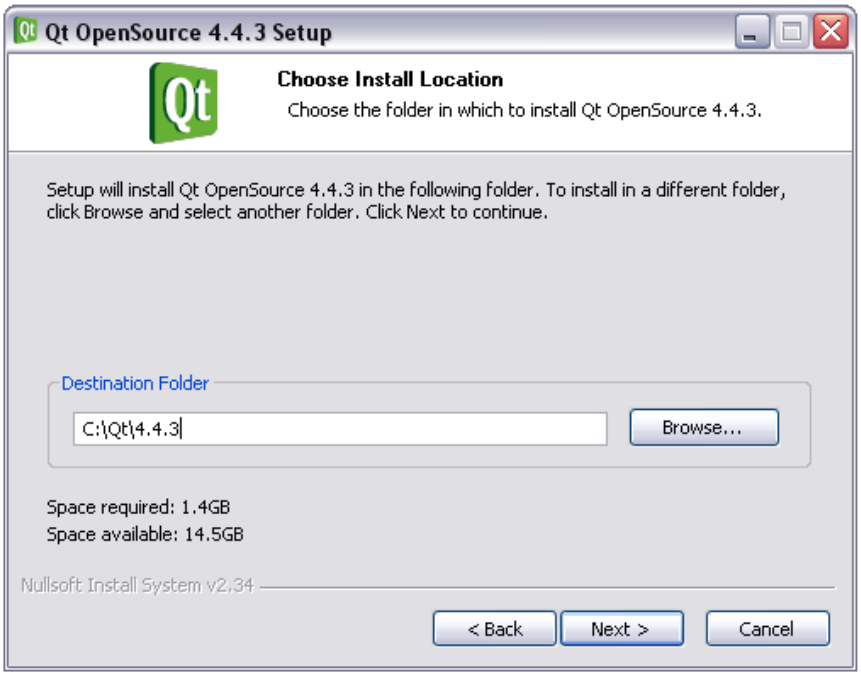

Figure 5.4: Setting Qt's installation directory.

5.3.3 Setting the PATH Environment Variable

In order to have Windows find all programs needed for the installation of R source packages, one has to adapt the `PATH` environment variable: Add the following entries to the `PATH` variable:

```
C:\Qt\4.4.3\bin;
C:\MinGW\bin;
C:\Perl\bin;
C:\R-2.8.1\bin;
C:\Unxutils;
```

Note that these paths are only assumptions we make in this chapter and may differ from system to system. It is therefore necessary to discover the correct locations of the programs listed above, and replace the paths accordingly. In the same manner, paths to the LaTeX and the HTML Workshop directories have to be added.

5.3.4 Makevars.win

When installing `mvgraph` on Windows, the `Makevars.win` file takes precedence over the `Makevars` file used on Linux. As on Linux, the file has to be adapted to your system so the Qt and OpenGL libraries can be linked successfully. If the path assumptions we have made so far are correct, your `Makevars.win` file should look like this:

```
DEFINES = -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB
          -DQT_SHARED -DQT_OPENGL_LIB
PKG_CPPFLAGS = -I'C:\Qt\4.4.3\include'
               -I'C:\Qt\4.4.3\include\QtCore'
               -I'C:\Qt\4.4.3\include\QtGui'
               -I'C:\Qt\4.4.3\include\QtOpenGL'
PKG_CFLAGS = -pipe -W -D_REENTRANT $(DEFINES)
PKG_CXXFLAGS = -pipe -W -D_REENTRANT $(DEFINES)
PKG_LIBS = $(SUBLIBS)  -L'C:\Qt\4.4.3\lib' -L'C:\MinGW\lib'
           -lQtGui4 -lQtCore4 -lQtOpenGL4 -lglu32 -lopengl32
```

Note again that these paths have to be adapted to your system.

5.3.5 Installing mvgraph

We can now change to the `mvgraph`s parent directory and run

```
$ R CMD check mvgraph
```

If no error messages are issued, the package is ready for installation, which is achieved by

```
$ R CMD INSTALL mvgraph
```

Congratulations!

Appendix A

FAQ

This section is mainly focused on troubleshooting as far as the installation of mvgraph and the usage of "Spin & Brush" and "Multivariate Graphics" are concerned. By the way: Although this is the FAQ part, not all of the following points are questions.

A.1 When running R CMD check mvgraph ...

A.1.1 I get the warning " * checking if this is a source package ... WARNING: Subdirectory 'mvgraph/src' contains object files"

This warning occurs when the sources of mvgraph have already been compiled (for examply by R CMD check) and can be ignored.

A.1.2 I get the message " * checking for working pdflatex ...sh: pdflatex: not found NO"

This warning occurs when LaTeX is not installed or cannot be found. Install LaTeX (on Linux this can be done via package manager, on Windows visit http://miktex.org/ and download the newest version) and/or check, if the LaTeX/bin directory is added to the `PATH` environment variable. The warning will not cause the installation of `mvgraph` to fail, but due to it, no manual for `mvgraph` might be created.

A.1.3 I get the error " * checking package dependencies ... ERROR: Packages required but not available: StatDA"

Since `mvgraph` depends on the R package `StatDA`, it is necessary to install `StatDA` first. Try typing

```
> install.packages("StatDA")
```

on the R command line interface. This will only work, if an internet connection is available.

A.1.4 I get the error " * checking whether package 'mvgraph' can be installed ... ERROR: Installation failed. See '[..]/00install.out' for details."

Different reasons may lead to this error message. Thus it is recommended to scan `00install.out` for troubleshooting. We will list some of the possible reasons:

/bin/bash: gcc/g++: command not found: If `00install.out` contains this or a similar message, there is a problem with your C/C++ compiler. On Windows, check if MinGW is installed and the MinGW/bin directory is added to the `PATH` environment variable. On Linux, try (re-)installing `gcc/g++` via a package manager.

QTimer: No such file or directory: QTimer is only an example. If this or a similar message appears, there is something wrong with the include path, since the Qt include files cannot be found. If Qt is not installed, see Chapter 5 for installation instructions. If it is, most likely the include path in the `mvgraph\src\Makevars` file is wrong. How to set up the `Makevars` file correctly is also explained in Chapter 5 and its subsections.

/usr/bin/ld: cannot find -lQtCore: If this error occurs, the Qt libraries cannot be found by the compiler. Check if Qt is installed (see Chapter 5) and if `Makevars` is set up correctly.

A.1.5 I get the error " * checking whether the package can be loaded ... ERROR"

This error will most likely occur if there is a problem with the library path in the `Makevars` file. In this case, the C/C++ compiler doesn't link against the Qt libraries, and a message like this appears:

```
Error in dyn.load(file, DLLpath = DLLpath, ...) :
  unable to load shared library '[..]/mvgraph.so':
  [..]/mvgraph.so: undefined symbol: _ZNK7QDialog10metaObjectEv
Error in library(mvgraph) : .First.lib failed for 'mvgraph'
Execution halted.
```

On Windows a new window may open saying "The application could not be started, because QtCore4.dll could not be found. Reinstalling the application could possibly solve the problem."

Check if Qt is installed (see Chapter 5) and if `Makevars` is set up correctly. On Windows, check if the Qt/bin directory is added to the `PATH` environment variable.

A.1.6 I get the error "The command 'sh' is either wrongly spelled or could not be found."

It is improbable that something is misspelled. More likely on Windows this may happen, if Unix utilities like `sh` are not installed or can not be found. See Chapter 5.3.2 for installation instructions and make sure that the directory containing `sh.exe` is included in the `PATH` environment variable.

A.1.7 I get the error "The command 'perl' is either wrongly spelled or could not be found".

On Windows this may happen, if `perl` is not installed or cannot be found. See Chapter 5 for installation instructions and make sure that the perl/bin directory is included in the `PATH` environment variable.

A.1.8 I get the error "The command 'R' is either wrongly spelled or could not be found".

On Windows this may happen, if `R` is not installed or cannot be found. Visit http://www.r-project.org/, download and install the newest `R` version and add the R/bin directory to the `PATH` environment variable.

A.2 When running R CMD INSTALL mvgraph ...

A.2.1 I get an error. What can I do?

`R CMD INSTALL mvgraph` corresponds 1:1 to the line

```
* checking whether the package 'mvgraph' can be installed ...
```

when running `R CMD check mvgraph`, so also the error messages correspond. See Section A.1.4 for troubleshooting.

A.2.2 Everything works fine, but when I try loading mvgraph in R I get an error. What's wrong?

It is possible that installing the package succeeds but loading fails. Most likely there is a linker problem. Try running `R CMD check mvgraph` and see Section A.1 for further details.

A.3 Spin & Brush

A.3.1 Why isn't it possible to choose more than three variables in the "Spin & Brush" variable selection dialog?

Since "Spin & Brush" is inteded to visualize three-dimensional data, only the selection of exactly three variables will work.

A.3.2 Why is the OK button in the "Spin & Brush" variable selection dialog disabled?

This happens, if fewer than three variables are selected. You have to select exactly three.

A.3.3 In the "Spin & Brush" item selection dialog the items aren't displayed correctly. What could be the problem?

There is most likely a problem with your graphics card or OpenGL. Check if your graphics card supports OpenGL or try installing newer graphic drivers.

A.3.4 Why is the rotation of data points so slow?

This may happen if many data points are plotted. It is recommended not to use more than 10000 data points. Another possible reason of this problem: an old computer system.

A.3.5 Why is it that, if I run "Spin & Brush" twice or more times, some items appear colored?

This is the effect of the memory function. If you run "Spin & Brush" once, `R` will remember your settings including the colors and use these settings the next time you start "Spin & Brush" with the same dataset.

A.3.6 May I change the three selected variables, when "Spin & Brush" is already running?

No, you have to restart "Spin & Brush" and select the variables you want.

A.3.7 May I change the selected item type, when "Spin & Brush" is already running?

Yes, this is possible. Click the "Change Item Type" button on the left hand side of the window and select the item you want.

A.4 Multivariate Graphics

A.4.1 What are the X and Y coordinate combo boxes for?

"Multivariate Graphics" is intended to display multivariate data which are linked to geographical coordinates. If a dataset contains coordinates they should be selected in the corresponding combo boxes.

A.4.2 When pressing the OK button of the "Multivariate Graphics" dialog, nothing happens. What's wrong?

Did you select variables? You have to select them by clicking the >> button.

A.4.3 Why is it that, if I run "Multivariate Graphics" twice or more times, some variables are already selected?

This is the effect of the memory function. If you run "Multivariate Graphics" once, `R` will remember your settings including the selected variables and use these settings the next time you start "Multivariate Graphics" with the same dataset.

A.4.4 Why is the legend always placed in the upper right corner of the window?

"Multivariate Graphics" was optimized for visualizing datasets from the KOLA Project (see Section 3.2). As one can see in the figures in Chapter 3, the choice of the upper right corner is convenient.

Appendix B

R Help Files

B.1 Spin & Brush

```
Description
        Visualize 3d data

Usage
        spin(x, y=0, z=0, groups=NULL)

Arguments
x       Matrix or data.frame containing data. When x is
        one-dimensional, y and z have to be supplied.
y       Optional vector argument, when x is a vector
z       Optional vector argument, when x is a vector
groups  Group factor; has to be between 0 and 6

Details
        Spin and Brush is a three-dimensional data viewing
        program which offers the possibility to rotate and
        zoom. Data points can be clustered by using different
        colors and identified by name or selection.

Value
        spin returns a data.frame consisting of the
        three-dimensional data and an additional group factor
        depending on the brushed data points.

Examples
        #data(swiss)
        #spin(swiss)
```

B.2 Multivariate Graphics

```
Description
        Draw multivariate Graphics

Usage
        multiGraph(x)

Arguments
x       Matrix or data.frame

Details
        Multivariate Graphics was made for datasets which
        include geographical coordinates. Data points are
        drawn as "multivariate items" on a geographical
        background map (the locations of the items are
        specified by their coordinates) which makes it
        possible to show the values of many variables at a
        time. Several multivariate items such as stars,
        trees or castles are provided.

Value
        If multiGraph is started via DASplusR, a string,
        representing the command, is returned.

Examples
        #data(swiss)
        #multiGraph(swiss)
```

Appendix C

R Scripts

C.1 Figure 2.1

```
par(mfrow=c(2,3))

plot(0, 0, type="n", xlab="Starting position", ylab="",
     xlim=c(-3,3), ylim=c(-3,3))
abline(h=0)
abline(v=0)

plot(0, 0, type="n", xlab="Translate system by (1, 1)", ylab="",
     xlim=c(-3,3), ylim=c(-3,3))
abline(h=0, col="grey")
abline(v=0, col="grey")
abline(h=1)
abline(v=1)

plot(1, 1, xlab="Drawing circle at new origin", ylab="",
     xlim=c(-3,3), ylim=c(-3,3))
abline(h=1)
abline(v=1)

plot(1, 1, xlab="Resetting coordinate system", ylab="",
     xlim=c(-3,3), ylim=c(-3,3))
abline(h=0)
abline(v=0)

plot(-2, 2, xlab="Translating system by (-2, 2) and drawing
     at new origin", ylab="", xlim=c(-3,3), ylim=c(-3,3))
points(1, 1)
```

```
abline(h=0, col="grey")
abline(v=0, col="grey")
abline(v=-2)
abline(h=2)

plot(-2, 2, xlab="Resetting coordinate system", ylab="",
     xlim=c(-3,3), ylim=c(-3,3))
points(1, 1)
abline(h=0)
abline(v=0)
```

C.2 Figures 3.3 - 3.8

```
require(mvgraph)
require(StatDA)
data(kola.background)
data(moss)
myData <- moss[1:40,]

pdf("stars.pdf")
plot(moss$XCOO,moss$YCOO, frame.plot=FALSE,
     xaxt="n",yaxt="n",xlab="",ylab="",type="n")
plotbg(map.col=c("gray","gray","gray","gray"),add.plot=T)
multiGraph(myData)
dev.off()


pdf("segments.pdf")
plot(moss$XCOO,moss$YCOO, frame.plot=FALSE,
     xaxt="n",yaxt="n",xlab="",ylab="",type="n")
plotbg(map.col=c("gray","gray","gray","gray"),add.plot=T)
multiGraph(myData)
dev.off()

pdf("boxes.pdf")
plot(moss$XCOO,moss$YCOO, frame.plot=FALSE,
     xaxt="n",yaxt="n",xlab="",ylab="",type="n")
plotbg(map.col=c("gray","gray","gray","gray"),add.plot=T)
multiGraph(myData)
dev.off()

pdf("polys.pdf")
plot(moss$XCOO,moss$YCOO, frame.plot=FALSE,
     xaxt="n",yaxt="n",xlab="",ylab="",type="n")
plotbg(map.col=c("gray","gray","gray","gray"),add.plot=T)
multiGraph(myData)
dev.off()

pdf("trees.pdf")
plot(moss$XCOO,moss$YCOO, frame.plot=FALSE,
     xaxt="n",yaxt="n",xlab="",ylab="",type="n")
plotbg(map.col=c("gray","gray","gray","gray"),add.plot=T)
multiGraph(myData)
dev.off()
```

```
pdf("castles.pdf")
plot(moss$XCOO,moss$YCOO, frame.plot=FALSE,
     xaxt="n",yaxt="n",xlab="",ylab="",type="n")
plotbg(map.col=c("gray","gray","gray","gray"),add.plot=T)
multiGraph(myData)
dev.off()
```

Bibliography

Jasmin Blanchette and Mark Summerfield. *C++ GUI Programmierung mit Qt 4: Die offizielle Einfuehrung*. Addison-Wesley, 2008a.

Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt4 (Prentice Hall Open Source Software Development)*. Prentice-Hall, 2008b.

Olaf Borkner-Delcarlo. *GUI-Programmierung mit Qt*. Hanser, 2002.

Diane Cook and Deborah Swayne. *Interactive and Dynamic Graphics for Data Analysis*. Springer, 2007.

Michael J. Crawley. *The R Book*. Wiley & Sons, 2007.

Michael J. Crawley. *Statistics: An Introduction Using R*. Wiley & Sons, 2005.

Peter Dalgaard. *Introductory Statistics with R*. Springer, 2008.

Matthias K. Dalheimer and Jesper Pedersen et al. *Practical Qt*. Dpunkt, 2004.

Brian S. Everitt and Hothorn Torsten. *A Handbook of Statistical Analyses Using R*. Crc Pr Inc, 2006.

Alan Ezust and Paul Ezust. *An Introduction to Design Patterns in C++ with Qt 4*. Prentice-Hall, 2006.

Hans Havlicek. *Lineare Algebra 1*, 2003a.

Hans Havlicek. *Lineare Algebra 2*, 2003b.

Friedrich Leisch. Tutorial on creating R packages. 2008.

John Maindonald and John Braun. *Data Analysis and Graphics Using R: An Example-based Approach*. Cambridge University Press, 2006.

Paul Martz. *OpenGL Distilled*. Addison-Wesley Longman, 2006.

Daniel Molkentin. *The Book of Qt 4: The Art of Building Qt Applications*. No Starch Press, 2007.

Paul Murrell. *R Graphics*. Chapman & Hall, 2005.

Dieter Orlamuender and Wilfried Mascolus. *Computergrafik und OpenGL: Eine systematische Einfuehrung*. Hanser, 2004.

Clemens Reimann, Peter Filzmoser, R. G. Garrett, and Rudolf Dutter. *Statistical Data Analysis Explained. Applied Environmental Statistics with R*. Wiley, 2008.

Dave Shreiner. *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.4*. Addison-Wesley Longman, 2004.

Dave Shreiner, Mason Woo, and Jackie Neider. *OpenGL Library. Boxed Set*. Addison-Wesley Longman, 2006.

Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2*. Addison-Wesley Longman, 2007.

Bjarne Stroustrup. *The C++ Programming Language: Special Edition*. Addison-Wesley Longman, 2000.

R Development Core Team. *R Data Import/Export*, 2008a.

R Development Core Team. *Writing R Extensions*, 2008b.

R Development Core Team. *R Installation and Administration*, 2008c.

R Development Core Team. *R Internals*, 2008d.

R Development Core Team. *An Introduction to R*, 2008e.

R Development Core Team. *The R Language Definition*, 2008f.

R Development Core Team. *The R Reference Index*, 2008g.

Juergen Wolf. *C von A bis Z*. Galileo Press, 2006a.

Juergen Wolf. *C++ von A bis Z: Das umfassende Handbuch*. Galileo Press, 2006b.

Juergen Wolf. *Qt 4 - GUI-Entwicklung mit C++: Das umfassende Handbuch*. Galileo Press, 2007.

Richard S. Wright and Michael R. Sweet. *OpenGL Super Bible*. Sams, 2004.