Computation of Correspondence Analysis

B

In this appendix the computation of CA is illustrated using the object-oriented computing language R, which can be freely downloaded from the website:

`http://www.r-project.org`

We assume here that the reader has some basic knowledge of this language, which has become the *de facto* standard for statistical computing. If not, the above website gives many resources for learning it. The scripts which are given in this appendix are available at the website of the CARME network:

`http://www.carme-n.org`

(CARME = Correspondence Analysis and Related MEthods). At the end of this appendix we shall also comment on commercially available software, and describe different graphical options for producing maps.

Contents

The R *program*

The R program provides all the tools necessary to produce CA maps, the most important one being the singular value decomposition (SVD). These tools are encapsulated in R *functions*, and several functions and related material can be gathered together to form an R *package*. An R package called **ca** is already available for doing every type of CA described in this book, to be demonstrated later in this appendix. But before that, we show step-by-step how to perform various computations using R. The three-dimensional graphics package **rgl** will also be demonstrated in the process. In the following we use a Courier font for all R instructions and R output; for example, here we create the matrix (13.2) on page 99, calculate its SVD and store it in an R "svd" object, and then ask for the part of the object labelled '`d`' (the singular values):

```
> table.T   <- matrix(c(8,5,-2,2,4,2,0,-3,3,6,2,3,3,-3,-6,
+                    -6,-4,1,-1,-2),nrow=5)
> table.SVD <- svd(table.T)
> table.SVD$d
[1] 1.412505e+01 9.822577e+00 1.376116e-15 7.435554e-32
```

The commands are indicated in slanted script (the prompt `>` is not typed), while the results are given in regular typewriter script. A `+` at the start of the line indicates continuation of the command.

Entering data into R

Entering data into R has its peculiarities, but once you have managed to do it, the rest is easy! The `read.table()` function is one of the most useful ways to input data matrices, and the easiest data sources are a text file or an Excel file. For example, suppose we want to input the 5×3 data table on readership given in Exhibit 3.1. Here are three options for reading it in.

1. Suppose the data are in a text file as follows:

```
      C1   C2   C3
E1     5    7    2
E2    18   46   20
E3    19   29   39
E4    12   40   49
E5     3    7   16
```

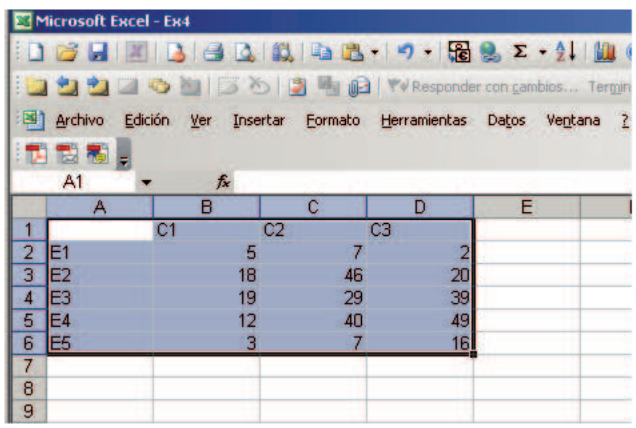
   and suppose the file is called `reader.txt` and stored in the present R working directory. Then execute the following statement in R:

```
> read.table("reader.txt")
```

2. An easier alternative is to copy the above file into the clipboard by selecting the contents in the text- or word-processor and copying using the pull-down Edit menu or right-clicking the mouse (assuming Windows platform). Then execute a similar command by reading directly from the clipboard:

```
> read.table("clipboard")
```

3. In a similar fashion, data can be read from an Excel file* via the clipboard, assuming the data are in an Excel file, as displayed below:



   The cells of this table have been selected and then copied. The command

```
> table <- read.table("clipboard")
```

   results in the table being stored as an R "data frame" object with the name

* Using the R package **foreign**, distributed with the program, it is possible to read other data formats, e.g., Stata, Minitab, SPSS, SAS, Systat and DBF.

`table`. Notice that the success of this `read.table()` command relies on the fact that the first line of the copied table contains one less entity than the other lines — this is why there is an empty cell in the top left-hand corner of the Excel table, similarly in the text file. If the `read.table()` function finds one less entity in the first line, it realizes that the first line consists of column labels and the subsequent lines have the row labels in the first column. The contents of `table` can be seen by entering

```
> table
   C1 C2 C3
E1  5  7  2
E2 18 46 20
E3 19 29 39
E4 12 40 49
E5  3  7 16
```

The object includes the row and column names, which can be accessed by typing `rownames(table)` and `colnames(table)`, for example:

```
> rownames(table)
[1] "E1" "E2" "E3" "E4" "E5"
```

R scripts for each chapter

We now describe systematically the computations for each chapter, starting with Chapter 2. Only basic R functions and the three-dimensional plotting package **rgl**† will be used, leaving till later a demonstration of the **ca** package which does the calculations in a much more compact way.

Chapter 2: Profiles and the Profile Space

In Chapter 2 we showed some triangular plots of the travel data set. Suppose that the profile data of Exhibit 2.1 are input as described before and stored in the data frame `profiles` in R; that is, after copying the profile data:

```
> profiles <- read.table("clipboard")
> profiles
               Holidays HalfDays FullDays
Norway            0.333    0.056    0.611
Canada            0.067    0.200    0.733
Greece            0.138    0.862    0.000
France/Germany    0.083    0.083    0.833
```

*Example of three-dimensional graphics using **rgl** package*

(notice that the column names had been originally written without blanks, otherwise the data would not have been read correctly). We can do a three-dimensional view of the profiles using the **rgl** package as follows (assuming **rgl** has been installed and loaded — see the footnote below).

```
> rgl.lines(c(0,1.2), c(0,0), c(0,0))
> rgl.lines(c(0,0), c(0,1.2), c(0,0))
> rgl.lines(c(0,0), c(0,0), c(0,1.2))
> rgl.lines(c(0,0), c(0,1), c(1,0), size=2)
```

† The **rgl** package is not one of the packages provided as standard with R, but needs to be installed by downloading it from the R website or `www.carme-n.org`.

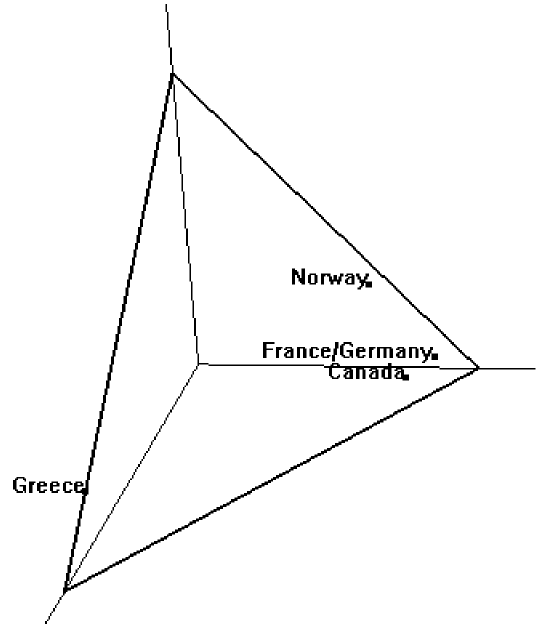


Exhibit B.1: *Three-dimensional view of the country row profiles of the travel data set, using the* R *package* **rgl**.

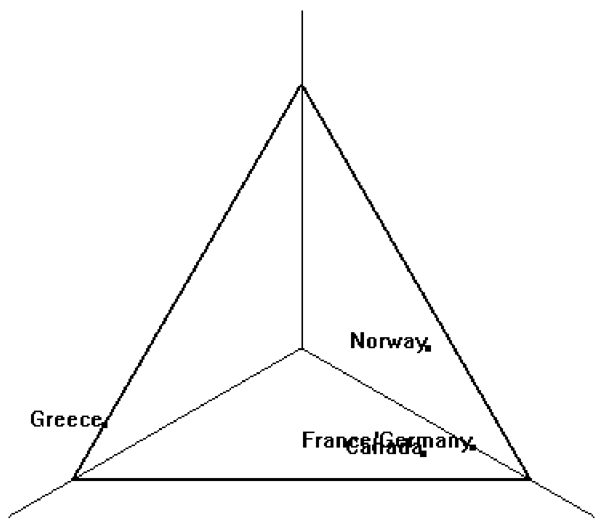


Exhibit B.2: *Rotation of the three-dimensional space to show the triangle in which the profile points lie.*

```
> rgl.lines(c(0,1), c(1,0), c(0,0), size=2)
> rgl.lines(c(0,1), c(0,0), c(1,0), size=2)
> rgl.points(profiles[,3],profiles[,1],profiles[,2],
+            size=4)
> rgl.texts(profiles[,3],profiles[,1],profiles[,2],
+           text=row.names(profiles))
```

The 3-D scatterplot from a certain viewpoint is shown in Figure B.1. Using the mouse while pressing the left button, this figure can be rotated to give a realistic three-dimensional feeling. Figure B.2 shows one of these rotations where the viewpoint is flat onto the triangle that contains the profile points. The mouse wheel allows zooming into the display.

Chapter 3: Masses and Centroid

As an illustration of the graphics in Chapter 3, we give the code to draw the triangular coordinate plot in Exhibit 3.2 using R, which needs some trigonometry to calculate the (x, y) positions of each point. Assuming the table has been read as on the bottom of page 214 into the data frame `table`, the following commands in R produce the figure in Exhibit B.3. The first statement calculates the row profiles in `table.pro` using the `apply()` function — this function can operate on rows or columns; here the parameter "`1`" indicates rows, and "`sum`" the operation required. R stores matrices as a string of columns, so the division of `table` by the row sums does just the right thing, dividing the first column by the row sums, then the second and so on. The two subsequent statements calculate the x and y coordinates of the five profiles in an equilateral triangle of side 1, using the first and third profile values (only two out of the three are needed to situate the point).

`apply()` *function*

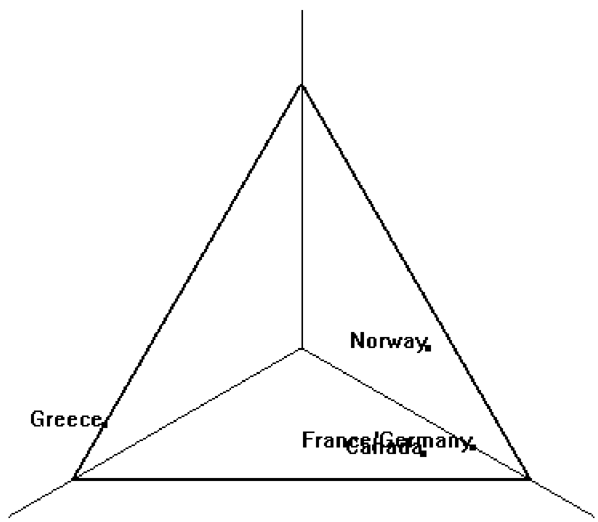


Exhibit B.3: *Plot of five education group profiles in the triangular coordinate space.*

Example of two-dimensional graphics

```
> table.pro <- table / apply(table, 1, sum)
> table.x <- 1 - table.pro[,1] - table.pro[,3] / 2
> table.y <- table.pro[,3] * sqrt(3) / 2
> plot.new()
> lines(c(0,1,0.5,0), c(0,0,sqrt(3)/2,0), col="gray")
> text(c(0,1,0.5), c(0,0,sqrt(3)/2), labels=colnames(table))
> text(table.x, table.y, labels=rownames(table))
```

Chapter 4: Chi-square Distances and Inertia

In Chapter 4 the χ^2 statistic, inertia and χ^2-distances were calculated, each of which is illustrated here. The calculations are performed on the readership data frame `table` used previously.

— *χ^2 statistic and total inertia:*

```
> table.rowsum <- apply(table, 1, sum)
> table.colsum <- apply(table, 2, sum)
> table.sum <- sum(table)
> table.exp <- table.rowsum %o% table.colsum / table.sum
> chi2 <- sum((table - table.exp)^2 / table.exp)
> chi2
[1] 25.97724
> chi2 / table.sum
[1] 0.08326039
```

Outer product operator `%o%`

Notice the use of the *outer product* operator `%o%` in the fourth command above; this multiplies every element in the vector to the left of the operator with every element in the vector on the right.

— *χ^2-distances of row profiles to centroid:*

We first show the least elegant (but most obvious) way of calculating the square of the χ^2-distance for the fifth row of the table, as shown in brackets in (4.4). A `for` loop in R is used to build up the sum of the three terms:

Example of `for` *loop in* R

```
> chidist <- 0
> for(j in 1:3) {
+  chidist<-chidist+
+  (table.pro[5,j] - table.colmass[j])^2 / table.colmass[j]
+  }
> chidist
       C1
0.1859165
```

The label `C1` is given to the value of `chidist` probably because this is the first column of the loop. A more elegant way is to compute all five distances in one step. We need to subtract the row of column masses from each row of the profile matrix, square these differences, and then divide each row again by the column masses, finally adding up the rows. Row operations are slightly more difficult in R because matrices are stored as column vectors, so one solution is to transpose the profile matrix first, using the `t()` transpose function. Then all the columns of the transposed object (previously rows) are summed, using the `apply()` function with parameters `2,sum` indicating column sums:

Transpose function `t()`

```
> apply((t(table.pro)-table.colmass)^2 / table.colmass,2,sum)
        E1         E2         E3         E4         E5
0.35335967 0.11702343 0.02739229 0.03943842 0.18591649
```

Distance function `dist()`

Finally, all χ^2-distances can be computed, between all profiles and in particular between the profiles and their average, using the `dist()` function, which by default computes a Euclidean distance matrix between rows of a matrix.

First, the row of column masses (average row profile) is appended to the profile matrix using the function `rbind()` (row binding) to form a 6×3 profile matrix `tablec.pro`. Second, we need to divide each profile element by the corresponding square root of the average. An alternative to having to use the transpose operation again is to use the versatile `sweep()` function, which acts like `apply()` but has more options. In the second command below the options of `sweep` are 2 (operating down the columns), `sqrt(table.colmass)` (the vector used for the operation) and `"/"` (the operation is division):

Row binding using `rbind()`

Versatile `sweep()` *function*

```
> tablec.pro <- rbind(table.pro, table.colmass)
> rownames(tablec.pro)[6]<-"ave"
> dist(sweep(tablec.pro, 2, sqrt(table.colmass), FUN="/"))
           E1        E2        E3        E4        E5
E2  0.3737004
E3  0.6352512 0.4696153
E4  0.7919425 0.5065568 0.2591401
E5  1.0008054 0.7703644 0.3703568 0.2845283
ave 0.5944406 0.3420869 0.1655062 0.1985911 0.4311803
```

The last line, which was labelled "`ave`" for the appended average profile (see second command above), gives the χ^2-distances (square roots of the squared values calculated above). All other distances between the five row profiles are also given, and the result of `dist()` is an R distance object which stores only the triangular matrix of distances.

In Chapter 5 the χ^2-distances are visualized by stretching the coordinate axes by amounts inversely proportional to the square roots of the corresponding masses. So this is a similar sequence of code to that given previously for the three-dimensional plot of Chapter 2, except that each coordinate is divided by `sqrt(table.colmass)`. The trickier aspect is to decide which profile elements go with which dimensions, to reproduce Exhibit 5.2 — we leave this as a small exercize for the reader, but the actual script is given on the website.

Chapter 5: Plotting Chi-square Distances

Chapter 6 starts with actual CAs in that dimension-reduction is involved, so here we shall perform our first singular value decomposition (SVD). We first input the health self-assessment data set and call it `health`; then we follow the steps given on page 202 of the Theoretical Appendix. The preparatory steps (A.1–3) are as follows:

Chapter 6: Reduction of Dimensionality

```
> health.P <- health / sum(health)
> health.r <- apply(health.P, 1, sum)
> health.c <- apply(health.P, 2, sum)
> health.Dr <- diag(health.r)
> health.Dc<-diag(health.c)
> health.Drmh <- diag(1/sqrt(health.r))
> health.Dcmh <- diag(1/sqrt(health.c))
```

Diagonal matrix function `diag()`

The last two commands above create $\mathbf{D}_r^{-\frac{1}{2}}$ and $\mathbf{D}_c^{-\frac{1}{2}}$, respectively, since we need them repeatedly later (in the object name `mh` stands for "minus half").

Matrix multiplication operator %*%

In order to perform the matrix multiplication in (A.4), the data frame `health.P` has to be converted to a regular matrix, and then the matrix multiplication is performed using the operator `%*%`, finally the SVD in (A.5) using function `svd()`:

```
> health.P <- as.matrix(health.P)
> health.S <- health.Drmh %*% (health.P - health.r %o% health.c)
+                            %*% health.Dcmh
> health.svd <- svd(health.S)
```

Example of SVD function `svd()`

The principal and standard coordinates (`pc` and `sc`) are calculated as in (A.6–9):

```
> health.rsc <- health.Drmh %*% health.svd$u
> health.csc <- health.Dcmh %*% health.svd$v
> health.rpc <- health.rsc %*% diag(health.svd$d)
> health.cpc <- health.csc %*% diag(health.svd$d)
```

And that's it! The previous 14 R commands are the whole basic CA algorithm — simply replace `health` with any other data frame to compute the point coordinates.

To see the values of, for example, the principal coordinates of the rows on the first principal axis:

```
> health.rpc[,1]
[1] -0.37107411 -0.32988430 -0.19895401  0.07091332  0.39551813  ...
```

(notice that the signs are reversed compared to the display of Exhibit 6.3 — this can often occur using different software, but the user can reverse the signs of all the coordinates on an axis at will).

Chapter 7: Optimal Scaling

Chapter 7 deals with optimal scaling properties of the CA solution, and there are no challenging calculations in this chapter. We illustrate the calculation of the transformed optimal scale values in (7.5) using R functions for calculating minimum and maximum values (again, because of the sign change in the coordinates on the first axis, the scale is reversed, in other words the transformed scale goes from 0=very good to 100=very bad; hence subtracting the results below from 100 will give the results of Exhibit 7.2):

```
> health.range <- max(health.csc[,1]) - min(health.csc[,1])
> health.scale <- (health.csc[,1] - min(health.csc[,1])) * 100 /
+                   health.range
> health.scale
[1]   0.00000  18.86467  72.42164  98.97005 100.00000
```

Chapter 8: Symmetry of Row and Column Analyses

Chapter 8 is another chapter demonstrating properties of the solution rather than making new calculations. Exhibit 8.5 was not constructed using R but was typeset directly in LaTeX (see descriptions of graphical typsetting at the end of this appendix). The maximal correlation properties of CA can be illustrated with some R commands, for example Equation (8.2) on page 63. The correlation between the age and health scale values on the first dimension is first calculated as $\phi^{\mathsf{T}}\mathbf{P}\gamma$ where ϕ and γ are the standard coordinates on the first dimension, and $\mathbf{P}$ is the correspondence matrix:

```
> health.cor <- t(health.rsc[,1]) %*% health.P %*% health.csc[,1]
> health.cor^2
          [,1]
[1,] 0.1366031
```

Thus the square of this correlation is the first principal inertia (the above result is given as a 1×1 matrix since it is the result of matrix-vector multiplications).

The following demonstrates the standardization in (A.12) for the rows, for example, which justifies the above way of calculating the correlation since the variances are 1:

```
> t(health.rsc[,1]) %*% health.Dr %*% health.rsc[,1]
     [,1]
[1,]    1
```

Chapter 9: Two-dimensional displays

Chapter 9 explains the geometry of two-dimensional maps and compares asymmetric and symmetric maps. The smoking data set is part of the **ca** package so perhaps this is a good time to make a first introduction to that package. Once the package is installed and loaded, these data can be called up simply by issuing the command:

Initial contact with **ca** *package*

```
> data(smoke)
```

which gives the data frame `smoke`:

```
> smoke
   none light medium heavy
SM    4     2      3     2
JM    4     3      7     4
SE   25    10     12     4
JE   18    24     33    13
SC   10     6      7     2
```

One of the functions in the **ca** package is `ca()` to perform simple CA. The CA of the smoking data is obtained easily by saying `ca(smoke)`:

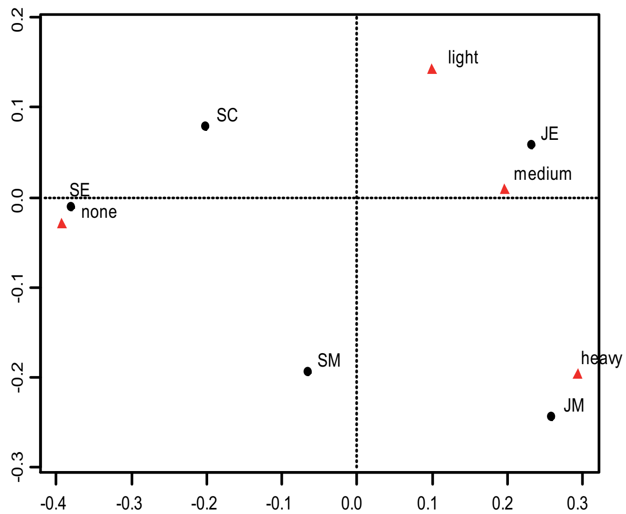
```
> ca(smoke)

 Principal inertias (eigenvalues):
           1        2        3
Value      0.074759 0.010017 0.000414
Percentage 87.76%   11.76%   0.49%


 Rows:
                SM        JM        SE       JE        SC
Mass      0.056995  0.093264  0.264249 0.455959  0.129534
ChiDist   0.216559  0.356921  0.380779 0.240025  0.216169
Inertia   0.002673  0.011881  0.038314 0.026269  0.006053
Dim. 1   -0.240539  0.947105 -1.391973 0.851989 -0.735456
Dim. 2   -1.935708 -2.430958 -0.106508 0.576944  0.788435
```

```
 Columns:
             none    light   medium     heavy
Mass     0.316062 0.233161 0.321244  0.129534
ChiDist  0.394490 0.173996 0.198127  0.355109
Inertia  0.049186 0.007059 0.012610  0.016335
Dim. 1  -1.438471 0.363746 0.718017  1.074445
Dim. 2  -0.304659 1.409433 0.073528 -1.975960
```

Several numerical results are listed which should be familiar: the principal inertias and their percentages, and then for each row and column the mass, χ^2-distance to the centroid, the inertia, and the standard coordinates on the first two dimensions. The features of this package will be described in much more detail later on, but just to show now how simple the plotting is, simply put the `plot()` function around `ca(smoke)` to get the default symmetric CA map shown in Exhibit B.4:

```
> plot(ca(smoke))
```

Exhibit B.4: *Symmetric map of the data set* `smoke`*, using the* **ca** *package.*



Notice that both principal axes have been inverted compared to the map of Exhibit 9.5. To obtain the asymmetric maps, add the option `map="rowprincipal"` or `map="colprincipal"` to the `plot()` function; for example, Exhibit 9.2 is obtained with the following command:

```
> plot(ca(smoke), map="rowprincipal")
```

Chapter 10: Three More Examples

With this short introduction to the **ca** package, the analyses of Chapter 10 will be easy to reproduce. The three data sets are available on the website `www.carme-n.org` in text and Excel formats for copying and reading into R. The author data set is also provided with the **ca** package so this can be obtained, like the smoking data, with the R command `data(author)`. To see the author data in a three-dimensional CA map, try this (again, assuming you have loaded the **ca** package):

```
> data(author)
> plot3d.ca(ca(author), labels=c(2,1), sf=0.000001)
```

Chapter 11: Contributions to Inertia

Chapter 11 involves a certain amount of new computations, all of which are actually part of the **ca** package, but here again we choose to demonstrate them first "by hand". The data set for the scientific research funding is read as described before — suppose the data frame is called `fund`. As in Chapter 4, the matrix of standardized residuals is calculated for this table, and then the inertias in Exhibit 11.1 are the sums-of-squares of the rows and columns:

```
> fund.P <-as.matrix(fund / sum(fund))
> fund.r <-apply(fund.P, 1, sum)
> fund.c <-apply(fund.P, 2, sum)
> fund.Drmh <-diag(1 / sqrt(fund.r))
> fund.Dcmh <-diag(1 / sqrt(fund.c))
> fund.res <-fund.Drmh %*% (fund.P - fund.r %o% fund.c) %*% fund.Dcmh
> round(apply(fund.res^2, 1, sum), 5)
 [1] 0.01135 0.00990 0.00172 0.01909 0.01621 0.01256 0.00083
 [8] 0.00552 0.00102 0.00466
> round(apply(fund.res^2, 2, sum), 5)
[1] 0.01551 0.00911 0.00778 0.02877 0.02171
```

Contributions of each cell of the table to the total inertia

The permill contributions in Exhibit 11.2 are the squared standardized residuals relative to the total:

```
> round(1000*fund.res^2 / sum(fund.res^2), 0)
      [,1] [,2] [,3] [,4] [,5]
 [1,]    0   32   16    0   89
 [2,]    0   23    4   44   48
 [3,]    3   12    1    0    5
 [4,]    9   15   11  189    8
 [5,]  106   11    2   74    3
 [6,]    1   11   38    1  102
 [7,]    2    0    0    3    5
 [8,]   51    4    0   10    2
 [9,]   10    0    0    2    0
[10,]    5    3   22   26    0
```

(the row and column labels have been lost because of the matrix multiplications, but can be restored again if necessary using `rownames()` and `colnames()` functions).

The principal inertias in Exhibit 11.3 are the squares of the singular values from the SVD of the residuals matrix:

```
> fund.svd <- svd(fund.res)
> fund.svd$d^2
[1] 3.911652e-02 3.038081e-02 1.086924e-02 2.512214e-03 3.793786e-33
```

(five values are given, but the fifth is theoretically an exact zero).

To calculate the individual components of inertia of the row points, say, on all four axes, we first need to calculate the principal coordinates f_{ik} (see (A.8)) and then the values of $r_i f_{ik}^2$:

```
> fund.F <- fund.Drmh %*% fund.svd$u %*% diag(fund.svd$d)
> fund.rowi <- diag(fund.r) %*% fund.F^2
> fund.rowi[,1:4]
              [,1]         [,2]         [,3]         [,4]
 [1,] 6.233139e-04 9.775878e-03 8.222230e-04 1.301601e-04
 [2,] 1.178980e-03 7.542243e-03 8.385857e-04 3.423076e-04
 [3,] 2.314352e-04 8.787604e-04 2.931994e-04 3.211261e-04
 [4,] 1.615600e-02 1.577160e-03 6.274587e-04 7.271264e-04
 [5,] 1.426048e-02 1.043783e-04 1.691831e-03 1.562740e-04
 [6,] 1.526183e-03 9.407586e-03 1.273528e-03 3.573707e-04
 [7,] 7.575664e-06 5.589276e-04 7.980532e-05 1.868385e-04
 [8,] 3.449918e-03 1.601539e-04 1.799425e-03 1.091335e-04
 [9,] 5.659639e-04 7.306881e-06 4.185906e-04 3.022249e-05
[10,] 1.116674e-03 3.684113e-04 3.024590e-03 1.516545e-04
```

which agrees with Exhibit 11.5. Notice in the last command above that only the first four columns are relevant (`fund.rowi[,1:4]`); there is a fifth column of tiny values which are theoretically zero because the fifth singular value is zero. Finally, to relativize these components with respect to the inertia of a point (row sums) or inertia of an axis (column sums, i.e., principal inertias) (see (A.27) and (A.26) respectively), and converting them to permills at the same time:

Calculating relative contributions (squared cosines or correlations)

```
> round(1000*(fund.rowi / apply(fund.rowi, 1, sum))[,1:4], 0)
    [,1] [,2] [,3] [,4]
 [1,]   55  861   72   11
 [2,]  119  762   85   35
 [3,]  134  510  170  186
 [4,]  846   83   33   38
 [5,]  880    6  104   10
 [6,]  121  749  101   28
 [7,]    9  671   96  224
 [8,]  625   29  326   20
 [9,]  554    7  410   30
[10,]  240   79  649   33
```

which agrees with Exhibit 11.6 (to obtain the qualities in Exhibit 11.8, add up the first two columns of the above table). With respect to column sums, i.e., principal inertias:

Calculating contributions to each principal axis

```
> round(1000*t(t(fund.rowi) / fund.svd$d^2)[,1:4], 0)
      [,1] [,2] [,3] [,4]
 [1,]   16  322   76   52
 [2,]   30  248   77  136
 [3,]    6   29   27  128
 [4,]  413   52   58  289
 [5,]  365    3  156   62
```

```
 [6,]   39  310  117  142
 [7,]    0   18    7   74
 [8,]   88    5  166   43
 [9,]   14    0   39   12
[10,]   29   12  278   60
```

which shows how each axis is constructed; for example, rows 4 and 5 (Physics and Zoology) are the major contributors to the first axis.

Anticipating the fuller description of the **ca** package later, we point out that the complete set of these numerical results can be obtained using the `summary()` function around `ca(fund)` as follows:

```
> summary(ca(fund))

Principal inertias (eigenvalues):

 dim    value     %     cum%  scree plot
 1      0.039117  47.2  47.2  *************************
 2      0.030381  36.7  83.9  *******************
 3      0.010869  13.1  97.0  ******
 4      0.002512   3.0 100.0
        -------- -----
 Total: 0.082879 100.0

Rows:
     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1  |  Gel |  107  916  137 |   76  55  16 |  303 861 322 |
2  |  Bic |   36  881  119 |  180 119  30 | -455 762 248 |
3  |  Chm |  163  644   21 |   38 134   6 |   73 510  29 |
4  |  Zol |  151  929  230 | -327 846 413 |  102  83  52 |
5  |  Phy |  143  886  196 |  316 880 365 |   27   6   3 |
6  |  Eng |  111  870  152 | -117 121  39 | -292 749 310 |
7  |  Mcr |   46  680   10 |   13   9   0 | -110 671  18 |
8  |  Bot |  108  654   67 | -179 625  88 |  -39  29   5 |
9  |  Stt |   36  561   12 |  125 554  14 |   14   7   0 |
10 |  Mth |   98  319   56 |  107 240  29 |  -61  79  12 |

Columns:
    name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1 |    A |   39  587  187 |  478 574 228 |   72  13   7 |
2 |    B |  161  816  110 |  127 286  67 |  173 531 159 |
3 |    C |  389  465   94 |   83 341  68 |   50 124  32 |
4 |    D |  162  968  347 | -390 859 632 |  139 109 103 |
5 |    E |  249  990  262 |  -32  12   6 | -292 978 699 |
```

Chapter 12: Supplementary Points

Chapter 12 shows how to add points to an existing map, using the barycentric relationship between standard coordinates of the column points, say, and the principal coordinates of the row points; i.e., profiles lie at weighted averages of vertices. The example at the top of page 94 shows how to situate the supplementary point *Museums* which has data [4 12 11 19 7] summing up to 53. Calculating the profile, say the vector $\mathbf{m}$ and then its scalar products with the standard column coordinates: $\mathbf{m}^\mathsf{T}\mathbf{\Gamma}$ gives its coordinates in the map:

```
> fund.m <- c(4,12,11,19,7)/53
> fund.Gamma <- fund.Dcmh%*%fund.svd$v
> t(fund.m) %*% fund.Gamma[,1:2]
           [,1]      [,2]
[1,] -0.3143203 0.3809511
```

(the sign of the second axis is reversed in this solution compared to Exhibit 12.2). It is clear that if we perform the same operation with the unit vectors of Exhibit 12.4 as supplementary points, then multiplying these with the standard coordinates is just the same as the standard coordinates.

Chapter 13: Correspondence Analysis Biplots

In Chapter 13 the different scaling of a CA map are discussed from the point of view of the biplot. In the standard CA biplot of Exhibit 13.3 the rows are in principal coordinates while the columns are in rescaled standard coordinates where each column point has been pulled in by multiplying its coordinates by the square root of the column mass. Given the standard coordinates `fund.Gamma` calculated above, these rescaled coordinates on the first two dimensions are calculated as:

```
> diag(sqrt(fund.c)) %*% fund.Gamma[,1:2]
            [,1]        [,2]
[1,]  0.47707276  0.08183444
[2,]  0.25800640  0.39890356
[3,]  0.26032157  0.17838093
[4,] -0.79472740  0.32170520
[5,] -0.08046934 -0.83598151
```

In the following commands, the scalar products on the right-hand side of (13.7), for $K^* = 2$, are first stored in `fund.est` and then estimated profiles are calculated by multiplying by the square roots $\sqrt{c_j}$ and adding c_j, all using matrix algebra:

```
> fund.est <- fund.F[,1:2] %*% t(diag(sqrt(fund.c)) %*%
+              fund.Gamma[,1:2])
> oner <- rep(1,dim(fund)[1])
> round(fund.est %*% diag(sqrt(fund.c)) + oner %o% fund.c, 3)
         A     B     C     D     E
 [1,] 0.051 0.217 0.436 0.177 0.120
 [2,] 0.049 0.107 0.368 0.046 0.431
 [3,] 0.044 0.176 0.404 0.160 0.217
 [4,] 0.010 0.143 0.348 0.280 0.219
 [5,] 0.069 0.198 0.444 0.065 0.225
 [6,] 0.023 0.102 0.338 0.162 0.375
 [7,] 0.038 0.145 0.379 0.144 0.294
 [8,] 0.021 0.136 0.356 0.214 0.272
 [9,] 0.051 0.176 0.411 0.124 0.238
[10,] 0.048 0.162 0.400 0.120 0.270
```

This can be compared with the true profile values:

```
> round(fund.P/fund.r, 3)
```

```
         A     B     C     D     E
Geol 0.035 0.224 0.459 0.165 0.118
Bioc 0.034 0.069 0.448 0.034 0.414
Chem 0.046 0.192 0.377 0.162 0.223
Zool 0.025 0.125 0.342 0.292 0.217
Phys 0.088 0.193 0.412 0.079 0.228
Engi 0.034 0.125 0.284 0.170 0.386
Micr 0.027 0.162 0.378 0.135 0.297
Bota 0.000 0.140 0.395 0.198 0.267
Stat 0.069 0.172 0.379 0.138 0.241
Math 0.026 0.141 0.474 0.103 0.256
```

Calculating the differences between the true and estimated profile values gives the individual errors of approximation, and the sum of squares of these differences, suitably weighted, gives the overall error in the two-dimensional CA. Each row of squared differences has to be weighted by the corresponding row mass r_i and each column by the inverse of the expected value $1/c_j$. The calculation is the following (this is one command, wrapped over two lines here, a concentrated example in R programming!):

```
> sum(diag(fund.r) %*% (fund.est%*%diag(sqrt(fund.c))+
+  oner %o% fund.c - fund.P / fund.r)^2 %*% diag(1/fund.c))
[1] 0.01338145
```

To demonstrate that this is correct, add the principal inertias *not* on the first two axes:

```
> sum(fund.svd$d[3:4]^2)
[1] 0.01338145
```

which confirms the previous calculation (this is the 16% unexplained inertia reported at the bottom of page 103).

Biplot axis calibration

The calculation of the biplot calibrations is quite intricate since it involves a lot of trigonometry. Rather than list the whole procedure here, the interested reader is referred to the website where the script for the function `biplot.ca` is given and which calculates the coordinates of the starting and ending points of all the tic marks on the biplot axes for the columns.

Chapter 14: Transition and Regression Relationships

In Chapter 14 various linear relationships between row and column coordinates and the data are given. Here we shall demonstrate some of these using R's linear modelling function `lm()` which allows weights to be specified in the least-squares regression. For example, let's perform the weighted least-squares regression of the standard row coordinates (y-axis in Exhibit 14.2) on the column standard coordinates (x-axis). The variables of the regression have 10×5 values, and these will be vectorized in columns corresponding to the original matrix. Thus the x variable is the vector (called `fund.vecc` below) where the first column coordinate on the first dimension is repeated 10 times, then the second coordinate 10 times and so on, whereas the y variable (`fund.vecr`) has the set of first dimension's row coordinates repeated five times in a column (the row standard coordinates are calculated as `fund.Phi` below). Check the values of fund.vecc and fund.vecr below as you perform the computations. The weights of the regression will be the frequencies in the original table fund — to vectorize these, the data frame has to be first converted to a matrix and then to a vector using as.vector():

Conversion of data objects using as.matrix() *and* as.vector()

```
> fund.vec <- as.vector(as.matrix(fund))
> fund.Phi <- fund.Drmh %*% fund.svd$u
> fund.vecr <- rep(fund.Phi[,1], 5)
> fund.vecc <- as.vector(oner %*% t(fund.Gamma[,1]))
```

The weighted least-squares regression is then performed as follows:

Example of lm() *function for linear regression, using* weights *option*

```
> lm(fund.vecr~fund.vecc, weight = fund.vec)

Call:
lm(formula = fund.vecr ~ fund.vecc, weights=fund.vec)

Coefficients:
(Intercept)    fund.vecc
 -4.906e-16    1.978e-01
```

showing that the constant is zero and the coefficient is 0.1978, the square root of the first principal inertia.

To perform the regression described on page 110 between Geology's contingency ratios and the standard coordinates on the first two dimensions, the response fund.y is regressed on the first two columns of the standard coordinate matrix $\mathbf{\Gamma}$ in fund.Gamma, with weights $\mathbf{c}$ in fund.c, as follows (here the summary() function is used around the lm command to get more results):

```
> fund.y <- (fund.P[1,] / fund.r[1]) / fund.c
> summary(lm(fund.y ~ fund.Gamma[,1] + fund.Gamma[,2],
+              weights=fund.c))

Call:
lm(formula = fund.y ~ fund.Gamma[, 1]+fund.Gamma[, 2], weights=fund.c)

Residuals:
        A         B         C         D         E
-0.079708  0.016013  0.037308 -0.030048 -0.003764

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      1.00000    0.06678  14.975  0.00443 **
fund.Gamma[, 1]  0.07640    0.06678   1.144  0.37105
fund.Gamma[, 2]  0.30257    0.06678   4.531  0.04542 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.06678 on 2 degrees of freedom
Multiple R-Squared: 0.9161,     Adjusted R-squared: 0.8322
F-statistic: 10.92 on 2 and 2 DF,  p-value: 0.0839
```

confirming the coefficients at the bottom of page 110 (again, the second coefficient has reversed sign because the second dimension coordinates are reversed) and the R^2 of 0.916.

The `lm()` function does not give standardized regression coefficients, but these can be obtained by calculating weighted correlations using the weighted covariance function `cov.wt()` with option `cor=TRUE`:

Example of `cov.wt` *function to calculate weighted correlation*

```
> cov.wt(cbind(fund.y,fund.Gamma[,1:2]), wt=fund.c, cor=TRUE)$cor
$cor
          [,1]         [,2]         [,3]
[1,] 1.0000000 2.343286e-01 9.280040e-01
[2,] 0.2343286 1.000000e+00 2.359224e-16
[3,] 0.9280040 2.359224e-16 1.000000e+00
```

which agrees with the correlation matrix at the top of page 111, apart from possible sign changes.

Chapter 15: Clustering the Rows and Columns

Chapter 15 deals with Ward clustering of the row or column profiles using Ward clustering, weighting the profiles by their masses. The R function for performing hierarchical clustering is `hclust()`, which does not allow differential weights in the option for Ward clustering (see (15.2)); neither does the function `agnes()` in the package **cluster**. The commercial statistical package XLSTAT, described later, does have this possibility. In addition, Fionn Murtagh's R programs also include Ward clustering with weights (see page 253).

Chapter 16: Multiway Tables

In Chapter 16 the interactive coding of variables was described. To be able to code the data in this way, either the multiway table is needed or the original data. For example, in the case of the health data used in Chapter 16, the raw data looks like this, showing the first four rows of data out of 6371):

```
. . .   health   age   gender  . . .
. . .     4       5       2    . . .
. . .     2       3       1    . . .
. . .     2       4       1    . . .
. . .     3       5       1    . . .
. . .     .       .       .    . . .
. . .     .       .       .    . . .
```

To obtain Exhibit 16.2, the seven categories of `age` and the two categories of `gender` need to be combined into one variable `age_gender` with 14 categories. This is achieved with a simple transformation such as:

Interactive coding

```
> age_gender <- 7 * (gender - 1) + age
```

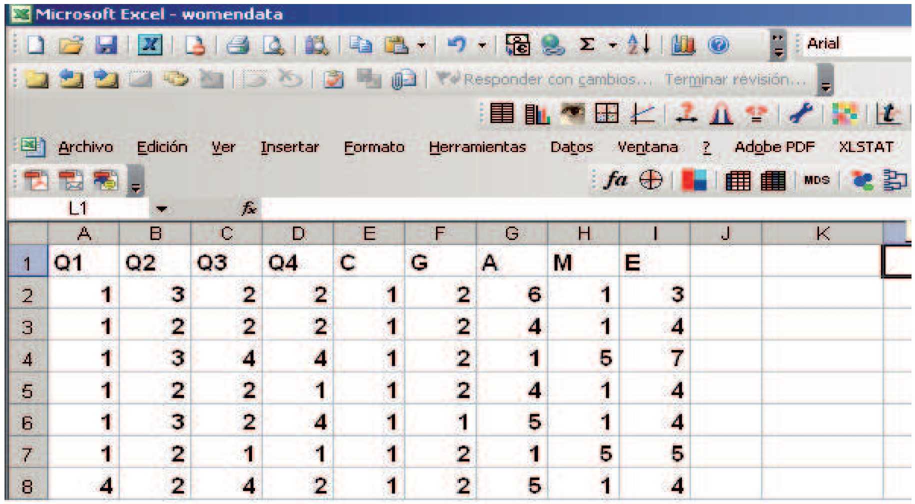
which will make the age groups of male (`gender=1`) numbered 1 to 7, and those of female (`gender=2`) numbered 8 to 14. From then on, everything continues as before, with a cross-tabulation being made of the variable `age_gender` with health. Cross-tabulations in R are made with the function `table()`; for example,

Cross-tabulations with `table()`

```
> table(age_gender, health)
```

would give the cross-tabulation in Exhibit 16.2.

Now suppose the raw data from the data set on working women is in an Excel file as shown below: four questions from `Q1` to `Q4`, country (`C`), gender (`G`), age (`A`), marital status (`M`) and education (`E`). To input the data into R, copy the



	A	B	C	D	E	F	G	H	I	J	K
1	Q1	Q2	Q3	Q4	C	G	A	M	E		
2	1	3	2	2	1	2	6	1	3		
3	1	2	2	2	1	2	4	1	4		
4	1	3	4	4	1	2	1	5	7		
5	1	2	2	1	1	2	4	1	4		
6	1	3	2	4	1	1	5	1	4		
7	1	2	1	1	1	2	1	5	5		
8	4	2	4	2	1	2	5	1	4		

columns to the clipboard as before, using function `read.table()`. But now the table does not have row names, and so no blank in the top left-hand cell; hence the option `header=T` needs to be specified (`T` is short for `TRUE`):

```
> women <- read.table("clipboard", header=T)
```

The column names of data frame `women` are obtained using function `colnames`:

```
> colnames(women)
 [1] "Q1" "Q2" "Q3" "Q4" "C"  "G"  "A"  "M"  "E"
```

Example of `attach()` *function*

In order to obtain the table in Exhibit 16.4, it is convenient to use the `attach()` function, which allows all the column names listed above to be available as if they were regular object names (to make these names unavailable the inverse operation `detach()` should be used):

```
> attach(women)
> table(C, Q3)
    Q3
C       1    2    3    4
   1  256 1156  176  191
   2  101 1394  581  248
   3  278  691   62   66
   4  161  646   70  107
   .    .    .    .    .
   .    .    .    .    .
  21  243  448  484   25
  22  468  664   92   63
  23  203  671  313  120
  24  738 1012  514  230
```

(cf. Exhibit 16.4).

To get the interactively coded row variable in Exhibit 16.6 and the table itself:

```
> CG <- 2 * (C - 1) + G
> table(CG, Q3)
    Q3
CG     1   2   3   4
  1  117 596 114  82
  2  138 559  60 109
  3   43 675 357 123
  4   58 719 224 125
  .    .   .   .   .
  .    .   .   .   .
  47 348 445 294 112
  48 390 566 218 118
  51   1   2   0   0
  55   1   1   2   1
```

Notice that the last two rows of the table correspond to a few missing values for gender that were coded as `9`; to see frequency counts for each column, enter the command `lapply(women,table)`. So we should remove all the missing data first — see page 237 how to remove cases with missing values. Alternatively, missing values can be assigned R's missing value code NA, for example the missing values for gender in column 6:

```
> women[,6][G==9]<-NA
> attach(women)
> CG <- 2 * (C - 1) + G
```

(notice that data frame `women` has to be attached again and `CG` recomputed). Assuming that all missing values have been recoded (or cases removed), the combinations of `CG` and `A` are coded as follows in order to construct the variable with 288 categories that interactively codes country, gender and age group (there are no missings for age):

```
> CGA <- 6 * (CG - 1) + A
```

Chapter 17: Stacked Tables

Chapter 17 considers the CA of several cross-tables concatenated (or juxtaposed). The two functions `rbind()` and `cbind()` provide the tools for binding rows or columns together. For example, assuming the 33590×10 raw data matrix `women` is available and attached as described above, then the five cross-tabulations corresponding to Question 3 depicted in Exhibit 17.1 can be built up as follows in a `for` loop:

```
> women.stack<-table(C, Q3)
> for(j in 6:9){
+  women.stack <- rbind(women.stack, table(women[,j], Q3))
+  }
```

Notice how the columns of `women` can be accessed by name or by column number. If you look at the contents of `women.stack` you will see several rows corresponding to missing data codes for all demographic variables except country and age group. These would have to be omitted before the CA is performed, which can be done in three different ways: (i) by excluding these rows from the matrix, e.g., if rows 38, 39, 47 and 48 correspond to missing values, then remove as follows:

```
> women.stack <- women.stack[-c(38,39,47,48),]
```

(the negative sign before the set of row numbers indicates exclusion); (ii) by changing the missing value codes to NAs as described on the previous page; or (iii) by declaring the missing rows outside the subset of interest in a subset CA as described in Chapter 21 (this is the best option, since it keeps the sample size in each table the same).

χ^2 statistic using χ^2 test function `chisq.test()`

To check the inertias in the table on page 124, we can try R's χ^2 test function `chisq.test` which has as one of its results the χ^2 statistic specified by `$statistic`. We make the calculation for the age variable's cross-tabulation with question 3, which corresponds to rows 27 to 32 of the stacked matrix (after 24 rows for country and 2 rows for gender). Dividing the statistic by the sample size, the total of the table, gives the inertia:

```
> chisq.test(women.stack[27:32,])$statistic /
+                          sum(women.stack[27:32,])

X-squared
0.0421549
```

which agrees with the value for age in the table on page 135.

To build up the table in Exhibit 17.5, the four stacked tables (each with five tables) for the four questions are column-bound using `cbind()`.

The **ca** *package*

At this point, before we enter the intricacies of MCA and its related methods, we are going to leave the "by hand" R exercizes behind and start to use the functions in the **ca** package routinely. The package comprises functions for simple, multiple and joint CA with support for subset analyses and the inclusion of supplementary variables. Furthermore, it offers functions for the graphical display of the results in two and three dimensions. The package is comprised of the following components:

- Simple CA:
  - Computation: `ca()`
  - Printing and summaries: `print.ca()` and `summary.ca()` (and `print.summary.ca()`)
  - Plotting: `plot.ca()` and `plot3d.ca()`
- MCA and JCA:
  - Computation: `mjca()`
  - Printing and summaries: `print.mjca()` and `summary.mjca()` (and `print.summary.mjca()`)
  - Plotting: `plot.mjca()` and `plot3d.mjca()`
- Data sets:
  - `smoke`, `author` and `wg93`

The package contains further functions, such as `iterate.mjca()` for the updating of the Burt matrix in JCA.

The function `ca()` computes simple CA, for example

`ca()` *function*

```
> library(ca)   #this loads ca if not already done using R menu
> data(smoke)
> ca(smoke)
```

performs a simple CA on the `smoke` data set (see pages 221–222). A list of all available entries that are returned by `ca()` is obtained with `names()`:

```
> names(ca(smoke))
 [1] "sv"         "nd"         "rownames"   "rowmass"    "rowdist"
 [6] "rowinertia" "rowcoord"   "rowsup"     "colnames"   "colmass"
[11] "coldist"    "colinertia" "colcoord"   "colsup"     "call"
```

The output of `ca()` is structured as a list-object; for example, the row standard coordinates are obtained with

```
> ca(smoke)$rowcoord
```

Optional arguments for the `ca()` function include an option for setting the dimensionality of the solution (`nd`), options for marking selected rows and/or columns as supplementary ones (`suprow` and `supcol`, respectively) and options for setting subset rows and/or columns (`subsetrow` and `subsetcol`, respectively) for subset CA.

As an extension to the printing method, a summary method is also provided. This gives a more detailed output as follows:

```
> summary(ca(smoke))
```

returns the summary of the CA:

```
  Principal inertias (eigenvalues):
   dim    value      %     cum%  scree plot
   1      0.074759  87.8  87.8  *************************
   2      0.010017  11.8  99.6  ***
   3      0.000414   0.5 100.0
          -------- -----
   Total: 0.085190 100.0

  Rows:
      name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
  1 |   SM |   57  893   31 |  -66  92   3 | -194 800 214 |
  2 |   JM |   93  991  139 |  259 526  84 | -243 465 551 |
  3 |   SE |  264 1000  450 | -381 999 512 |  -11   1   3 |
  4 |   JE |  456 1000  308 |  233 942 331 |   58  58 152 |
  5 |   SC |  130  999   71 | -201 865  70 |   79 133  81 |

  Columns:
      name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
  1 |  non |  316 1000  577 | -393 994 654 |  -30   6  29 |
  2 |  lgh |  233  984   83 |   99 327  31 |  141 657 463 |
  3 |  mdm |  321  983  148 |  196 982 166 |    7   1   2 |
  4 |  hvy |  130  995  192 |  294 684 150 | -198 310 506 |
```

Again, eigenvalues and relative percentages of explained inertia are given for all available dimensions. Additionally, cumulated percentages and a scree plot are shown. The items given in `Rows` and `Columns` include the principal coordinates for the first two dimensions (`k=1` and `k=2`). Squared correlation (`cor`) and contributions (`ctr`) for the points are displayed next to the coordinates. The quantities in these tables are multiplied by 1000 (e.g., the coordinates and masses), which for `cor` and `ctr` means that they are expressed in thousandths, or permills (‰). Quality (`qlt`) is given for the requested solution; i.e., in this case it is the sum of the squared correlations for the first two dimensions. In the case of supplementary variables, an asterisk is appended to the variable names in the output; for example, the summary for the CA of the `smoke` data, where the `none` category (the first column) is treated as supplementary, is:

```
> summary(ca(smoke, supcol=1))
```

In the corresponding section of the output the following is given:

```
...
Columns:
      name   mass  qlt  inr    k=1 cor  ctr    k=2 cor  ctr
1 | (*)non | <NA>   55 <NA> |  292  39 <NA> | -187  16 <NA> |
...
```

showing that masses, inertias and contributions are "not applicable".

Graphical displays in the **ca** *package*

The graphical representation of CA and MCA solutions is commonly done with *symmetric* maps, and this is with the default option in the `plot()` function (`map="symmetric"`). The complete set of `map` options is as follows:

- `"symmetric"` Rows and columns in principal coords (default) i.e., scaled to have inertia equal to principal inertia (eigenvalue, or square of singular value)
- `"rowprincipal"` Rows in principal and columns in standard coords
- `"colprincipal"` Columns in principal and rows in standard coords
- `"symbiplot"` Row and column coords are scaled to have inertias equal to the singular values
- `"rowgab"` Rows in principal coords and columns in standard coords times mass (according to a proposal by Gabriel)
- `"colgab"` Columns in principal coords and rows in standard coords times mass
- `"rowgreen"` Rows in principal coords and columns in standard coords times square root of mass (according to a proposal by Greenacre — see Chapter 13)
- `"colgreen"` Columns in principal coords and rows in standard coords times square root of mass

By default, supplementary variables are added to the plot with a different symbol. The symbols can be defined with the `pch` option in `plot.ca()`. This option takes four values in the following order: plotting point character or symbol for (i) active rows, (ii) supplementary rows, (iii) active columns and (iv) supplementary columns. As a general rule, options that contain entries for rows and for columns contain the entries for the rows first and then those for the columns. For example, the colour of the symbols is specified with the `col` option; by default it is `col=c("#000000", "#FF0000")` — black for rows and red for columns. Instead of these hexadecimal codes, there is a reduced list with names such as `"black"`, `"red"`, `"blue"`, `"green"`, `"gray"`, etc.

The option `what` controls the content of the plot. It can be set to `"all"`, `"active"`, `"passive"` or `"none"` for the rows and for the columns. For example, a plot of only the active (i.e., excluding supplementary) points is created by using `what=c("active", "active")`.

In addition to the `map` scaling options, various options allow certain values to be added to the plot as graphical attributes. The option `mass` selects if the masses of rows or columns should be indicated by the size of the point. Similarly, relative or absolute contributions can be indicated by the colour intensity in the plot by using the `contrib` option.

The option `dim` selects which dimensions to plot, the default being `dim=c(1,2)`, i.e., the first two dimensions are plotted. A plot of the second and third dimensions, for example, is obtained by setting `dim=c(2,3)`. Another possibility for adding the third dimension to the plot is given with the functions `plot3d.ca()` and `plot3d.mjca()`. These two functions rely on the **rgl** package for three-dimensional graphics in R. Their structure is kept similar to their counterparts for two dimensions; for example,

```
> plot3d(ca(smoke, nd=3))
```

creates a three-dimensional display of the CA, shown in Exhibit B.5.

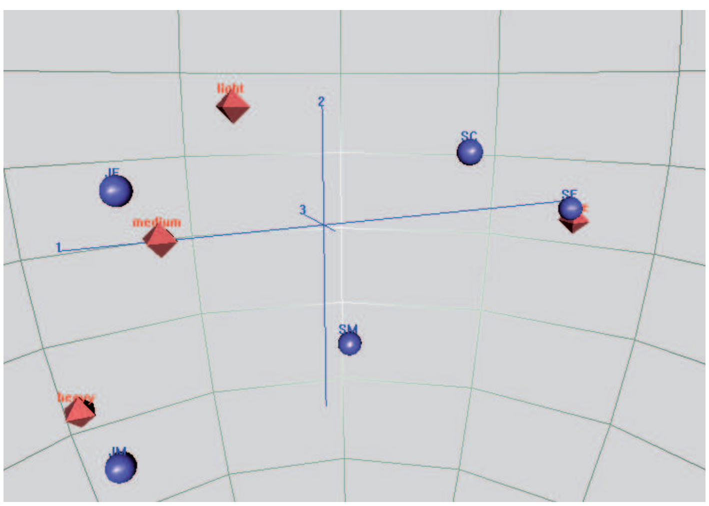


Exhibit B.5: *Three-dimensional display of a simple CA (compare with two-dimensional map in Exhibit B.4).*

This display can be rotated and also zoomed in or out using the mouse and its buttons.

`mjca()` *functions in* **ca** *package*

MCA and JCA are performed with the function `mjca()`. The structure of the function is kept similar to its counterpart from simple CA. The two most striking differences are the format of the input data and the restriction to columns for the analyses. The function `mjca()` takes a response pattern matrix as input.

Within the function, the response pattern matrix is converted to an indicator matrix and a Burt matrix, depending on the type of analysis. Restricting to columns means that only values for the columns are given in the output and the specification of supplementary variables is limited to columns. The "approach" to MCA is specified by the `lambda` option in `mjca()`:

- `lambda="indicator"`: Analysis based on a simple CA of the indicator matrix
- `lambda="Burt"`: Analysis based on an eigenvalue-decomposition of the Burt matrix
- `lambda="adjusted"`: Analysis based on the Burt matrix with an adjustment of inertias (default)
- `lambda="JCA"`: Joint correspondence analysis

By default, `mjca()` performs an adjusted analysis, i.e., `lambda="adjusted"`. For JCA (`lambda="JCA"`), the Burt matrix is updated iteratively by weighted least squares, using the internal function `iterate.mjca()`. This updating function has two convergence criteria, namely `epsilon` and `maxit`. Option `epsilon` sets a convergence criterion by means of maximum absolute difference of the Burt matrix in an iteration step compared to the Burt matrix of the previous step. The maximum number of iterations is given by the option `maxit`. The program iterates until any one of the two conditions is satisfied. Setting one option to `NA` results in ignoring that criterion; for example, exactly 50 iterations without considering convergence are performed with `maxit=50` and `epsilon=NA`.

As with simple CA, the solution is restricted by the `nd` option to two dimensions. However, eigenvalues are given for all possible dimensions, which number $(J-Q)$ for the "indicator" and "Burt" versions of MCA. In the case of an adjusted analysis or a JCA, the eigenvalues are given only for those dimensions k, where the singular values from the Burt matrix λ_k (i.e., the principal inertias of the indicator matrix) satisfy the condition $\lambda_k > 1/Q$.

Chapter 18: Multiple Correspondence Analysis

In Chapter 18 the data on working women, for West and East German samples, were analysed using the indicator and Burt versions of MCA. Assuming that the `women` data frame (with 33590 rows) read previously is available and "attached", the two German samples have country codes 2 and 3 repectively. The part of the `women` corresponding to these two samples can be accessed using a *logical* vector which we call `germany`:

Example of logical operation

```
> germany <- C==2 | C==3
> womenG  <- women[germany,]
```

The first command creates a vector of length 33590 with values `TRUE` corresponding to the rows of the German samples, otherwise `FALSE`. The second command then passes only those rows with `TRUE` values to the new data frame `womenG`. There are 3421 rows in `womenG`, whereas the matrix analysed in Chapter 18 has 3418 rows — three cases that have some missing demographic data have been eliminated (i.e., *listwise deletion* of missings). Variables gender, marital status and education have missing value codes `9`, except for education where they are `98` and `99`. The steps needed to eliminate the missing rows use the same method as above to flag the rows and then eliminate them:

Listwise deletion of missing values

```
> missing <- G==9 | M==9 | E==98 | E==99
> womenG  <- (womenG[!missing,])
```

(If missing values have been replaced by R's `NA` code, as described on page 231, then use `NA` in the above.)

The indicator version of MCA for the first four columns (the four questions on women working or staying at home) is obtained simply as follows:

```
> mjca(womenG[,1:4], lambda="indicator")

 Eigenvalues:
           1        2        3        4        5       6
Value      0.693361 0.513203 0.364697 0.307406 0.21761 0.181521
Percentage 23.11%   17.11%   12.16%   10.25%   7.25%   6.05%
           7        8        9        10       11       12
Value      0.164774 0.142999 0.136322 0.113656 0.100483 0.063969
Percentage 5.49%    4.77%    4.54%    3.79%    3.35%    2.13%

 Columns:
             Q1.1      Q1.2      Q1.3      Q1.4      Q2.1      Q2.2
Mass     0.182929  0.034816  0.005778  0.026477  0.013239  0.095012 ...
ChiDist  0.605519  2.486096  6.501217  2.905510  4.228945  1.277206 ...
Inertia  0.067071  0.215184  0.244222  0.223523  0.236761  0.154988 ...
Dim. 1  -0.355941 -0.244454 -0.279167  2.841498 -0.696550 -0.428535 ...
Dim. 2  -0.402501  1.565682  3.971577 -0.144653 -2.116572 -0.800930 ...
```

and the Burt version:

```
> mjca(womenG[,1:4], lambda="Burt")

 Eigenvalues:
           1        2        3        4        5        6
Value      0.480749 0.263377 0.133004 0.094498 0.047354 0.03295
Percentage 41.98%   23%      11.61%   8.25%    4.13%    2.88%
           7        8        9        10       11       12
Value      0.027151 0.020449 0.018584 0.012918 0.010097 0.004092
Percentage 2.37%    1.79%    1.62%    1.13%    0.88%    0.36%

 Columns:
             Q1.1     Q1.2     Q1.3      Q1.4      Q2.1      Q2.2 ...
Mass     0.182929 0.034816 0.005778  0.026477  0.013239  0.095012 ...
ChiDist  0.374189 1.356308 3.632489  2.051660  2.354042  0.721971 ...
Inertia  0.025613 0.064046 0.076244  0.111452  0.073363  0.049524 ...
Dim. 1   0.355941 0.244454 0.279167 -2.841498  0.696550  0.428535 ...
```

```
Dim. 2  -0.402501 1.565682 3.971577 -0.144653 -2.116572 -0.800930 ...
```

The total inertia can be computed in the two cases as the sum of squared singular values, as in the simple CA case:

```
> sum(mjca(womenG[,1:4], lambda="indicator")$sv^2)
[1] 3

> sum(mjca(womenG[,1:4], lambda="Burt")$sv^2)
[1] 1.145222
```

The contributions of each subtable of the Burt matrix to the total inertia is given in the component called `subinertia` of the `mjca` object, so the sum of these also gives the total inertia:

```
> sum(mjca(womenG[,1:4], lambda="Burt")$subinertia)
[1] 1.145222
```

Since the total inertia is the average of the 16 subtables, the inertia of individual subtables are 16 times the values in `$subinertia`:

```
> 16*mjca(womenG[,1:4], lambda="Burt")$subinertia
          [,1]      [,2]      [,3]      [,4]
[1,] 3.0000000 0.3657367 0.4261892 0.6457493
[2,] 0.3657367 3.0000000 0.8941517 0.3476508
[3,] 0.4261892 0.8941517 3.0000000 0.4822995
[4,] 0.6457493 0.3476508 0.4822995 3.0000000
```

To obtain the positions of the supplementary variables:

```
> summary(mjca(womenG, lambda="Burt", supcol=5:9))

Principal inertias (eigenvalues):

 dim    value      %     cum%   scree plot
 1      0.480749   42.0  42.0   *************************
 2      0.263377   23.0  65.0   **************
 3      0.133004   11.6  76.6   *******
 4      0.094498   8.3   84.8   *****
 5      0.047354   4.1   89.0   **
 6      0.032950   2.9   91.9   **
 7      0.027151   2.4   94.2   *
 8      0.020449   1.8   96.0   *
 9      0.018584   1.6   97.6   *
 10     0.012918   1.1   98.8
 11     0.010097   0.9   99.6
 12     0.004092   0.4  100.0
        -------- -----
 Total: 1.145222 100.0

Columns:
       name   mass  qlt  inr     k=1  cor  ctr     k=2  cor  ctr
1  |   Q1.1 |  183  740    6 |   247  435   23 |  -207  305   30 |
2  |   Q1.2 |   35  367   14 |   169   16    2 |   804  351   85 |
3  |   Q1.3 |    6  318   16 |   194    3    0 |  2038  315   91 |
4  |   Q1.4 |   26  923   24 | -1970  922  214 |   -74    1    1 |
```

```
5  |    Q2.1 |   13  255   16 |   483   42    6 | -1086  213   59 |
6  |    Q2.2 |   95  494   11 |   297  169   17 |  -411  324   61 |
.         .       .    .    .        .    .    .        .    .    .
.         .       .    .    .        .    .    .        .    .    .
17 | (*)C.2 | <NA>  283 <NA> |   -89   48 <NA> |   195  234 <NA> |
18 | (*)C.3 | <NA>  474 <NA> |   188   81 <NA> |  -413  393 <NA> |
19 | (*)G.1 | <NA>   26 <NA> |   -33    5 <NA> |    67   21 <NA> |
20 | (*)G.2 | <NA>   24 <NA> |    34    5 <NA> |   -68   19 <NA> |
21 | (*)A.1 | <NA>   41 <NA> |  -108   12 <NA> |  -170   29 <NA> |
22 | (*)A.2 | <NA>   52 <NA> |   -14    0 <NA> |  -172   52 <NA> |
.         .       .    .    .        .    .    .        .    .    .
.         .       .    .    .        .    .    .        .    .    .
```

The supplementary categories are marked by a `*` and have no masses, inertia values (`inr`) nor contributions to the principal axes (`ctr`).

Chapter 19: Joint Correspondence Analysis

To obtain the JCA of the same data as in Exhibit 19.3, simply change the `lambda` option to `"JCA"`. Here percentages of inertia are not given for individual axes, but only for the solution space as a whole, since the axes are not nested:

```
> summary(mjca(womenG[,1:4], lambda="JCA"))

Principal inertias (eigenvalues):

 1      0.353452
 2      0.128616
 3      0.015652
 4      0.003935
        --------
 Total: 0.520617

 Diagonal inertia discounted from eigenvalues: 0.125395
 Percentage explained by JCA in 2 dimensions: 90.2%
 (Eigenvalues are not nested)
 [Iterations in JCA: 31 , epsilon = 9.33e-05]

Columns:
     name   mass  qlt  inr     k=1  cor  ctr    k=2  cor  ctr
1  | Q1.1 |  183  969   21 |   204  693   22 | -129  276   24 |
2  | Q1.2 |   35  803   23 |   144   61    2 |  503  742   69 |
3  | Q1.3 |    6  557   32 |   163    9    0 | 1260  548   71 |
4  | Q1.4 |   26  992  137 | -1637  991  201 |  -45    1    0 |
5  | Q2.1 |   13  597   31 |   394  125    6 | -764  471   60 |
6  | Q2.2 |   95  956   26 |   250  431   17 | -276  525   56 |
.      .       .    .    .       .    .    .      .    .    .
.      .       .    .    .       .    .    .      .    .    .
```

The squared correlations, and thus the qualities too, are all much higher in the JCA.

Notice that in the JCA solution, the "total" inertia is the inertia of the modified Burt matrix, which includes a part due to the modified diagonal blocks — this additional part is the "`Diagonal inertia discounted from eigenvalues: 0.125395`" which has to be subtracted from the total to get the total inertia due to the off-diagonal blocks. Since the solution requested is two-dimensional and fits the diagonal blocks exactly by construction, the first two eigenvalues also contain this additional part, which has to be discounted as well. The proportion of (off-diagonal) inertia explained is thus:

$$\frac{0.3534 + 0.1286 - 0.1254}{0.5206 - 0.1254} = 0.9024$$

i.e., the percentage of 90.2% reported above (see Theoretical Appendix, (A.32)). The denominator above, the adjusted total $0.5206 - 0.1254 = 0.3952$, can be verified to be the same as:

$$\text{inertia of } \mathbf{B} - \frac{J - Q}{Q} = 1.1452 - \frac{12}{16} = 0.3952$$

To obtain the adjusted MCA solution, that is the same standard coordinates as in MCA but (almost) optimal scaling factors ("almost" optimal because the nesting property is retained, whereas the optimal adjustments do not preserve nesting), either use the `lambda` option `"adjusted"` or leave out this option since it is the default:

```
> summary(mjca(womenG[,1:4]))
Principal inertias (eigenvalues):

 dim    value      %     cum%  scree plot
 1      0.349456  66.3  66.3  *************************
 2      0.123157  23.4  89.7  *********
 3      0.023387   4.4  94.1  *
 4      0.005859   1.1  95.2

 Adjusted total inertia: 0.526963

Columns:
     name   mass  qlt  inr     k=1  cor ctr    k=2  cor ctr
1  | Q1.1 |  183  996   22 |   210  687  23 | -141  309  30 |
2  | Q1.2 |   35  822   26 |   145   53   2 |  549  769  85 |
3  | Q1.3 |    6  562   38 |   165    8   0 | 1394  554  91 |
4  | Q1.4 |   26 1009  141 | -1680 1008 214 |  -51    1   1 |
5  | Q2.1 |   13  505   36 |   412  119   6 | -743  387  59 |
6  | Q2.2 |   95  947   27 |   253  424  17 | -281  522  61 |
.       .      .    .    .       .    .   .      .    .   .
.       .      .    .    .       .    .   .      .    .   .
```

The adjusted total inertia, used to calculate the percentages above, is calculated just after (19.5) on page 149. The first two adjusted principal inertias (eigenvalues) are calculated just after (19.6) (see also (A.35) and (A.36)).

Chapter 20: Scaling Properties of MCA

Chapter 20 generalizes the ideas of Chapter 7, also Chapter 8, to the multivariate case. The data set used in this chapter is the science and environment data available as a data set in our **ca** package, so all you need to do to load this data set is to issue the command:

```
> data(wg93)
```

The resulting data frame `wg93` contains the four questions described on page 153, as well as three demographic variables: gender, age and education (the last two have six categories each). The MCA map of Exhibit 20.1 is obtained as follows, this time after saving the MCA results in object `wg93.mca`:

```
> wg93.mca <- mjca(wg93[,1:4], lambda="indicator")
> plot(wg93.mca, what=c("none", "all"))
```

The map might turn out inverted on the first or second axis, but that is — as we have said before — of no consequence.

Exhibit 20.2 is obtained by formatting the contributions to axis 1 as a 5×4 matrix (first the principal coordinates `wg93.F` are calculated, then raw contributions `wg93.coli`):

```
> wg93.F    <- wg93.mca$colcoord %*% sqrt(wg93.mca$sv)
> wg93.coli <- diag(wg93.mca$colmass)%*%wg93.F^2
> matrix(round(1000*wg93.coli[,1] / wg93.mca$sv[1]^2, 0), nrow=5)

     [,1] [,2] [,3] [,4]
[1,]  115  174  203   25
[2,]   28   21    6    3
[3,]   12    7   22    9
[4,]   69   41   80    3
[5,]   55   74   32   22
```

The following commands assign the first standard coordinates as the four item scores for each of the 871 respondents, and the average score:

```
> Ascal <- wg93.mca$colcoord[1:5,1]
> Bscal <- wg93.mca$colcoord[6:10,1]
> Cscal <- wg93.mca$colcoord[11:15,1]
> Dscal <- wg93.mca$colcoord[16:20,1]
> As    <- Ascal[wg93[,1]]
> Bs    <- Bscal[wg93[,2]]
> Cs    <- Cscal[wg93[,3]]
> Ds    <- Dscal[wg93[,4]]
> AVEs  <- (As+Bs+Cs+Ds)/4
```

All squared correlations between the item scores and the average can be calculated at once by binding them together into a matrix and using the correlation function `cor()`:

Correlation function `cor()`

```
> cor(cbind(As,Bs,Cs,Ds,AVEs))^2
              As          Bs         Cs          Ds      AVEs
As   1.000000000 0.139602528 0.12695057 0.005908244 0.5100255
Bs   0.139602528 1.000000000 0.18681032 0.004365286 0.5793057
Cs   0.126950572 0.186810319 1.00000000 0.047979010 0.6273273
Ds   0.005908244 0.004365286 0.04797901 1.000000000 0.1128582
AVEs 0.510025458 0.579305679 0.62732732 0.112858161 1.0000000
```

The squared correlations (or discrimination measures in homogeneity analysis) on page 157 are recovered in the last column (or row). Their average gives the first principal inertia of the indicator matrix:

```
> sum(cor(cbind(As,Bs,Cs,Ds,AVEs))[1:4,5]^2) / 4
[1] 0.4573792

> wg93.mca$sv[1]^2
[1] 0.4573792
```

Another result, not mentioned in Chapter 20, is that MCA also maximizes the average covariance between all four item scores. First, calculate the 4×4 covariance matrix between the scores (multiplying by $(N-1)/N$ to obtain the "biased" covariances, since the function `cov()` computes the usual "unbiased" estimates by dividing by $N-1$), and then calculate the average value of the 16 values using the function `mean()`:

Covariance function `cov()`

```
> cov(cbind(As,Bs,Cs,Ds)) * 870 / 871
           As         Bs        Cs         Ds
As 1.11510429 0.44403796 0.4406401 0.04031951
Bs 0.44403796 1.26657648 0.5696722 0.03693604
Cs 0.44064007 0.56967224 1.3715695 0.12742741
Ds 0.04031951 0.03693604 0.1274274 0.24674968

> mean(cov(cbind(As,Bs,Cs,Ds)) * 870 / 871)
[1] 0.4573792
```

Notice that the sum of the variances of the four item scores is equal to 4:

```
> sum(diag(cov(cbind(As,Bs,Cs,Ds)) * 870 / 871))
[1] 4
```

The individual respondents' variance measure in (20.2) is calculated and averaged over the whole sample:

```
> VARs <- ((As-AVEs)^2 + (Bs-AVEs)^2 +( Cs-AVEs)^2 +
+          (Ds-AVEs)^2)/4
> mean(VARs)
[1] 0.5426208
```

which is the loss of homogeneity, equal to 1 minus the first principal inertia.

Exhibit 20.3 can be obtained as the `"rowprincipal"` map, suppressing the row labels (check the plotting options by typing `help(plot.ca)`):

```
> plot(wg93.mca, map="rowprincipal", labels=c(0,2))
```

Chapter 21: Subset Correspondence Analysis

Subset CA in Chapter 21 is presently implemented only in the `ca()` function, but since the Burt matrix is accessible from `mjca()` one can easily do the subset MCA on the Burt matrix. First, an example of subset CA using the `author` dataset, provided with the **ca** package. To reproduce the subset analyses of consonants and vowels:

```
> data(author)
> vowels <- c(1,5,9,15,21)
> consonants <- c(1:26)[-vowels]
> summary(ca(author,subsetcol=consonants))
```

```
Principal inertias (eigenvalues):

 dim    value     %     cum%  scree plot
 1      0.007607  46.5  46.5  *************************
 2      0.003253  19.9  66.4  ***********
 3      0.001499  9.2   75.6  *****
 4      0.001234  7.5   83.1  ****
 .       .         .      .   .
 .       .         .      .   .
        -------- -----
 Total: 0.01637 100.0

Rows:
      name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1  |   td( |   85   59   29 |    7   8   1 |  -17  50   7 |
2  |   d() |   80  360   37 |  -39 196  16 |  -35 164  31 |
3  |   lw( |   85  641   81 | -100 637 111 |    8   4   2 |
4  |   ew( |   89  328   61 |   17  27   4 |   58 300  92 |
.       .       .    .    .      .   .   .      .   .   .
.       .       .    .    .      .   .   .      .   .   .

Columns:
     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1  |    b |   16  342   21 |  -86 341  15 |   -6   2   0 |
2  |    c |   23  888   69 | -186 699 104 |  -97 189  66 |
3  |    d |   46  892  101 |  168 783 171 |  -63 110  56 |
4  |    f |   19  558   33 | -113 467  33 |  -50  91  15 |
.       .      .    .    .      .   .   .      .   .   .
.       .      .    .    .      .   .   .      .   .   .

> summary(ca(author,subsetcol=vowels))

Principal inertias (eigenvalues):

 dim    value     %     cum%  scree plot
 1      0.001450  61.4  61.4  *************************
 2      0.000422  17.9  79.2  ******
 3      0.000300  12.7  91.9  ****
 4      0.000103   4.4  96.3  *
 5      0.000088   3.7 100.0
        -------- -----
 Total: 0.002364 100.0

Rows:
      name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1  |   td( |   85  832  147 |   58 816 195 |    8  15  13 |
2  |   d() |   80  197   44 |  -12 118   9 |  -10  79  20 |
3  |   lw( |   85  235   33 |   14 226  12 |   -3   9   2 |
4  |   ew( |   89  964  109 |   31 337  60 |   42 627 382 |
.       .       .    .    .      .   .   .      .   .   .
.       .       .    .    .      .   .   .      .   .   .
```

```
Columns:
    name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1 |    a |   80  571   79 |    9  34   4 |  -35 537 238 |
2 |    e |  127  898  269 |   67 895 393 |    4   3   5 |
3 |    i |   70  800  221 |  -59 468 169 |   50 332 410 |
4 |    o |   77  812  251 |  -79 803 329 |   -8   9  12 |
5 |    u |   30  694  179 |  -71 359 105 |  -69 334 335 |
```

We now demonstrate the subset MCA that is documented on pages 165–166, i.e., for the Burt matrix of the working women data set stored in `womenG`, after elimination of missing data for the demographics (see page 237). First, the function `mjca()` is used merely to obtain the Burt matrix, and then the subset CA is applied to that square part of the Burt matrix not corresponding to the missing data categories (see re-arranged Burt matrix in Exhibit 21.3). The selection is performed by defining a vector of indices named `subset` below:

```
> womenG.B <- mjca(womenG)$Burt
> subset <- c(1:16)[-c(4,8,12,16)]
> summary(ca(womenG.B[1:16,1:16], subsetrow=subset,
+                subsetcol=subset))
```

```
Principal inertias (eigenvalues):

 dim    value     %     cum%   scree plot
 1      0.263487  41.4  41.4  *************************
 2      0.133342  21.0  62.4  *************
 3      0.094414  14.9  77.3  *********
 4      0.047403   7.5  84.7  *****
 5      0.032144   5.1  89.8  ***
 6      0.026895   4.2  94.0  ***
 7      0.019504   3.1  97.1  **
 8      0.013096   2.1  99.1  *
 9      0.005130   0.8  99.9
 10     0.000231   0.0 100.0
 11     0.000129   0.0 100.0
        -------- -----
 Total: 0.635808 100.0

Rows:
     name   mass  qlt  inr     k=1 cor ctr    k=2 cor ctr
1  | Q1.1 |  183  592   25 |  -228 591  36 |   11   1   0 |
2  | Q1.2 |   35  434   98 |   784 345  81 | -397  88  41 |
3  | Q1.3 |    6  700  119 |  2002 306  88 | 2273 394 224 |
4  | Q2.1 |   13  535  113 | -1133 236  65 | 1276 299 162 |
5  | Q2.2 |   95  452   69 |  -442 421  71 | -119  30  10 |
6  | Q2.3 |  120  693   64 |   482 688 106 |  -40   5   1 |
7  | Q3.1 |   28  706  114 | -1040 412 114 |  878 294 160 |
8  | Q3.2 |  152  481   38 |  -120  91   8 | -249 390  71 |
9  | Q3.3 |   47  748  106 |   990 681 175 |  312  67  34 |
10 | Q4.1 |  143  731   49 |  -390 702  83 |   80  29   7 |
11 | Q4.2 |   66  583   84 |   582 414  84 | -371 168  68 |
12 | Q4.3 |    7  702  119 |  1824 312  90 | 2041 391 222 |
```

The adjustments of the scale by linear regression to best fit the off-diagonal tables of the Burt submatrix is not easily done. The code is rather lengthy, so is not described here but rather put on the website for the moment, to be implemented in the **ca** package at a later date.

Chapter 22: Analysis of Square Tables

As shown in Chapter 21 the CA of a square asymmetric matrix consists in splitting the table into symmetric and skew-symmetric parts and then performing CA on the symmetric part and an uncentred CA on the skew-symmetric part, with the same weights and χ^2-distances throughout. Both analyses are neatly subsumed in the CA of the block matrix shown in (22.4). After reading the mobility table into a data frame named `mob`, the sequence of commands to set up the block matrix and then do the CA is as follows. Notice that `mob` has to be first converted to a matrix; otherwise we cannot bind the rows and columns together properly to create the block matrix `mob2`.

```
> mob <- as.matrix(mob)
> mob2 <- rbind(cbind(mob,t(mob)), cbind(t(mob), mob))
> summary(ca(mob2))
```

```
Principal inertias (eigenvalues):

 dim    value      %    cum%  scree plot
 1      0.388679  24.3  24.3  *************************
 2      0.232042  14.5  38.8  ***************
 3      0.158364   9.9  48.7  **********
 4      0.158364   9.9  58.6  **********
 5      0.143915   9.0  67.6  *********
 6      0.123757   7.7  75.4  ********
 7      0.081838   5.1  80.5  *****
 8      0.070740   4.4  84.9  *****
 9      0.049838   3.1  88.0  ***
 10     0.041841   2.6  90.6  ***
 11     0.041841   2.6  93.3  ***
 12     0.022867   1.4  94.7  *
 13     0.022045   1.4  96.1  *
 14     0.012873   0.8  96.9  *
 15     0.012873   0.8  97.7  *
 16     0.010360   0.6  98.3  *
 17     0.007590   0.5  98.8  *
 18     0.007590   0.5  99.3  *
 19     0.003090   0.2  99.5
 20     0.003090   0.2  99.7
 21     0.001658   0.1  99.8
 22     0.001148   0.1  99.9
 23     0.001148   0.1  99.9
 24     0.000620   0.0  99.9
 25     0.000381   0.0 100.0
 26     0.000381   0.0 100.0
 27     0.000147   0.0 100.0
        -------- -----
 Total: 1.599080 100.0
```

```
Rows:
     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1   |  Arm |   43  426   54 | -632 200  44 |  671 226  84 |
2   |  Art |   55  886  100 | 1521 793 327 |  520  93  64 |
3   |  Tcc |   29   83   10 | -195  73   3 |   73  10   1 |
4   |  Cra |   18  293   32 |  867 262  34 | -298  31   7 |
.        .     .    .    .      .   .   .      .   .   .
.        .     .    .    .      .   .   .      .   .   .
15  |  ARM |   43  426   54 | -632 200  44 |  671 226  84 |
16  |  ART |   55  886  100 | 1521 793 327 |  520  93  64 |
17  |  TCC |   29   83   10 | -195  73   3 |   73  10   1 |
18  |  CRA |   18  293   32 |  867 262  34 | -298  31   7 |
.        .     .    .    .      .   .   .      .   .   .
.        .     .    .    .      .   .   .      .   .   .

Columns:
     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1   |  ARM |   43  426   54 | -632 200  44 |  671 226  84 |
2   |  ART |   55  886  100 | 1521 793 327 |  520  93  64 |
3   |  TCC |   29   83   10 | -195  73   3 |   73  10   1 |
4   |  CRA |   18  293   32 |  867 262  34 | -298  31   7 |
.        .     .    .    .      .   .   .      .   .   .
.        .     .    .    .      .   .   .      .   .   .
15  |  Arm |   43  426   54 | -632 200  44 |  671 226  84 |
16  |  Art |   55  886  100 | 1521 793 327 |  520  93  64 |
17  |  Tcc |   29   83   10 | -195  73   3 |   73  10   1 |
18  |  Cra |   18  293   32 |  867 262  34 | -298  31   7 |
.        .     .    .    .      .   .   .      .   .   .
.        .     .    .    .      .   .   .      .   .   .
```

The principal inertias coincide with Exhibit 22.4, and since the first two dimensions correspond to the symmetric part of the matrix, each set of coordinates is just a repeat of the same set of values.

Dimensions 3 and 4, with repeated eigenvalues, correspond to the skew-symmetric part and their coordinates turn out as follows (to get more than the default two dimensions in the summary, change the original command to `summary(ca(mob2, nd=4))`):

```
Rows:
     name     k=3 cor ctr    k=4 cor ctr
1   |  Arm |   -11   0   0 |  416  87  47 |
2   |  Art |    89   3   3 |  423  61  62 |
3   |  Tcc |  -331 211  20 |  141  38   4 |
4   |  Cra |  -847 250  80 |   92   3   1 |
.        .       .   .   .      .   .   .
.        .       .   .   .      .   .   .
15  |  ARM |    11   0   0 | -416  87  47 |
16  |  ART |   -89   3   3 | -423  61  62 |
17  |  TCC |   331 211  20 | -141  38   4 |
18  |  CRA |   847 250  80 |  -92   3   1 |
```

```
.        .        .    .    .       .    .    .
.        .        .    .    .       .    .    .

Columns:
     name     k=3 cor ctr     k=4 cor ctr
1  |  ARM |  -416  87  47 |   -11   0   0 |
2  |  ART |  -423  61  62 |    89   3   3 |
3  |  TCC |  -141  38   4 |  -331 211  20 |
4  |  CRA |   -92   3   1 |  -847 250  80 |
.        .        .    .    .       .    .    .
.        .        .    .    .       .    .    .
15 |  Arm |   416  87  47 |    11   0   0 |
16 |  Art |   423  61  62 |   -89   3   3 |
17 |  Tcc |   141  38   4 |   331 211  20 |
18 |  Cra |    92   3   1 |   847 250  80 |
.        .        .    .    .       .    .    .
.        .        .    .    .       .    .    .
```

which shows that the skew-symmetric coordinates reverse sign within the row and column blocks, but also swap over, with the third axis row solution equal to the fourth axis column solution and vice versa. In any case, only one set of coordinates is needed to plot the objects in each map, but the interpretation of the maps is different, as explained in Chapter 22.

Chapter 23: Data Recoding

Chapter 23 involves mostly simple transformations of the data and then regular applications of CA. As an illustration of analyzing continuous data, we assume that the European Union indicators data have been read into a data frame named `EU`. Then the conversion to ranks (using R function `rank()` and again the very useful `apply()` function to obtain `EUr`), and the doubling (to obtain `EUd`) are performed as follows:

Converting to ranks with `rank()` *function*

```
> EUr <- apply(EU,2,rank) - 1
> EUd <- cbind(EUr, 11-EUr)
> colnames(EUd) <- c(paste(colnames(EU), "-", sep=""),
+                    paste(colnames(EU), "+", sep=""))
> EUd
   Unemp- GDPH- PCH- PCP- RULC- Unemp+ GDPH+ PCH+ PCP+ RULC+
Be      6     6    6  6.5   4.5      5     5    5  4.5   6.5
De      4    11   10  0.0   7.0      7     0    1 11.0   4.0
Ge      2    10   11  5.0   6.0      9     1    0  6.0   5.0
Gr      5     1    1  1.0  11.0      6    10   10 10.0   0.0
Sp     11     3    3 10.0   2.0      0     8    8  1.0   9.0
Fr      7     8    8  3.5   4.5      4     3    3  7.5   6.5
Ir     10     2    2 11.0   1.0      1     9    9  0.0  10.0
It      9     7    7  9.0   9.0      2     4    4  2.0   2.0
Lu      0     9    9  3.5   8.0     11     2    2  7.5   3.0
Ho      8     5    4  6.5   3.0      3     6    7  4.5   8.0
Po      1     0    0  8.0   0.0     10    11   11  3.0  11.0
UK      3     4    5  2.0  10.0      8     7    6  9.0   1.0
```

Notice how the column names are constructed with the `paste()` function. The analysis of Exhibit 23.5 is thus obtained simply as `ca(EUd)`.

Chapter 24: Canonical Correspondence Analysis

The results of Chapter 24 cannot be obtained using the **ca** package, but using either the XLSTAT program (described later) or Jari Oksanen's **vegan** package (see web resources in the Bibliographical Appendix), which not only does CCA but also CA and PCA (but without many of the options we have in the **ca** package). Since this package is usually used in an ecological context, like the example in Chapter 24, we shall refer here to "sites" (samples), "species" and (explanatory) "variables". Using **vegan** is just as easy as using **ca**: the main function is called `cca()` and can be used in either of the two following formats:

```
cca(X,Y,Z)
cca(X ~ Y + condition(Z))
```

where `X` is the sites×species matrix of counts, `Y` is the sites×variables matrix of explanatory data and `Z` is the sites×variables matrix of conditioning data if we want to perform (optionally) a partial CCA. The second format is in the form of a regression-type model formula, but here the first type will be used. If only `X` is specified, the analysis is a CA (so try, for example, one of the previous analyses, for example `summary(cca(author))` to compare the results with previous ones — notice that the books are referred to as "sites" and the letters as "species", and that the default plotting option, for example `plot(cca(author))`, is what we called `"colprincipal"`). If `X` and `Y` are specified, the analysis is a CCA. If `X`, `Y` and `Z` are specified, the analysis is a partial CCA.

Assuming now that the biological data of Chapters 10 and 24 are read into the data frame `bio` as a 13×92 table, and that the three variables `Ba`, `Fe` and `PE` are read into `env` as a 13×3 table whose columns are log-transformed to variables with names `logBa`, `logFe` and `logPE`; then the CCA can be performed simply as follows:

```
> summary(cca(bio, env))

Call:
cca(X = bio, Y = env)

Partitioning of mean squared contingency coefficient:

Total         0.7826
Constrained   0.2798
Unconstrained 0.5028

Eigenvalues, and their contribution to the
             mean squared contingency coefficient

             CCA1   CCA2    CCA3    CA1    CA2     CA3     CA4     CA5
lambda     0.1895 0.0615 0.02879 0.1909 0.1523 0.04159 0.02784 0.02535
accounted  0.2422 0.3208 0.35755 0.2439 0.4385 0.49161 0.52719 0.55957
               CA6     CA7     CA8     CA9
lambda     0.02296 0.01654 0.01461 0.01076
accounted  0.58891 0.61004 0.62871 0.64245

Scaling 2 for species and site scores
--- Species are scaled proportional to eigenvalues
--- Sites are unscaled: weighted dispersion equal on all dimensions
```

```
Species scores

                CCA1      CCA2      CCA3        CA1       CA2       CA3
Myri_ocul  0.1732392  0.245915 -0.070907  0.6359626 -0.063479  0.031990
Chae_seto  0.5747974 -0.270816  0.011814 -0.5029157 -0.674207  0.093354
Amph_falc  0.2953878 -0.114067  0.075979 -0.2224138  0.041797 -0.005020
Myse_bide -0.5271092 -0.505262 -0.103978 -0.0789909  0.176683 -0.484208
Goni_macu -0.1890403  0.122783 -0.044679 -0.1045244  0.030134  0.111827
Amph_fili -0.9989672 -0.075696  0.107184 -0.3506103  0.076968  0.004931
    .        .          .         .         .          .         .
    .        .          .         .         .          .         .

Site constraints (linear combinations of constraining variables)

        CCA1     CCA2     CCA3
S4  -0.06973  0.75885 -2.29951
S8  -0.35758  1.47282  2.27467
S9   0.48483 -0.72459 -0.66547
S12  0.02536  0.27129 -0.14677
S13  0.30041 -0.01531 -0.80821
S14  0.79386  1.16229  0.24314
S15  0.96326 -0.88970  0.14630
S18 -0.16753  0.25048 -0.77451
S19  0.36890 -0.81800  1.50620
S23 -0.09967 -1.90159  0.06877
S24  0.05478  0.96184 -0.10635
R40 -3.71393 -0.20698  0.53031
R42 -2.96641 -0.18264 -0.67736

Biplot scores for constraining variables

        CCA1     CCA2    CCA3
logBa 0.9957 -0.08413 0.03452
logFe 0.6044 -0.72088 0.33658
logPE 0.4654  0.55594 0.68710
```

Notice the following:

— the `mean squared contingency coefficient` is the total inertia;

— the principal inertias in the constrained space are headed `CCA1`, `CCA2`, etc., and the principal inertias in the unconstrained space `CA1`, `CA2`, etc.;

— the percentages are all expressed relative to the total inertia;

— `Scaling 2` means rows (sites) in standard coordinates, and columns (species) in principal coordinates, i.e., the `"colprincipal"` scaling in the `plot.ca()` function;

— the `Species scores` are column principal coordinates;

— the `Site constraints` are the row standard coordinates;

— the `Biplot scores for constraining variables` are the weighted correlation coefficients between the explanatory variables and the site coordinates.

Chapter 25: Aspects of Stability and Inference

Finally, Chapter 25 performs various bootstraps of tables to investigate their variability, as well as permutation tests to test null hypotheses. For example, the 1000 replications of the `author` data, shown in the partial bootstrap CA map of Exhibit 25.1, are obtained as follows (with comments interspersed):

```
> data(author)
> author.ca <- ca(author)
> nsim <- 1000
> # compute row sums
> author.rowsum <- apply(author, 1, sum)
> # compute nsim simulations of first book
> author.sim <- rmultinom(nsim, author.rowsum[1], prob = author[1,])
```

Multinomial random sampling using `rmultinom()`

```
> # compute nsim simulations of other books and column-bind
> for (i in 2:12) {
+  author.sim<-cbind(author.sim,
+                    rmultinom(nsim,author.rowsum[i],
+                              prob = author[i,]))
+  }
> # transpose to have same format as original matrix
> author.sim <- t(author.sim)
> author.sim2 <- matrix(rep(0,nsim*12*26), nrow=nsim*12)
> # reorganize rows so that matrices are together
> for (k in 1:nsim) {
+  for (i in 1:12) {
+     author.sim2[(k-1)*12+i,] <- author.sim[k+(i-1)*nsim,]
+     }
+  }
```

The coordinates are now calculated of the simulated columns using the transition formula from row standard to column principal coordinates:

```
> # get standard coordinates for rows
> author.rowsc <- author.ca$rowcoord[,1:2]
> # calculate pc's of all replicates using transition formula
> author.colsim <- t(t(author.rowsc) %*% author.sim2[1:12,]) /
+                  apply(author.sim2[1:12,],2,sum)
> for (k in 2:nsim) {
+  author.colsim <- rbind(author.colsim, t(t(author.rowsc) %*%
+                    author.sim2[((k-1)*12+1):(k*12),])/
+                    apply(author.sim2[((k-1)*12+1):(k*12),], 2, sum))
> # reorganize rows of coordinates so that letters are together
> author.colsim2 <- matrix(rep(0, nsim*26*2), nrow=nsim*26)
> for (j in 1:26) {
+  for (k in 1:nsim) {
+    author.colsim2[(j-1)*nsim+k,]<-author.colsim[j+(k-1)*26,]
+    }
+  }
```

The plotting of the points and the convex hulls:

```
> # plot all points (use first format of coords for labelling...)
> plot(author.colsim[,1], -author.colsim[,2], xlab="dim1",
+     ylab="dim2",type="n")
> text(author.colsim[,1], -author.colsim[,2], letters, cex=0.5, col="gray")
> # plot convex hulls for each letter
> # first calculate pc's of letters for original matrix
> author.col <- t(t(author.rowsc) %*% author) /
+             apply(author, 2, sum)
> for (j in 1:26) {
+  points <- author.colsim2[(nsim*(j-1)+1):(nsim*j),]
+ # note we are reversing second coordinate in all these plots
+  points[,2] <- -points[,2]
+  hpts <- chull(points)
+  hpts <- c(hpts,hpts[1])
+  lines(points[hpts,], lty=3)
+  text(author.col[j,1], -author.col[j,2], letters[j], font=2, cex=1.5)
+  }
```

Finally, peeling away the convex hulls until just under 5% of the points have been removed from each cloud of replicates, and then plotting the convex hulls of the remaining points:

```
> plot(author.colsim2[,1], -author.colsim2[,2], xlab="dim1",
+     ylab="dim2", type="n")
> for (j in 1:26) {
+  points <- author.colsim2[(nsim*(j-1)+1):(nsim*j),]
+ # note we are reversing second coordinate in all these plots
+  points[,2] <- -points[,2]
+  repeat {
+    hpts <- chull(points)
+    npts <- nrow(points[-hpts,])
+    if(npts/nsim<0.95) break
+    points <- points[-hpts,]
+        }
+  hpts <- c(hpts,hpts[1])
+  lines(points[hpts,], lty=3)
+  text(author.col[j,1], -author.col[j,2], letters[j],
+       font=2)
+  }
```

To plot concentration ellipses, the package **ellipse** needs to be downloaded from the R website `www.R-project.org`. The script for plotting concentration ellipses using the bootstrap replicates computed above is as follows:

```
> # confidence ellipses - needs package 'ellipse'
> plot(author.colsim2[,1],-author.colsim2[,2],xlab="dim1",
+      ylab="dim2", type="n")
> for (j in 1:26) {
```

```
+   points <- author.colsim2[(nsim*(j-1)+1):(nsim*j),]
+   # note we are reversing second coordinate in all these plots
+   points[,2] <- -points[,2]
+   covpoints <- cov(points)
+   meanpoints <- apply(points,2,mean)
+   lines(ellipse(covpoints, centre = meanpoints))
+   text(author.col[j,1],-author.col[j,2],letters[j],
+        font=2)
+                  }
```

To reproduce the ellipses based on the Delta method in Exhibit 25.3, the covariance matrix of the estimated principal coordinates is needed, which is provided in the output of the SPSS program. More details and additional R code are given on the CARME network website `www.carme-n.org`.

Permutation test

The permutation test on the author data is performed by exhaustively listing the $11 \times 9 \times 7 \times 5 \times 3 = 10395$ different ways of assigning the labels of the texts to the points in the CA map, recalculating the sum of distance between pairs of labels by the same author. The R script for obtaining every unique assignment is given on the website: it consists of listing the assignments $\{(1,2),(1,3),(1,4),...,(1,12)\}$ and then for each of these 11, listing 9 other pairings (e.g., for (1,2) the list would be $\{(3,4),(3,5),...,(3,12)\}$), and then for each of these 9 a similar list of 7 formed by the remaining integers, and so on. Exhibit B.6 shows the distribution of the 10395 values, with the observed value of 0.4711 indicated. As reported in Chapter 25, there are no other pairings of the labels in the two-dimensional CA map which give a smaller distance, which makes the P-value associated with the result equal to 1/10395, i.e., $P < 0.0001$. A similar test conducted on the subset analyses of Exhibit 21.1 (consonants only) and 21.2 (vowels only) yielded 47 and 67 permutations to the left of the observed value, i.e., P-values of $48/10395 = 0.0046$ and $68/10395 = 0.0065$, respectively.

Permutation tests in CCA

CCA looks at a part of the full space of the response variables (usually species abundances in ecology) that is linearly related to a given set of explanatory variables (usually environmental variables) — see Chapter 24. But what if the responses have no relation to the explanatory variables? The inertia in the constrained space is a measure of the relationship, so this needs to be compared to a null distribution of constrained inertias when there is no relationship. This can be generated by randomly permuting the cases (rows) in the matrix of explanatory variables (or response variables). Having mixed up the rows, losing their connections with the rows in the response matrix, the CCA is repeated and the constrained inertia recalculated. Doing this 999 times, say (or however many you need to establish an accurate P-value), the observed value is situated in the distribution to see if it is unusually high. If it lies in the top 5% of observations, the relationship will be deemed statistically significant, and the P-value can be estimated by counting how many values in the permutation distribution are higher than the observed value (in this case

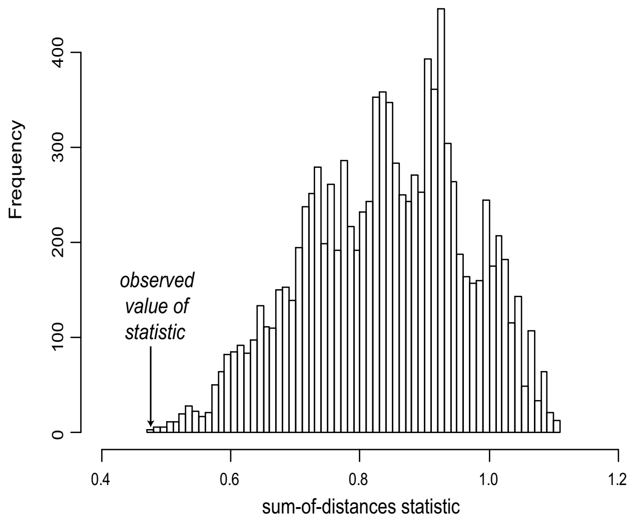


Exhibit B.6: *Exact null distribution of sum-of-distances statistic in the permutation test for testing randomness in the positions of pairs of texts by the same author in the CA map. The observed value is the second smallest out of 10395 possible values.*

the value has to be high to be significant). The **vegan** package incorporates this test, which can be obtained with the `anova()` function around the `cca()` analysis:

```
> anova(cca(bio,env))

Permutation test for cca under reduced model

Model: cca(X = bio, Y = env)
         Df  Chisq      F N.Perm  Pr(>F)
Model     3 0.2798 1.6696   1300 0.03462 *
Residual  9 0.5028
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The statistic used is not the inertia but a "pseudo-F" statistic, as if an analysis of variance is being performed (for more details, see the **vegan** documentation), but the important part of the printout is the P-value which shows that the F-statistic for the constrained space is significantly high (P=0.03462).

Fionn Murtagh's R *programs*

In his recent book *Correspondence Analysis and Data Coding with Java and* R (see Bibliographical Appendix), Fionn Murtagh gives many R scripts for CA and especially data recoding, all of which are available on the website `www.correspondances.info`. In particular, on pages 21–26 he describes a program for hierarchical clustering by Ward's method, with incorporation of weights, which is exactly what we needed in Chapter 15, but which is otherwise unavailable in R. Assuming you have been able to download the code from his website, and have read the table of data of Exhibit 15.3 as the data frame `food`, then the cluster analysis of the row profiles in Exhibit 15.5 can be achieved using Murtagh's `hierclust()` function as follows:

```
> food.rpro <- food /apply(food,1,sum)
> food.r <- apply(food,1,sum) / sum(food)
> food.rclust <- hierclust(food.rpro, food.r)
> plot(as.dendrogram(food.rclust))
```

XLSTAT One of the best commercial alternatives for performing all the analyses in this book, and more, is the suite of Excel add-on packages called XLSTAT (`www.xlstat.com`). The CA and MCA programs in XLSTAT now include the adjustments of inertia in MCA and the subset options in both CA and MCA. A program for CCA is also implemented, including a permutation test for testing whether the explanatory variables are significantly related to the principal axes of the constrained solution. Other multivariate analysis software in XLSTAT includes principal component analysis, factor analysis, discriminant analysis, cluster analysis, partial least squares and generalized Procrustes analysis. The programs are particularly easy to use because they function in an Excel environment. For example, to execute the CA on the `food` data set used above in the hierarchical clustering, click on the CA icon and select the table (with row and column labels) that you want to analyze:

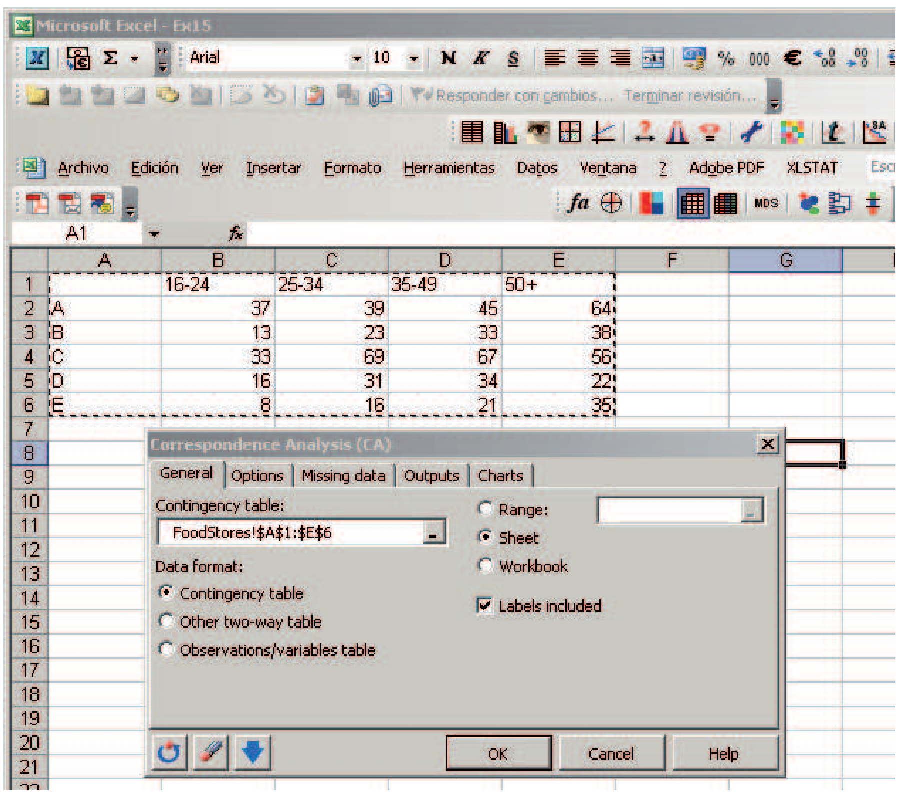


Exhibit B.7: *XLSTAT menu for executing CA on a table selected in Excel.*

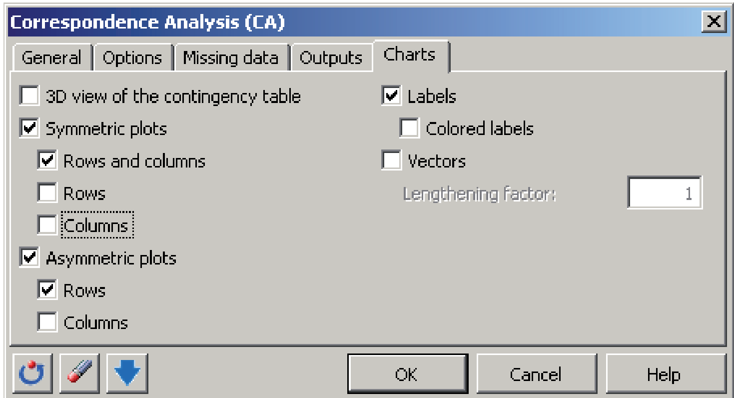


Exhibit B.8: *XLSTAT menu for selecting graphical options in CA.*

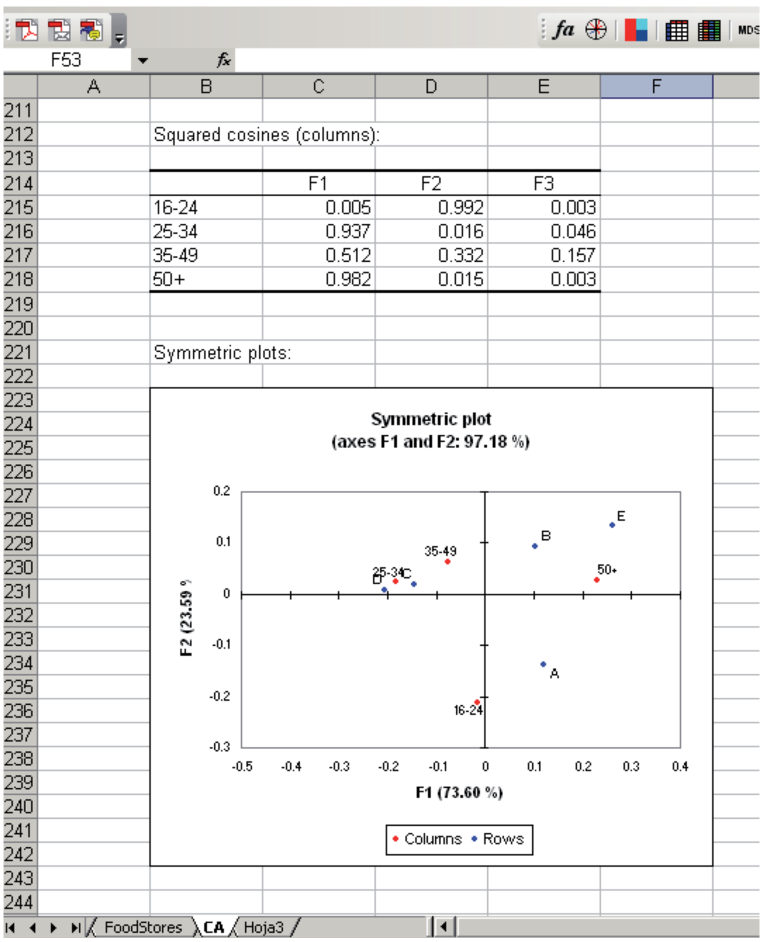


Exhibit B.9: *Part of the output of the CA program in XLSTAT, which is returned in a separate worksheet.*

There is an "Options" menu which allows selection of supplementary points or subsets, a "Missing data" menu for deciding what to do with missing values, and "Outputs" menu for selecting the various numerical tables (profiles, χ^2-distances, principal coordinates, standard coordinates, contributions, squared correlations, etc.) and a "Charts" menu which allows selection of the various maps — see Exhibit B.8, where the symmetric map of rows and columns is selected as well as the asymmetric map of the rows (i.e., `"rowprincipal"` option in the **ca** package). Exhibit B.9 shows part of the output.

In the cluster analysis module of XLSTAT it is possible to assign weights to points, so performing a Ward clustering on the profiles, weighted by their masses, would reproduce the cluster analyses of Chapter 15.

Graphical options

Producing a CA map with certain characteristics and which is ready for publication is not a trivial task. In this section we describe the three different technologies that were used in this book to produce the graphical exhibits.

LaTeX *graphics*

This book was typeset in LaTeX. LaTeX itself and various LaTeX macros can produce maps directly, without passing through another graphics package. Most of the maps produced in the first part of the book were produced using the macro package PicTeX. As an example, the following code, which is embedded in the LaTeX text of the book itself, produced Exhibit 9.2, the asymmetric map of the smoking data:

```
\beginpicture
\setcoordinatesystem units <2.5cm,2.5cm>
\setplotarea x from -2.40 to 1.70, y from -1.6 to 2.25
\accountingoff
\gray
\setdashes <5pt,4pt>
\putrule from 0 0 to 1.7 0
\putrule from 0 0 to -1.4 0
\putrule from 0 0 to 0 2.25
\putrule from 0 0 to 0 -1.6
\put {+} at 0 0
\black
\small
\put {Axis 1} [Br] <-.2cm,.15cm> at 1.70 0
\put {0.0748 (87.8\%)} [tr] <-.2cm,-.15cm> at 1.70 0
\put {Axis 2} [Br] <-.1cm,-.4cm> at 0 2.25
\put {0.0100 (11.8\%)} [Bl] <.1cm,-.4cm> at 0 2.25
\setsolid
\putrule from 1.3  -1.3  to 1.4  -1.3
\putrule from 1.3  -1.32 to 1.3  -1.28
\putrule from 1.4  -1.32 to 1.4  -1.28
\put {\it scale} [b] <0cm,.25cm> at 1.35 -1.3
\put {0.1} [t] <0cm,-.2cm> at  1.35 -1.3
\multiput {$\bullet$} at
 0.06577  0.19373
-0.25896  0.24330
```

```
 0.38059  0.01066
-0.23295 -0.05775
 0.20109 -0.07891
 /
\sf
\put {SM} [l] <.15cm,0cm> at  0.06577  0.19373
\put {JM} [r] <-.15cm,0cm> at -0.25896  0.24330
\put {SE} [bl] <.15cm,0cm> at  0.38059  0.01066
\put {JE} [r] <-.15cm,0cm> at -0.23295 -0.05775
\put {SC} [tl] <.15cm,0cm> at  0.20109 -0.07891
\gray
\multiput {$\circ$} at
 1.4384  0.3046
-0.3638 -1.4094
-0.7180 -0.0735
-1.0745  1.9760
 /
\sl
\put {none} [b] <0cm,.2cm> at    1.4384  0.3046
\put {light} [b] <0cm,.2cm> at  -0.3638 -1.4094
\put {medium} [T] <0cm,-.3cm> at -0.7180 -0.0735
\put {heavy} [b] <0cm,.2cm> at  -1.0745  1.9760
\black
\endpicture
```

Comparing the above code with Exhibit 9.2 itself should be enough for you to see how each line and each character is laboriously placed in the plotting area. One advantage of this approach, however, is that once you have set the units on the horizontal and vertical coordinate axes to be the same (`2.5cm` per unit in the example above), then you are assured that the aspect ratio of 1 is perfectly preserved in the eventual result.

Excel graphics

Since many of the new maps produced for this second edition were made in Excel using XLSTAT, many exhibits are produced in this style, for example those in Chapters 17–19. In Excel, a certain amount of trimming of the maps was done, redefining the maxima and minima on the axes, and also stretching the graph window vertically or horizontally until the aspect ratio appeared correct. The graphic was then copied as a metafile and pasted into *Adobe Illustrator*, where further trimming and character redefinition was performed. The aspect ratio becomes slightly deformed when copying into *Adobe Illustrator*, with a vertical unit appearing slightly longer than a horizontal unit, so some resizing is necessary at this stage as well. The graphic was then saved as an *Encapsulated PostScript* (EPS) file and then included in the LaTeX file using the `\includegraphics` instruction, for example:

```
\begin{figure}[h]
\center{\includegraphics[width=10cm,keepaspectratio]{Ex18_5.eps}}
\caption{\sl MCA map of Burt matrix of four questions on women
         working, showing first and second dimensions;
         total inertia = 1.145, percentage inertia in map: 65.0\%.}
\end{figure}
```

R *graphics* Finally, many maps were also produced in R, for example all the graphics of Chapter 25. These were also copied as metafiles and pasted into *Adobe Illustrator*, fine-tuned and then saved as an EPS file for including in the text.