Technical note:
Optimal partitioning of soil transects with R

D G Rossiter

October 16, 2013

Contents

This note describes procedures to identify soil boundaries along a transect where soil samples have been taken at regular intervals. It is based on the work of Webster [3–6] and used as an example in the text of Davis [1, pp. 234-243].

Two partitioning methods are described by Webster [4]: Split moving-window (SMW) (§2) and Maximum Level Variance (MLV) (§4). Only the first is developed in this technical note.

> **Note:** The code in this document was tested with R version 3.0.1 (2013-05-16) and packages from that version or later running on Mac OS X 10.7.5. The text and graphical output you see here was written as a NoWeb file, including both R code and regular LaTeX source, and then run through the excellent `knitr` package Version: 1.4.1 [7] on R to generate a LaTeX document that includes formatted R code input and the results of running the code, both text results and graphs. Then the LaTeX document was compiled into the PDF version you are now reading. If you run the R code from this document, your output may be slightly different on different versions and on different platforms.

1 Example transect

We illustrate optimal partitioning with a transect of 42 stations spaced at 25 m intervals in an undisturbed forest near Juruena, Mato Grosso in the southern Amazon, collected by Steven Jirka of Cornell University as part of the LBA[1] project. Both field and laboratory measurements were made; for illustrative purposes we use only the sand and clay content, measured in g kg^{-1}, of three layers. These are organized as an R data frame with the rows being the stations and the columns the variables. The sample dataset is provided as a comma-separated value file `tr.csv` and is reproduced in Appendix §A.

Task 1 : Read the transect from the CSV file into R and display its structure. •

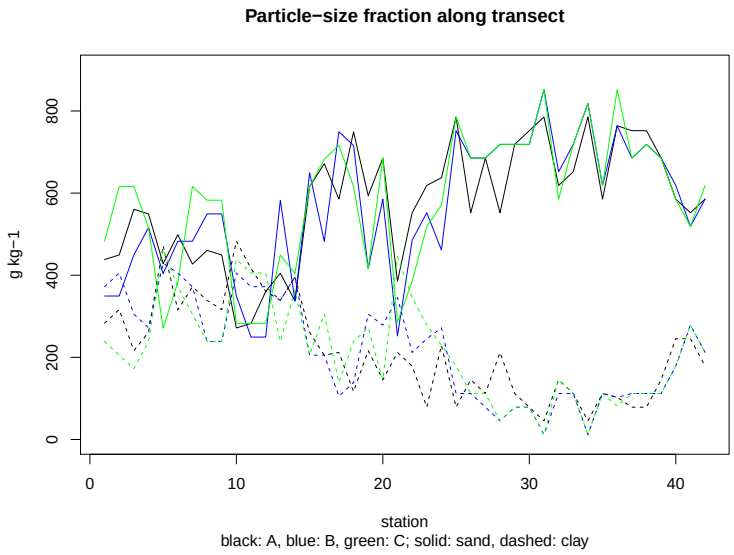
```
> transect <- read.csv("tr.csv")
> str(transect)

'data.frame': 42 obs. of  6 variables:
 $ sand.A: num  438 449 560 549 428 ...
 $ clay.A: num  283 316 216 261 472 ...
 $ sand.B: num  349 349 449 516 404 ...
 $ clay.B: num  372 405 305 272 429 ...
 $ sand.C: num  483 616 616 516 271 ...
 $ clay.C: num  239 205 172 239 463 ...
```

Task 2 : Plot the six variables along the transect. •

```
> plot(transect$sand.A, type="l", ylim=c(0,900),
+      main="Particle-size fraction along transect",
+      xlab="station", ylab="g kg-1",
+      sub="black: A, blue: B, green: C; solid: sand, dashed: clay")
> lines(transect$sand.B, type="l", col="blue")
```

[1] Large-scale Biosphere-Atmosphere Experiment in Amazonia, see http://earthobservatory.nasa.gov/Study/LBA/

```
> lines(transect$sand.C, type="l", col="green")
> lines(transect$clay.A, type="l", col=1, lty=2)
> lines(transect$clay.B, type="l", col="blue", lty=2)
> lines(transect$clay.C, type="l", col="green", lty=2)
```

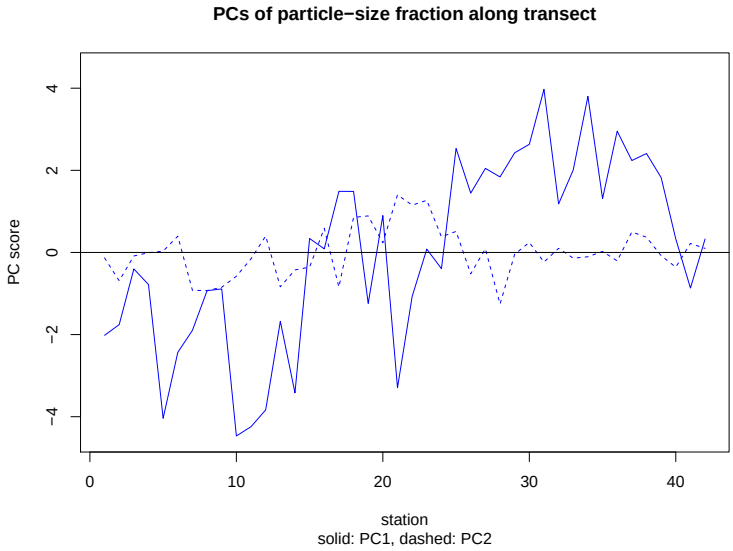


It is evident that there is *spatial autocorrelation*, i.e., nearby stations are likely to be similar. It also appears that the sequence can be broken up into several more homogeneous sections. A clear difference is that the beginning of the sequence has fairly equal sand and clay, whereas after about station 15 there is much more sand than clay. However, there is a break in this pattern near station 21, where again the clay increases.

It is also evident that there is much redundant information; this is because an increase in one particle-size fraction must be compensated by a decrease in one of the others. Further, the sand and clay contents in the three horizons at each station are similar. We can check the redundancy and reduce the number of variables with principal components analysis (PCA). Although all the variables are in the same units of measure, using *standardized* components gives equal weight to all variables, regardless of their ranges.

Task 3 : Compute the standardized principal components of the six variables, and plot the scores of the first two PCs along the transect. •

```
> pc <- prcomp(transect, center=T, scale=T, retx=T)$x
> plot(pc[,1], type="l", col="blue",
+      main="PCs of particle-size fraction along transect",
+      xlab="station", ylab="PC score",
+      sub="solid: PC1, dashed: PC2",
+      ylim=c(-4.5, 4.5))
> lines(pc[,2], type="l", lty=2, col="blue")
> abline(h=0)
```



The graph of the PCs shows that much more variance is captured by the first PC than by the second, since the PC scores are much larger for PC1 than PC2. The first PC has many sharp peaks, but there is nonetheless a clear trend from negative to positive scores moving along the transect.

For both the original variables and the PCs, the question is, where to put boundaries; in particular, which algorithm will find the ones that we believe correspond to real soil type differences?

2 Split moving-window

The split moving-window (SMW) approach computes the contrast between two halves of a window as it moves along the transect, and reports this contrast. The higher the contrast, the more likely it is that the station at which the window is split is a soil boundary. The contrast is measured by the squared Mahalanobis distance between halves of the window. This is defined as:

$$D^2 = (\overline{\mathbf{x}}_s - \overline{\mathbf{x}}_d)^T \mathbf{W}^{-1} (\overline{\mathbf{x}}_s - \overline{\mathbf{x}}_d) \quad (1)$$

where $\overline{\mathbf{x}}_s$ is the average vector of the *left* half, $\overline{\mathbf{x}}_d$ is is the average vector of the *right* half, and W is the pooled within-half variance-covariance matrix, after correcting for the respective means. The squared differences between the mean vectors of each window half are thus corrected in two ways[2]:

1. Variables with lower pooled within-half residual variances are given more weight;
2. Two variables with positively-correlated residuals are not "double-counted", that is, a difference with the same sign in both axes is corrected downwards; by contrast, a difference with opposite sign is emphasized (because it is unexpected).

[2] See numeric example in Appendix D

The variables are weighted by their discriminating power in the particular window; this changes as the window moves.

If **W** is replaced by the identity matrix **I**, this is equivalent to the squared Euclidean distance:

$$E^2 = (\overline{\mathbf{x}}_s - \overline{\mathbf{x}}_d)^T \mathbf{I} (\overline{\mathbf{x}}_s - \overline{\mathbf{x}}_d) \tag{2}$$

where all axes are weighted equally and there is no correction for discriminating power in the window, either for different residual variance or for covariances.

> **Note:** If the pooled within-half variance-covariance matrix **W** is singular, it can not be inverted and thus the Mahalanobis distance is not defined; however the Euclidean distance is zero, and it makes sense to also make the Mahalanobis distance zero.

2.1 Functions for the SMW method

I have written three R functions to implement the SMW method; they are contained in file `smw.R` and loaded with the `source` function. The source code is also shown in Appendix §B.

```
> source("smw.R")
> ls(pattern = "smw.*")

[1] "smw.dw"    "smw.graph" "smw.pc"
```

These must be called in the following order:

1. `smw.pc` : (optional) extract PCs;
2. `smw.dw` : SMW analysis of PCs or original variables;
3. `smw.graph` : visualise results of SMW. analysis

2.2 Options for the SMW method

The analyst must make several important choices that have a large effect on the presumed boundaries:

1. Whether to use the original variables or some number of their standardized principal components;
2. Whether to use Euclidean or Mahalanobis distances;
3. How wide a window to use;
4. Window parity, i.e., whether to use an odd or even window;
5. Whether to use a “mullion”, i.e., leave out some stations near the middle of the window to allow for a more diffuse boundary;
6. What distance in feature space (as a proportion of the maximum in the whole transect) probably signals a boundary, and should be reported?

2.2.1 Standardized principal components

Function `smw.pc` computes the *standardized* principal components from the *correlation* matrix[3] and replaces the original variables with one or more synthetic variables that are uncorrelated and arranged in descending order of the amount of variance of the original data that they explain. This corrects for different measurement scales, and also for correlation between variables (redundancy). The default is to extract two components; this can be over-ridden with the `n.pc` named argument.

```
> pc <- smw.pc(transect)

[1] "PCA: selected components explain 0.923 of the variance"
[1] "Loadings for the first 2 components:"
           PC1      PC2
sand.A  0.4061  0.47370
clay.A -0.3968 -0.62764
sand.B  0.4133 -0.25254
clay.B -0.4149  0.02935
sand.C  0.4095 -0.36782
clay.C -0.4087  0.42631

> str(pc)

'data.frame': 42 obs. of  2 variables:
 $ PC1: num  -2.019 -1.757 -0.401 -0.782 -4.039 ...
 $ PC2: num  -0.13327 -0.68999 -0.08597 -0.00673 0.03252 ...
```

Here we see that the two PCs explain most of the variance.

The PCs can be interpreted by examining the *loadings*. These reveal which soil variables on this transect are combined into each component. Interpretation in this case is easy. The first component is associated with high sand (positive loading) and low clay (negative loading), i.e. with coarser textures, throughout the profile; the second component is associated with higher sand in the topsoil than subsoil, and the reverse for clay. In other words, the first component is the overall texture and the second is textural contrast.

The structure returned by `smw.pc` is a data frame with the principal component scores for each station on the transect; these replace the original variables.

2.2.2 Original variables

Distances can be computed in the multivariate space of the original variables, rather than in the space of their PCs. To compute the Euclidean distances between original variables, these must be scaled: corrected individually to zero mean and unit variance; computation of Mahalanobis distances implicitly scales them. A data frame can be scaled with the `scale` function; the result must be converted back into a data frame:

```
> tr.scale <- data.frame(scale(transect))
> str(tr.scale)

'data.frame': 42 obs. of  6 variables:
 $ sand.A: num  -0.96 -0.882 -0.104 -0.181 -1.031 ...
```

[3] rather than the *variance-covariance* matrix

```
 $ clay.A: num  0.5823 0.8658 0.0153 0.3933 2.1888 ...
 $ sand.B: num  -1.341 -1.341 -0.722 -0.309 -1.002 ...
 $ clay.B: num  1.228 1.496 0.692 0.424 1.689 ...
 $ sand.C: num  -0.616 0.214 0.214 -0.409 -1.936 ...
 $ clay.C: num  0.2005 -0.0768 -0.3542 0.2005 2.0642 ...
```

Note all the variables have zero mean and unit variance, and the covariance matrix is a correlation matrix:

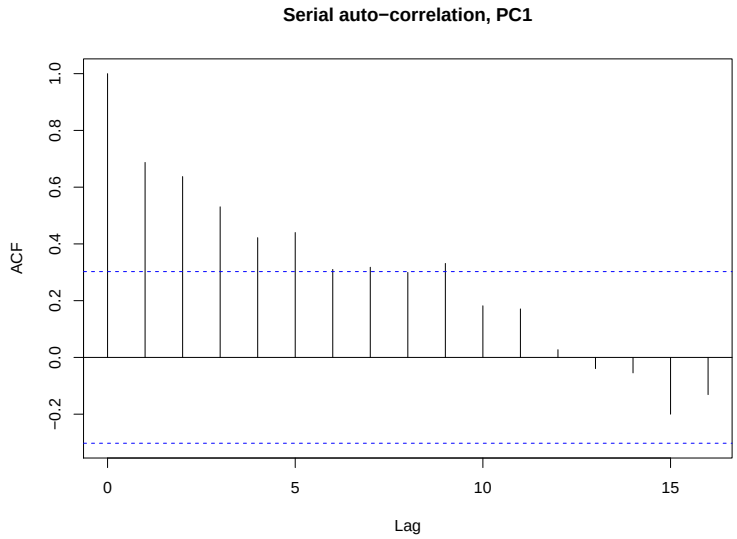
```
> round(apply(tr.scale, 2, mean), 4)

sand.A clay.A sand.B clay.B sand.C clay.C
     0      0      0      0      0      0

> var(tr.scale)

        sand.A  clay.A  sand.B  clay.B  sand.C  clay.C
sand.A  1.0000 -0.9016  0.8317 -0.8175  0.8237 -0.7670
clay.A -0.9016  1.0000 -0.7596  0.8417 -0.7541  0.7697
sand.B  0.8317 -0.7596  1.0000 -0.9064  0.8661 -0.8600
clay.B -0.8175  0.8417 -0.9064  1.0000 -0.8221  0.8581
sand.C  0.8237 -0.7541  0.8661 -0.8221  1.0000 -0.9129
clay.C -0.7670  0.7697 -0.8600  0.8581 -0.9129  1.0000
```

2.2.3 Window width

The narrower the window, the narrower the soil unit that can be distinguished, but the more likely are spurious boundaries (and more likely that the system will be computationally singular). Webster [4] recommends 2/3 of the expected distance between boundaries, in this case the expected width of a soil unit across the transect.

This can be estimated by *auto-correlation* of one or more variables, usually the first PC. By default, the `acf` ("Auto- and Cross-Covariance and -Correlation Function Estimation") function of the default `stats` package displays a graph of autocorrelation by lag; the results can also be printed on the console:

```
> acf(pc[, 1], main = "Serial auto-correlation, PC1")
> print(acf(pc[, 1], plot = FALSE))


Autocorrelations of series 'pc[, 1]', by lag

     0      1      2      3      4      5      6      7      8      9
 1.000  0.687  0.637  0.531  0.422  0.440  0.310  0.318  0.300  0.331
    10     11     12     13     14     15     16
 0.182  0.171  0.027 -0.039 -0.055 -0.200 -0.131
```



The auto-correlation decreases to zero near lag 12, suggesting that there are about three boundaries in this 42-station transect; the suggested window size is then 8.

If the named argument `wid` is not supplied to smw.dw by the user, it is computed as a quarter of the total length of the transect.

2.2.4 Window parity

If the window width n is *even*, the transect is split at station $l + n/2$, where l is the left-most station of the window, and any inferred boundary is there; i.e., neither half of the window uses the data from that station. For example, with window width 8, the first window (at the beginning of the transect) is from station $1 \ldots 9$, the middle of which is station5. The left half of the split window includes stations $1 \ldots 4$ and the right half $6 \ldots 9$; station 5 is the potential boundary. If the window width n is *odd*, the transect is split half-way between stations $l + (n-1)/2$ and $l + (n+1)/2$. For example,with window width 7, the first window (at the beginning of the transect) is from station $1 \ldots 8$. The left half of the split window includes stations $1 \ldots 4$ and the right half $5 \ldots 8$; the potential boundary is between stations 4 and 5. This is reported as station 4 but is actually located at position 4.5.

2.2.5 Mullion

Especially for wide windows, we may accept a more diffuse boundary; in fact, the 'boundary' we are looking for could be identified as a separate soil unit if a narrower window were used. In this case we may increase the discriminating power of the method by leaving out some observations on either side of the potential boundary; this is the so-called "mullion". In this implementation of SMW the user-specified mullion (if any) applies to both halves of the window, i.e., it is the number of stations to omit. The default for the `mull` named argument to `smw.dw` is 0; note

that the middle station is omitted from an even transect in any case. It doesn't make sense for the mullion to be more than 1/4 of the window half width, and the program checks for this. In the present example, with a suggested window width of 8, the mullion can be 0 or 1.

2.2.6 What Mahanalobis distance to report?

The `smw.dw` function returns the distances for each possible boundary; however in its printed summary it only reports those above a certain threshold, which is a proportion of the maximum feature-space distance found in the transect. By default this is 0.25; to see more boundaries set to a lower value with the `p.mmd` named argument, e.g., `p.mmd=0.1`. This does not affect the calculation, only the printed output.

2.3 Analyzing the example transect

The core of the SMW analysis is performed by the `smw.dw` function. This has as its first argument the data frame representing the transect (probably transformed to PCs) and optional arguments for the window width, mullion, and boundary sensitivity.

2.3.1 Using PCs

The SMW method is often applied to the first few PCs, since these hold most of the information about the soil properties in compressed form. However, they are *not* guaranteed to be the most discriminating.

TASK 4 : Analyze the transect using the first two standardized PCs and a window width of 8. •

```
> d <- smw.dw(pc, wid = 8)

[1] "Window: 8 stations; mullion:  0"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station     D2
1      15 25.489
2      14 24.923
3       7 16.218
4      25 15.564
5       6  7.978
6      24  7.788
7      16  7.528
8      26  6.774
9      38  6.584
```

There are three main suggested boundaries: at stations 15, 7 and 25; these are somewhat diffuse, because the alternative adjacent stations (14 or 16), 6 and (24 or 26) are also identified. A less likely boundary is found towards the right of the transect at station 38.

TASK 5 : Analyze the transect using the first two standardized PCs and a window width of 7, i.e., an odd window size one unit smaller. •

Using the next smaller odd window width raises the absolute distances and sharpens the same three boundaries, here found at positions 6.5, 14.5, and 24.5 (rather than at stations 7, 15 and 25 with window width 8). Station 38 is not found at this reporting threshold.

TASK 6 : Analyze the transect using the first two standardized PCs and a window width of 10, i.e., slightly wider. •

```
> d <- smw.dw(pc, wid = 10)

[1] "Window: 10 stations; mullion:  0"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station     D2
1      15 28.024
2      14 14.356
3      24 10.196
4      25  8.823
5      16  8.262
6      18  7.805
7      26  7.409
```

Widening the window misses the boundaries near stations 7 and 38 (because they are closer than the half-width to the ends) but otherwise agrees with the previous results.

TASK 7 : Analyze the transect using the first two standardized PCs, a window width of 10, and a one-unit mullion. •

```
> d <- smw.dw(pc, wid = 10, mull = 1)

[1] "Window: 10 stations; mullion:  1"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station     D2
1      16 31.319
2      15 22.126
3      26 17.774
4      17 17.677
5      14 15.682
6      13 11.422
7      24  9.506
8      25  8.639
```

Adding a mullion to this wider window finds the same boundary many times, suggesting that the sharper analysis without a mullion was more appropriate.

Using a narrow window changes the picture radically.

TASK 8 : Analyze the transect using the first two standardized PCs and a window width of 6, i.e., narrower. •

```
> d <- smw.dw(pc, wid = 6)

[1] "Window: 6 stations; mullion:  0"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    D2
1       6 476.4
```

The narrow window finds a very sharp contrast at station 6 which overwhelms all the others. The boundary near the right of the transect (39) is now prominent rather than secondary. Stations 25 and 14 (and their neighbours) are also found. As the window gets smaller, the results tend to be more erratic and indeed the system may be computationally singular at some window positions.

TASK 9 : Analyze the transect using only the first standardized PCs and the recommended window width of 8. •

```
> d <- smw.dw(as.data.frame(pc$PC1), wid = 8)

[1] "Window: 8 stations; mullion:  0"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station     D2
1      14 20.699
2      15 10.972
3      25  9.403
4      24  5.239
```

With this suggested window width (8), using only one PC lowers the absolute distances a bit, and finds the boundaries at station 14 (shifted from 15 as suggested with two PCs) and 25, but misses the boundary near station 7. This latter boundary must therefore be associated with changes in PC2 (textural contrast) rather than PC1 (overall texture).

TASK 10 : Analyze the transect using only the *second* standardized PCs and the recommended window width of 8. •

```
> d <- smw.dw(as.data.frame(pc$PC2), wid = 8)

[1] "Window: 8 stations; mullion:  0"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    D2
1      25 8.841
2       7 7.066
3       6 6.570
4      18 5.269
5      17 4.818
6      24 4.575
7      26 3.976
8      23 3.256
9      19 2.329
```

And indeed, looking for boundaries with PC2 only we see station 25 (with 23, 24, and 26) but then station 7 (with 6); a new boundary is suggested at station 18 (with 17 and 19) which may also be associated with textural contrast.

Notes on use of the Mahalonobis distance with principal components
The Mahalonobis distance uses the covariance matrix of the pooled observations (corrected for their means) in both halves. This would be `diag(n.pc)` if there were only one window (since there is by definition no correlation between PCs), with the variances on the diagonal pro-

portional to the eigenvalues within the selected set. The PCs would be weighted according to their importance. However, for each pooled window this will be somewhat different; the off-diagonals will usually not be zero (there may be correlation between the PCs for just these observations) and the proportions will be different from that for the whole transect.

2.3.2 Using original variables

The SMW method can also be applied to original variables, either unscaled or scaled. A disadvantage is that more variables are needed to capture the same information as the PCs, so that wider windows are usually needed to avoid ill-conditioned covariance matrices at some window positions. For Mahalanobis distances, unscaled and scaled variables give the same

---

TASK 11 : Analyze the transect using the original variables, and with the first two PCs, both using a window size 9. •

> **Note:** Using all six PCs would give exactly the same result as original variables, since the information content is the same.

```
> d <- smw.dw(transect, wid = 9)

[1] "Window: 9 stations; mullion:  0"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary to the right of station"
  station     D2
1       6 298.92
2      24 208.50
3      25 160.12
4       9 113.35
5      35 108.06
6      14 105.03
7      28 101.51
8      17  77.56

> d <- smw.dw(pc, wid = 9)

[1] "Window: 9 stations; mullion:  0"
[1] "Distances up to 0.25 of the maximum"
[1] "Boundary to the right of station"
  station     D2
1      14 22.953
2      15  9.811
3      24  8.318
4      17  7.571
5      25  7.160
```

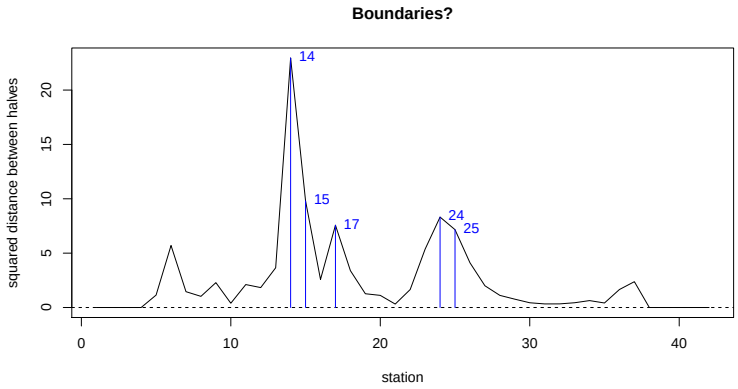
Both methods suggest a boundary near stations 24 and 25, but using the original variables suggests boundaries near stations 6, 9 and 35 as well, whereas using the first two PCs suggests a boundary near station 14; this is also found with the original variables but with lower priority.

2.4 Visualizing the boundaries

The results returned by `smw.dw` can be visualized with the `smw.graph` function.

TASK 12 : Display the boundaries for the analysis using two PCs and a window size of 9, as computed just above. •

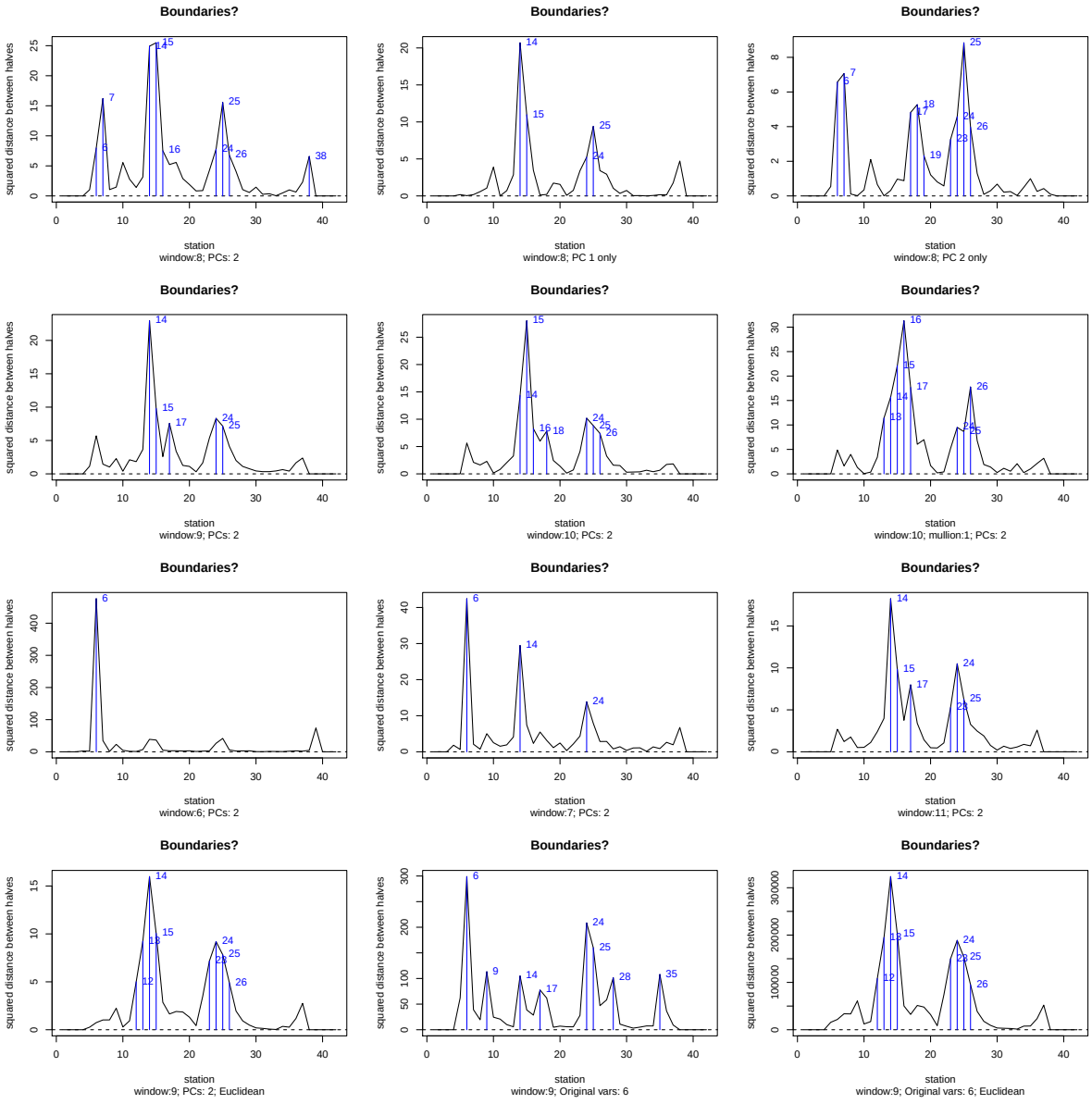
```
> smw.graph(d)
```



We can compare various approaches visually by writing a small script and putting the graphs in one frame; this repeats the examples of §2.3 with a few more variations. The following code can be entered at the R command line but is more easily placed in a file and loaded with `source`.

The results are quite variable: most approaches find the boundary near positions 14–16 but some miss this entirely; again the boundary near position 24–26 is often found as the second-most important but not always. The extreme case is given by window width 6, two PCs, which only finds a boundary at position 6; using a window width 7 with these two PCs still has this as the most important but does find positions 14 and 24. This example shows the importance of (1) selecting relevant variables whose transition across the transect should be used to defined boundaries; (2) deciding whether to combine these as standardized PCs or use original variables; (3) selecting a window width (and eventually a mullion) which corresponds to the width of the boundary in nature.

```
> par(mfrow=c(4,3))
> d <- smw.dw(pc, wid=8); smw.graph(d, text="window:8; PCs: 2")
> d <- smw.dw(as.data.frame(pc$PC1), wid=8); smw.graph(d, text="window:8; PC 1 only")
> d <- smw.dw(as.data.frame(pc$PC2), wid=8); smw.graph(d, text="window:8; PC 2 only")
> d <- smw.dw(pc, wid=9); smw.graph(d, text="window:9; PCs: 2")
> d <- smw.dw(pc, wid=10); smw.graph(d, text="window:10; PCs: 2")
> d <- smw.dw(pc, wid=10, mull=1); smw.graph(d, text="window:10; mullion:1; PCs: 2")
> d <- smw.dw(pc, wid=6); smw.graph(d, text="window:6; PCs: 2")
> d <- smw.dw(pc, wid=7); smw.graph(d, text="window:7; PCs: 2")
> d <- smw.dw(pc, wid=11); smw.graph(d, text="window:11; PCs: 2")
> d <- smw.dw(pc, wid=9, ident=T); smw.graph(d, text="window:9; PCs: 2; Euclidean")
> d <- smw.dw(transect, wid=9); smw.graph(d, text="window:9; Original vars: 6")
> d <- smw.dw(transect, wid=9, ident=T); smw.graph(d, text="window:9; Original vars: 6; Euclidean")
> par(mfrow=c(1,1))
```



3 When SMW fails to find a solution

The core of the SMW algorithm is the between-halves distance calculation; this is always possible using Euclidean distance (i.e., a diagonal variance-covariance matrix as in Equation 2) but if using Mahalanobis distance (as in Equation 1) with non-zero off-diagonals, the matrix may be singular. In that case the `smw.dw` function will report something like:

```
"Position 3: Singular pooled covariance matrix; not a boundary"
```

This implies a non-stable analytical context. Among the causes are:

- the window is too narrow, so the two halves are in fact quite similar; after subtracting the respective window half means, the pooled covariance matrix may have diagonal elements close to zero;
- the chosen variables do not in fact differ much at the window width; i.e., the variables are not discriminating. This is quite likely for higher PCs, which can be pure noise (explaining none of the true variability. The solution here is to only use the first few PCs (the ones contributing most to the variance); if using original variables use only those that show clear boundaries, not those that seem to fluctuate randomly.

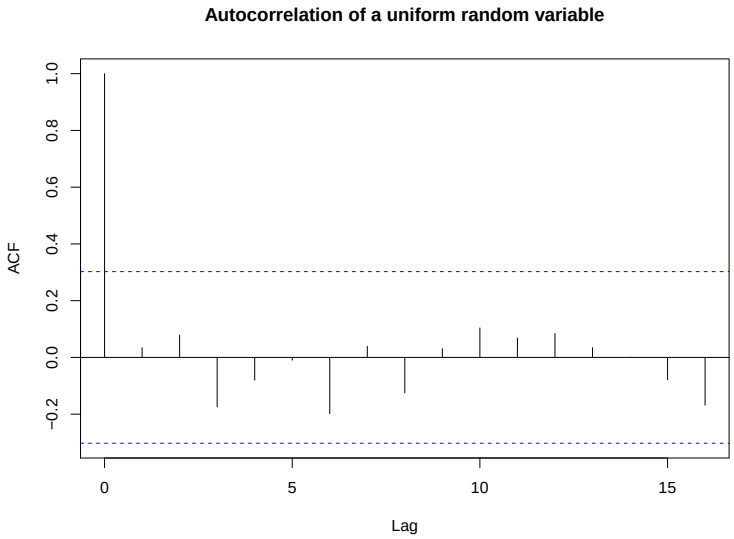
Both of these situations can be anticipated by examining the structure of the local spatial correlation with `acf`. We have already seen how to use it to set an appropriate window width. If the autocorrelation is already low at short lags this is evidence that the variable has no structure, and any boundaries that the SMW algorithms may find are an artefact and do not represent real boundaries.

Task 13 : Compute and display the autocorrelation of a uniform random variable along a transect of length 42. •

The `runif` function returns a vector of uniformly-distributed random numbers on $[0 \ldots 1]$.

> **Note:** We use `set.seed` so your results will be identical to these notes; in practice you would not use this.

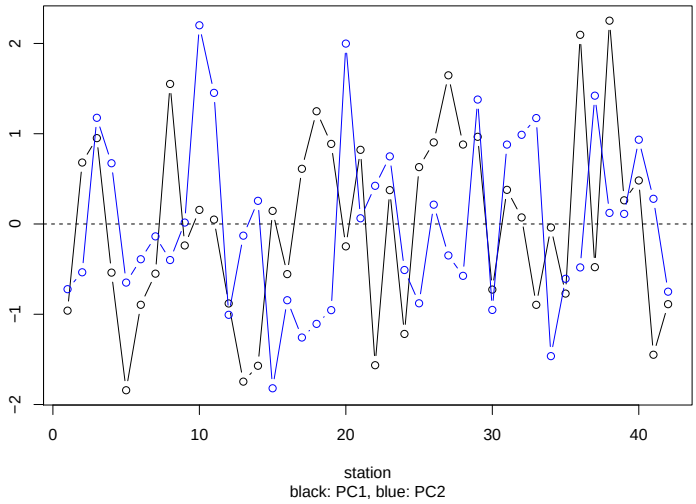
```
> set.seed(321)
> acf(runif(42), main = "Autocorrelation of a uniform random variable")
> acf(runif(42), plot = FALSE)


Autocorrelations of series 'runif(42)', by lag

     0      1      2      3      4      5      6      7      8      9
 1.000  0.064 -0.169 -0.001  0.098 -0.179 -0.242  0.144  0.127  0.097
    10     11     12     13     14     15     16
-0.036  0.122  0.111 -0.197 -0.292  0.000  0.040
```



Note that even at the first lag the autcorrelation is almost zero and well within the confidence limits around 0 (shown by the dashed blue lines).

TASK 14 : Build a dataframe with two uniformly-random variables and plot these along the transect. •

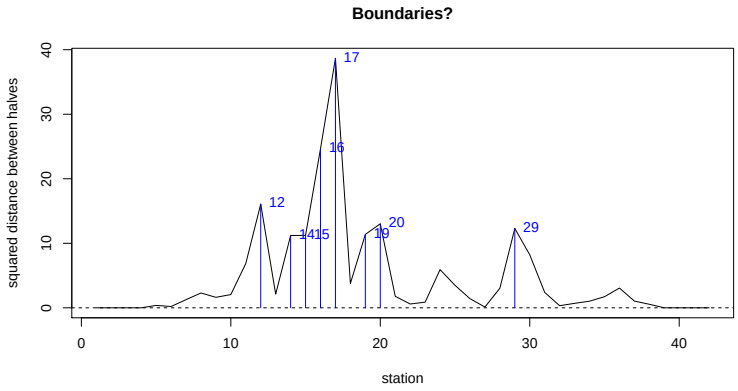
```
> tr <- data.frame(x=runif(42), y=runif(42))
> tr.pc <- smw.pc(tr)

[1] "PCA: selected components explain 1 of the variance"
[1] "Loadings for the first 2 components:"
      PC1    PC2
x -0.7071 0.7071
y  0.7071 0.7071

> plot(tr.pc$PC1, type="b",
+      main="",
+      xlab="station", ylab="",
+      sub="black: PC1, blue: PC2")
> lines(tr.pc$PC2, type="b", col="blue")
> abline(h=0, lty=2)
```



There is evidently no structure. However, maybe the automatic procedure will find structure which arose purely by chance.

TASK 15 : Attempt to find its boundaries. •

```
> d <- smw.dw(tr.pc, wid = 8)
> smw.graph(d)
```



This illustrates the danger of relying on automatic procedures.

4 Maximum Level Variance

The Maximum Level Variance (MLV) approach [4], based on the work of Hawkins and Merriam [2], considers the whole transect together and looks for the best way to divide it into a user-specified number of segments, so that the pooled within-segment variance is as small as possible. The advantage of MLV is that it can not be "tuned" with window width and mullion; only one classification can be found. In addition, it will find diffuse boundaries if they otherwise separate very contrasting zones, whereas SMW must be given a wide enough window and possibly a

mullion to find these. MLV can work on either Euclidean or Mahalanobis distances, and the classification can be stopped at any number of groups; that is, the user can specify the number of expected boundaries.

I have not yet implemented this; the mathematics are presented in the appendix to Webster [4].

References

[1] J C Davis. *Statistics and data analysis in geology*. John Wiley & Sons, New York, 3rd edition, 2002. ISBN 0-471-17275-8. 1

[2] D M Hawkins and D F Merriam. Zonation of multivariate sequences of digitized geologic data. *Mathematical Geology*, 6:263–269, 1974. 16

[3] R Webster. Automatic soil-boundary location from transect data. *Mathematical Geology*, 5:27–37, 1973. 1

[4] R Webster. Optimally partitioning soil transects. *Journal of Soil Science*, 29:388–402, 1978. 1, 6, 16, 17

[5] R Webster. DIVIDE, a FORTRAN IV program for segmenting multivariate one-dimensional spatial series. *Computers & Geosciences*, 6 (1):61–68, 1980.

[6] R Webster and I F T Wong. A numerical procedure for testing soil boundaries interpreted from air photographs. *Photogrammetria*, 24: 59–72, 1969. 1

[7] Yihui Xie. knitr: Elegant, flexible and fast dynamic report generation with R, 2011. URL http://yihui.name/knitr/. 1

A Example transect

```
"sand.A","clay.A","sand.B","clay.B","sand.C","clay.C"
438.24,283.1,349.36,371.99,482.69,238.66
449.36,316.43,349.36,405.32,616.02,205.33
560.46,216.44,449.36,305.32,616.02,171.99
549.35,260.88,516.02,271.99,516.02,238.66
428.02,471.98,404.02,429.32,270.7,462.65
498.69,315.54,482.69,405.32,382.69,371.99
427.13,371.99,482.69,371.99,616.02,305.32
460.47,338.65,549.35,238.66,582.68,238.66
449.36,316.43,549.35,238.66,582.68,238.66
271.58,483.09,349.36,405.32,282.7,438.65
282.7,416.43,249.36,371.99,282.7,405.32
360.47,360.87,249.36,371.99,282.7,405.32
404.91,338.65,582.68,338.65,449.36,238.66
337.36,395.98,337.36,395.98,404.02,362.65
616.02,260.88,649.35,205.33,616.02,205.33
671.57,205.33,482.69,205.33,682.68,305.32
585.35,211.99,749.34,105.33,716.01,138.66
749.34,116.44,716.01,138.66,616.02,238.66
593.79,216.44,416.02,305.32,416.02,271.99
685.35,145.33,585.35,278.66,685.35,145.33
385.36,211.99,252.03,345.32,285.36,445.32
552.02,178.66,485.35,211.99,385.36,345.32
618.68,78.66,552.02,245.32,518.69,278.66
637.35,229.32,461.35,271.99,570.68,229.32
785.34,78.66,752.01,112,785.34,178.66
552.02,145.33,685.35,112,685.35,112
685.35,112,685.35,78.66,685.35,112
552.02,211.99,718.68,45.33,718.68,45.33
718.68,112,718.68,78.66,718.68,78.66
752.01,78.66,718.68,78.66,718.68,78.66
785.34,45.33,852.01,12,852.01,12
618.68,145.33,652.01,112,585.35,145.33
652.01,112,718.68,112,718.68,112
785.34,45.33,818.67,12,818.67,12
585.35,112,618.68,112,618.68,112
764.01,102.66,764.01,102.66,852.01,81.33
752.01,78.66,685.35,112,685.35,112
752.01,78.66,718.68,112,718.68,112
685.35,145.33,685.35,112,685.35,112
585.35,245.32,618.68,178.66,585.35,178.66
552.02,245.32,518.69,278.66,518.69,278.66
585.35,178.66,585.35,211.99,618.68,211.99
```

B Split moving-window functions

```
> source("smw.R")
> ls(pattern = "smw.*")

[1] "smw.dw"    "smw.graph" "smw.pc"
```

B.1 PCA

```
> print(smw.pc)

function(transect, n.pc=2)
  {
                # can't ask for more PCs than variables
    n.pc <- min(n.pc, length(transect[1,]))
                # standardized PCs with scores
    pc <- prcomp(transect, center=T, scale=T, retx=T)
                # frame to return, with named synthetic variables
    pcs <- as.data.frame(pc$x[,1:n.pc])
                # print some diagnostics
    print(paste("PCA: selected component",
                ifelse(n.pc==1,"","s"),
                " explain",
                ifelse(n.pc==1,"s "," "),
                round(sum((pc$sdev^2/(sum(pc$sdev^2)))[1:n.pc]),3),
                " of the variance", sep=""))
    print(paste("Loadings for the first",
                ifelse(n.pc==1,"",paste("",n.pc)),"component",
                ifelse(n.pc==1,"","s"),":", sep=""))
    print(pc$rotation[,1:n.pc])
                # return frame of synthetic variables (PC scores)
    return(pcs)
  }
```

B.2 Find boundaries

```
> print(smw.dw)

function(tsect, wid=floor(length(tsect[,1])/4), mull=0, p.mmd=.25,
                   ident=FALSE)
{
                # silently correct unreasonable arguments
  mull=floor(mull); wid=floor(wid)
  p.mmd <- min(1, p.mmd); p.mmd <- max(0.05, p.mmd)
                # check if arguments make sense
  if (mull < 0) {
    print("Error: mullion must be positive");
    return(NULL) }
  if (mull > (wid/8)) {
    print("Error: mullion must be less than 1/4 of the half-window width");
    return(NULL) }
                # length of sequence
  len <- dim(tsect)[1]
  if ((wid < 0) || (wid > floor(len/2))) {
    print(paste(
       "Error: width must be positive and less than 1/2 of transect length:",
        len));
    return(NULL) }
                # number of variables
  nvar <- dim(tsect)[2]
                # offset from left to possible boundary;
                # for odd widths, this is the station to its left
  b.offset <- floor(wid/2)
                # half width of interval, maybe mullioned
                # remove one (common) point for even widths
  hwid <-  b.offset  - (!(wid%%2)) - mull
                # collect d's with their centre
```

```
                # both ends will be 0 (not wide enough for a window)
  d <- vector(mode="numeric", length=len)
                # move left boundary of window along transect
  for (l in 1:(len-wid)) {
                # right boundary of window
    r <- l + wid
                # option: Mahalanobis distance
                # extract halves
    win.l <- tsect[l:(l+hwid),]; win.r <- tsect[(r-hwid):r,]
                # compute column-wise means, or one mean for a vector
    if (is.null(dim(win.l))) {
        win.l.mean <- mean(win.l); win.r.mean <- mean(win.r)
    }
    else {
        win.l.mean <- apply(win.l, 2, mean)
        win.r.mean <- apply(win.r, 2, mean)
    }
    if (ident==F) {
                                        # pool halves after subtracting means
        pool <- (rbind(t(t(win.l) - win.l.mean), t(t(win.r) - win.r.mean)))
                                        #  store D^2 by boundary location
        tmp <- try(d[l + b.offset] <-
                   mahalanobis(win.l.mean, win.r.mean, cov(pool)),
                   silent=TRUE)
        if (class(tmp) == "try-error") {
            d[l + b.offset] <- 0
            print(paste("Position ", l+hwid,
                        ": Singular pooled covariance matrix; not a boundary",
                        sep=""))
        }
    }
                                        # option: Euclidean distance
    else
        d[l + b.offset] <- mahalanobis(win.l.mean, win.r.mean, diag(nvar))
}
                # sort in decreasing order of probable boundary
  ds <- sort(d, index=T, decreasing=T)
  ds <- data.frame(station=ds$ix[ds$x>0], D2=ds$x[ds$x>0])
  print(paste("Window:",wid,"stations; mullion: ",mull))
  print(paste("Distances up to", p.mmd, "of the maximum"))
  print(paste("Boundary",ifelse(!(wid%%2),"at","to the right of"),"station"))
  print(ds[ds$D2 >= (ds$D2[1]*p.mmd),])
  return(data.frame(station=seq(1:length(d)), D2=d))
}
```

Notes on this code

1. The expression `if (is.null(dim(win.l)))` tests whether the window half is a two-dimensional array (in which case the expression is FALSE) or a one-dimensional vector. Vectors do not have dimension attributes. The `mean` function can only be applied directly to a vector; for a multi-dimensional array the mean of each dimension must be found by using the `apply` function to compute the mean across the second dimension (i.e., column-wise) of the array.

2. The window half vector or array is transposed with the `t` function to make it conformable with the window half mean; then the difference between these is again transposed to row-wise, in the form expected by `cov`.

3. The `try` function surrounds code which may cause an error and which the programmer wants to handle. In this case it traps er-

rors caused by the attempt to invert a singular pooled within-half covariance matrix with `mahalanobis`. If there is such an error, as detected by the `class` function, the code sets the between-half distance to zero; see explanation at Equation 2. In addition a warning message is printed, explaining what happened.

B.3 Plot the transect

```
> print(smw.graph)

function (ds, p.mmd=.25, text="", vcol="blue", ymax=NULL) {
                # sanity check on proportion of maximum to call a boundary
  p.mmd <- min(1, p.mmd); p.mmd <- max(0.05, p.mmd)
  if (dev.cur() >1) {
                # the transect
    ylim <- (if (is.null(ymax)) NULL  else c(0, ymax))
    plot(ds, xlab="station", ylab="squared distance between halves",
         type="l", main="Boundaries?", sub=text, ylim = ylim)
    abline(h=0, lty=2)
                # main boundaries -- compare to imposed maximum if any
                #  otherwise within-transect maximum
    d.sig <- ds[ds$D2 > ifelse(is.null(ymax), max(ds$D2), ymax)*p.mmd,]
    for (i in 1:length(d.sig$D2)) {
      s <- d.sig$station[i]; d2 <- d.sig$D2[i]
      lines(c(s, s), c(0, d2), col=vcol)
      text(s,  d2, s, pos=4, col=vcol)
    }
  }
}
```

C Insight into Mahalanobis distance

This appendix is to give a feeling for the Mahalanobis distance, as opposed to Euclidean distance, and how it is affected by the variance-covariance structure of a window.

We compare two windows from a scaled and centred matrix, computed from the `trees` example dataset provided in the default `datasets` package.

> **Note:** This dataset provides measurements of the girth (diameter at breast height), height and volume of timber in 31 felled black cherry trees; see `?trees`. It is sorted in order of girth, so forms a sequence.

We consider the distance between the windows in bivariate space formed by the girth and height.

Task 16 : Load the data, scale and centre it, compute the covariance for two of variables, and then extract two sequences of four observations: (1) trees 1 . . . 4 and (2) trees 5 . . . 8 as window halves. •

```
> data(trees)
> test <- as.data.frame(scale(trees))
> var(test[, 1:2])

        Girth Height
Girth  1.0000 0.5193
Height 0.5193 1.0000

> (left <- test[1:4, 1:2])

    Girth  Height
1 -1.5769 -0.9416
2 -1.4813 -1.7264
3 -1.4175 -2.0402
4 -0.8758 -0.6278

> (right <- test[5:8, 1:2])

    Girth  Height
5 -0.8121  0.7847
6 -0.7802  1.0986
7 -0.7165 -1.5694
8 -0.7165 -0.1569

> l.mean <- apply(left, 2, mean)
> r.mean <- apply(right, 2, mean)
> l.mean - r.mean

  Girth  Height
-0.5816 -1.3732
```

Over the whole sequence, the variables both have unit variance (because of scaling) and moderate positive correlation (r = 0.52). There seems to be a good contrast between the windows because of the varying heights.

The Euclidean distance is simply the RMS of the difference; this can also be computed by the `mahalanobis` function, using an identity matrix **I** in place of the variance-covariance matrix **W**:

```
> sqrt(sum((l.mean - r.mean)^2))

[1] 1.491

> sqrt(mahalanobis(l.mean, r.mean, diag(2)))

[1] 1.491
```

Euclidean distance does not weight the variables by their discriminating power. In this case we see that the height shows more difference between the two sequences than does the girth. For this we use the pooled within-half variance-covariance matrix **W**, which takes into account the variance of each variable (lower is more discriminating) and their covariance.

TASK 17 : Compute the pooled within-half variance-covariance matrix **W**. •

```
> # note the transposition to compute column means
> (left.resid <- t(left) - l.mean)

             1       2        3      4
Girth  -0.2390 -0.1434 -0.07967 0.4621
Height  0.3924 -0.3924 -0.70624 0.7062

> (right.resid <- t(right) - r.mean)

              5       6        7        8
Girth  -0.05577 -0.0239  0.03983  0.03983
Height  0.74547  1.0594 -1.60865 -0.19618

> (resid <- (rbind(t(left.resid), t(right.resid))))

     Girth  Height
1 -0.23900  0.3924
2 -0.14340 -0.3924
3 -0.07967 -0.7062
4  0.46206  0.7062
5 -0.05577  0.7455
6 -0.02390  1.0594
7  0.03983 -1.6086
8  0.03983 -0.1962

> apply(resid, 2, sd)

 Girth Height
0.2085 0.8952

> (W <- cov(resid))

         Girth  Height
Girth  0.04348 0.02947
Height 0.02947 0.80137

> var(test[, 1:2])

        Girth Height
Girth  1.0000 0.5193
Height 0.5193 1.0000
```

So for this particular window, after subtracting the respective mean from each half, the first scaled variable has much less residual variance than the second; that is, observations of the first variable are closer, in each half, to that half's mean than is the case for the second variable. The

whole-sequence residual variance is of course 1 for each variable; this is the same as the variance because each variable has zero mean from the scaling.

This shows that the variance-covariance matrix of a window can vary greatly from that of the whole sequence In this case variances are 0.043 and 0.801, respectively, for the two variables, rather than 1 for the whole sequence) and the covariance is 0.029 (instead of 0.519 for the whole sequence). The two residuals are weakly correlated (i.e., girth is not a good predictor of height, and vice-versa), so this will hardly affect the distance.

We can now examine the contribution of each variable to the distance.

TASK 18: Compute the distance step-by-step according to the definition of Mahalanobis distance, then with the direct call to `mahalanobis`. •

```
> (diff <- l.mean - r.mean)

  Girth  Height
-0.5816 -1.3732

> (W.inv <- solve(W, diag(2)))

          [,1]    [,2]
Girth  23.5855 -0.8674
Height -0.8674  1.2798

> W.inv[1, 1]/W.inv[2, 2]

Girth
18.43

> (tmp <- drop(W.inv %*% diff))

  Girth  Height
-12.525  -1.253

> # note: drop removes redundant dimensions from arrays
> (d.squared <- drop(diff %*% tmp))

[1] 9.005

> sqrt(d.squared)

[1] 3.001

> sqrt(mahalanobis(l.mean, r.mean, W))

[1] 3.001
```

Notice that the first scaled variable has a much higher weight (over 18 times) than the second. This is because a variable with low variance in the residuals has higher discriminating power in the mean. The residuals of the two variables were slightly positively correlated; this lowers the distance slightly, because of the negative entries in the inverse. The final result is about twice the Euclidean distance, mainly because of the large discriminating power (low residual variance) of the first variable.

Task 19 : Extract another two sequences of four observations: (1) trees 22...25 and (2) treess 26...29 as window halves, and compute their Mahalanobis distance. •

These windows turn out to have higher variances (because the trees are larger in this section of the dataframe) and a slight negative correlation between the residuals. The negative correlation increases the weight of both variables, because it becomes positive in the inverse. Both variables discriminate well, but the first has about three times the weight as the second.

```
> (left <- test[22:25, 1:2])

    Girth  Height
22 0.3032  0.6278
23 0.3988 -0.3139
24 0.8768 -0.6278
25 0.9724  0.1569

> (right <- test[26:29, 1:2])

   Girth Height
26 1.291 0.7847
27 1.355 0.9416
28 1.482 0.6278
29 1.514 0.6278

> l.mean <- apply(left, 2, mean)
> r.mean <- apply(right, 2, mean)
> (diff <- l.mean - r.mean)

  Girth  Height
-0.7728 -0.7847

> sqrt(sum((l.mean - r.mean)^2))

[1] 1.101

> left.resid <- t(left) - l.mean
> right.resid <- t(right) - r.mean
> resid <- (rbind(t(left.resid), t(right.resid)))
> apply(resid, 2, sd)

 Girth Height
0.2303 0.3728

> (W <- cov(resid))

          Girth  Height
Girth   0.05306 -0.0384
Height -0.03840  0.1390

> (W.inv <- solve(W, diag(2)))

         [,1]  [,2]
Girth  23.559 6.509
Height  6.509 8.993

> W.inv[1, 1]/W.inv[2, 2]

Girth
 2.62

> tmp <- drop(W.inv %*% diff)
> (d.squared <- drop(diff %*% tmp))
```

```
[1] 27.5

> sqrt(d.squared)

[1] 5.244

> sqrt(mahalanobis(l.mean, r.mean, W))

[1] 5.244
```

In this case the Euclidean distance, 1.101, is increased dramatically to the Mahalanobis distance, 5.244.

Index of R Concepts