
Technical Note:

Optimal partitioning of soil transects with R

D G Rossiter

January 6, 2009

Contents

1 Example Data Set	1
2 Split moving-window	2
2.1 Options for the SMW method	4
2.2 Analyzing the transect	9
2.3 Visualising the boundaries	14
3 Maximum Level Variance	16
References	18
A Sample data frame	19
B Split moving-window functions	20
C Plotting the transect	23
D Weighting variables for Mahalanobis distance	24

Copyright© 2004, 2009 D G Rossiter. All rights reserved. Reproduction and dissemination of the work as a whole (not parts) freely permitted if this original copyright notice is included. Sale or placement on a web site where payment must be made to access this document is strictly prohibited.

To adapt or translate please contact the author (<http://www.itc.nl/personal/rossiter>).

This note describes procedures to indentify soil boundaries along a transect where soil samples have been taken at regular intervals. It is based on the work of Webster [7, 4, 5, 6] and used as an example in the text of Davis [1, pp. 234-243].

Algorithms are programmed with the R environment for statistical computing and visualisation [2, 3]; the language is a dialect of S.

1 Example Data Set

This is a single 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 LBA¹ project. Both field and laboratory measurements were made; for illustrative purposes we use only the sand and clay content in g kg^{-1} of three layers. These are organised as an R *data frame* with the rows being the stations and the columns the variables. The sample R object is shown in Appendix A; here is its structure:

R code:

```
R> 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 ...
```

Figure 1 shows both the original six variables and their first two principal components plotted along the transect.² It is evident that there is autocorrelation, i.e. nearby stations are likely to be similar, and that the sequence can be broken up into several more homogeneous sections. A very clear difference is that the beginning of the sequence has fairly equal sand and clay, and the end has much more sand than clay. The ratios between the horizons seems to differ in smaller bands as well.

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

² See Appendix C for the code to generate these graphs.

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

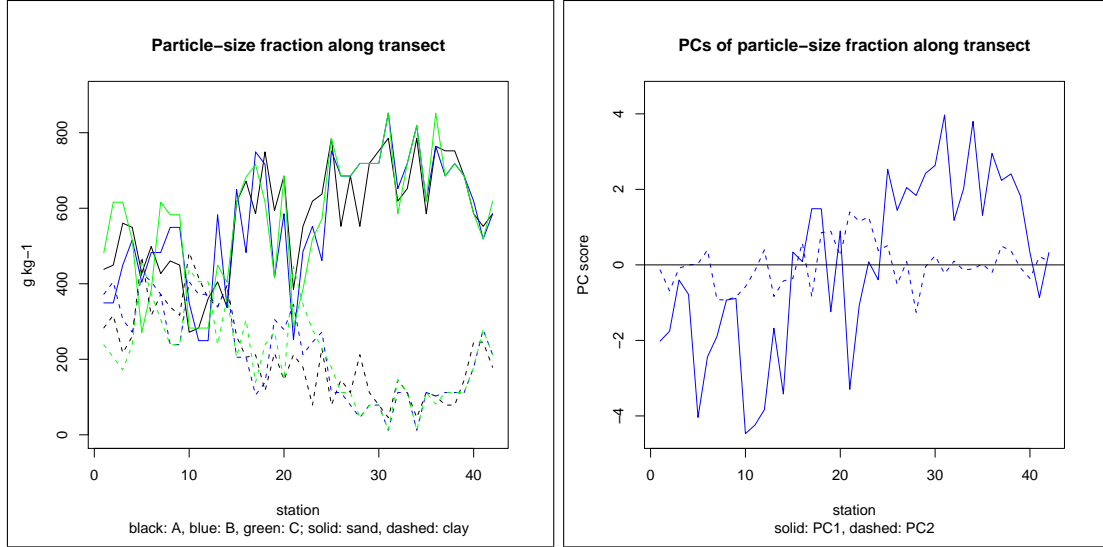


Figure 1: Variables and PCs along a transect

tion is, where to put boundaries; in particular, which algorithm will find the ones that we believe correspond to real soil 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:

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

where $\bar{\mathbf{x}}_s$ is the average vector of the *left* half, $\bar{\mathbf{x}}_d$ is the average vector of the *right* half, and \mathbf{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 ways³:

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).

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

If \mathbf{W} is replaced by the identity matrix \mathbf{I} , this is equivalent to the squared Euclidean distance:

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

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 \mathbf{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.

I have written three R functions to implement the SMW method; they are contained in file `smw.R` and loaded with the `source` method:

R code:

```
R> source("smw.R")
R> ls(pattern="smw.*")
[1] "smw.dw"      "smw.graph"  "smw.pc"
```

These must be called in the following order:

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

³ See numeric example in Appendix D

2.1 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”, that is, leave out some stations near the middle of the window;
6. What distance in feature space (as a proportion of the maximum in the whole transect) probably signals a boundary, and should be reported?

Standardized principal components Function `smw.pc` computes the standardized principal components from the correlation matrix⁴ and replace 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.

⁴ rather than the variance-covariance matrix

R code:

```
R> pc <- smw.pc(transect)
[1] "PCA: selected components explain 0.923 of the variance"
[1] "Loadings for the first 2 components:"
      Comp.1      Comp.2
sand.A  0.406091  0.4737026
clay.A  -0.396790 -0.6276406
sand.B  0.413279 -0.2525418
clay.B  -0.414862  0.0293524
sand.C  0.409541 -0.3678222
clay.C  -0.408673  0.4263109
R> str(pc)
`data.frame`: 42 obs. of  2 variables:
 $ PC1: num  -2.043 -1.779 -0.406 -0.792 -4.087 ...
 $ PC2: num  -0.13489 -0.69835 -0.08701 -0.00681  0.03292 ...
```

Here we see that the two PCs explain most (92%) of the variance. 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.

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.

Working with 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` method; the result must be converted back into a data frame:

R code:

```
R> tr.scale<-data.frame(scale(transect))
R> str(tr.scale)
`data.frame`: 42 obs. of 6 variables:
 $ sand.A: num -0.960 -0.882 -0.104 -0.181 -1.031 ...
 $ 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:

R code:

```
R> round(mean(tr.scale),4)
sand.A clay.A sand.B clay.B sand.C clay.C
      0      0      0      0      0      0
R> var(tr.scale)
      sand.A      clay.A      sand.B      clay.B      sand.C      clay.C
sand.A 1.000000 -0.901607 0.831688 -0.817524 0.823691 -0.766978
clay.A -0.901607 1.000000 -0.759604 0.841740 -0.754100 0.769720
sand.B 0.831688 -0.759604 1.000000 -0.906405 0.866090 -0.860030
clay.B -0.817524 0.841740 -0.906405 1.000000 -0.822096 0.858089
sand.C 0.823691 -0.754100 0.866090 -0.822096 1.000000 -0.912884
clay.C -0.766978 0.769720 -0.860030 0.858089 -0.912884 1.000000
```

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 [5] recommends 2/3 of the expected distance between boundaries, i.e. 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”) method⁵ displays a graph of autocorrelation by lag (Figure 2), and the results can also be printed on the console:

⁵ supplied by the standard `stats` package, which is installed and loaded by default in R version 2

R code:

```
R> acf(pc[,1])
R> print(acf(pc[,1]))
Autocorrelations of series 'pc[, 1]', by lag
  0    1    2    3    4    5    6    7    8    9   10
1.000 0.687 0.637 0.531 0.422 0.440 0.310 0.318 0.300 0.331 0.182
 11   12   13   14   15   16
0.171 0.027 -0.039 -0.055 -0.200 -0.131
```

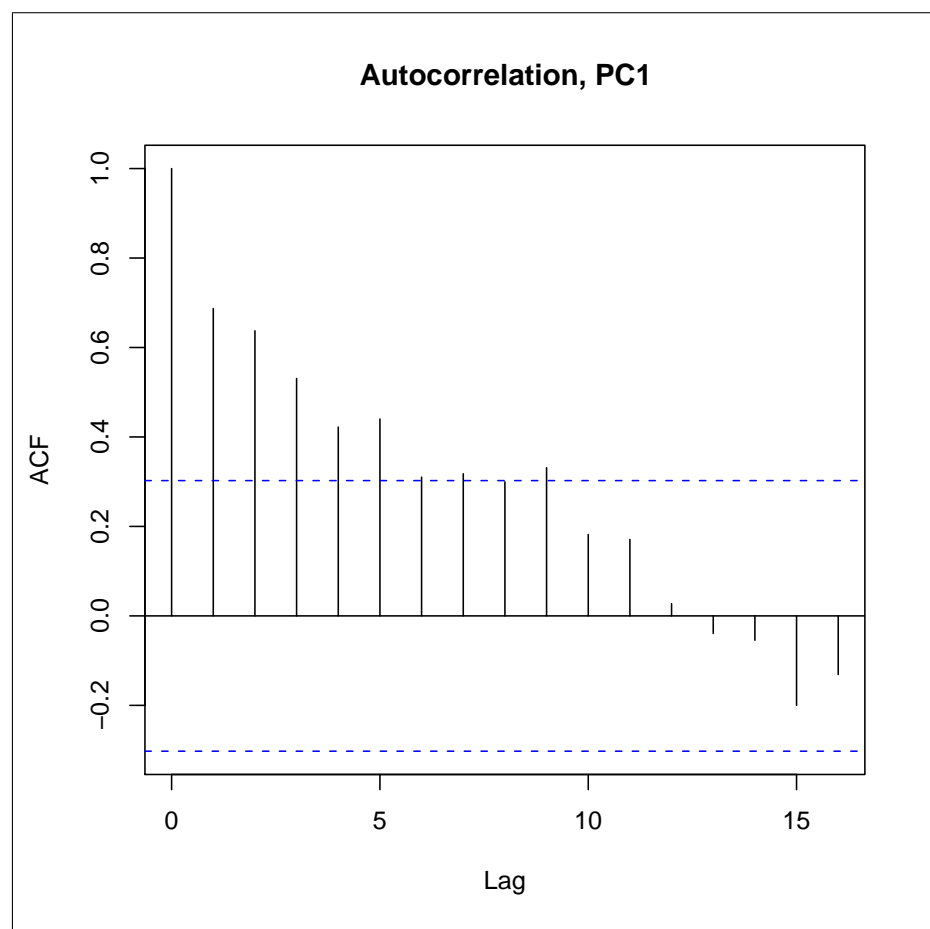


Figure 2: Autocorrelation function, PC 1

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.

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 ... 9, the middle of which is station 5. The left half of the split window includes stations 1 ... 4 and the right half 6 ... 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 ... 8. The left half of the split window includes stations 1 ... 4 and the right half 5 ... 8; the potential boundary is between stations 4 and 5. This is reported as station 4 but is actually located at position 4.5.

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 *mullion*. In this implementation of SMW user-specified the 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.

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=.1`. This does not affect the calculation.

2.2 Analyzing the 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.

2.2.1 Using PCs

The SMW method is usually 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.

R code:

```
R> d <- smw.dw(pc, wid=8)
[1] "Window: 8 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    MahD
1      15 25.48893
2      14 24.92310
3       7 16.21807
4      25 15.56409
5       6  7.97835
6      24  7.78813
7      16  7.52836
8      26  6.77424
9      38  6.58369
```

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.

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.

R code:

```
R> d <- smw.dw(pc, wid=7)
[1] "Window: 7 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary to the right of station"
  station    MahD
1         6 42.4561
2        14 29.4894
3        24 13.8729
```

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:

R code:

```
R> d <- smw.dw(pc, wid=10)
[1] "Window: 10 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    MahD
1        15 28.02404
2        14 14.35580
3        24 10.19626
4        25  8.82321
5        16  8.26165
6        18  7.80522
7        26  7.40862
```

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

R code:

```
R> d <- smw.dw(pc, wid=10, mull=1)
[1] "Window: 10 stations; mullion: 1"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    MahD
1      16 31.31891
2      15 22.12585
3      26 17.77400
4      17 17.67749
5      14 15.68168
6      13 11.42231
7      24  9.50595
8      25  8.63859
```

Using a narrow window changes the picture radically:

R code:

```
R> d <- smw.dw(pc, wid=6)
[1] "Window: 6 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    MahD
1         6 476.436
R> d <- smw.dw(pc, wid=6, p.mmd=.05)
[1] "Window: 6 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.05 of the maximum"
[1] "Boundary at station"
  station    MahD
1         6 476.4356
2        39  74.0673
3        25  41.6222
4        14  38.7734
5        15  36.5043
6         7  34.6747
7        24  26.4583
```

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.

Back to the 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 be associated with changes in PC2 (textural contrast) rather than PC1 (overall texture).

R code:

```
R> d <- smw.dw(as.data.frame(pc$PC1), wid=8)
[1] "Window: 8 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    MahD
1      14 20.69928
2      15 10.97163
3      25  9.40301
4      24  5.23946
```

And indeed, looking for boundaries with PC2 only:

R code:

```
R> d <- smw.dw(as.data.frame(pc$PC2), wid=8)
[1] "Window: 8 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary at station"
  station    MahD
1       25 8.84110
2        7 7.06594
3        6 6.57049
4       18 5.26879
5       17 4.81807
6       24 4.57482
7       26 3.97601
8       23 3.25627
9       19 2.32907
```

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 Mahalanobis distance with principal components The Mahalanobis 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 proportional 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.

2.2.2 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 result. Here we compare it to the same window size but using the first two PCs:⁶

R code:

```
R> d<-smw.dw(transect, wid=9)
[1] "Window: 9 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary to the right of station"
  station      D2
1      6 298.9197
2     24 208.5018
3     25 160.1176
4      9 113.3527
5     35 108.0591
6     14 105.0294
7     28 101.5057
8     17  77.5602
R> d<-smw.dw(pc, wid=9)
[1] "Window: 9 stations; mullion: 0"
[1] "Mahalanobis distances up to 0.25 of the maximum"
[1] "Boundary to the right of station"
  station      D2
1     14 22.95347
2     15  9.81062
3     24  8.31793
4     17  7.57148
5     25  7.16004
```

Both methods suggest a boundary near stations 24 and 25, but us-

⁶ Using all six PCs gives exactly the same result as original variables, since the information content is the same.

ing 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.

2.3 Visualising the boundaries

The results of `smw.dw` can be visualised with `smw.graph`; a graphics device must be open first. A graphics window on the screen is opened with the `windows()` command⁷, but the result can also be saved in a file:

R code:

```
R> pdf("SMW_PC2_Win8_Mull10.pdf")
R> smw.graph(d)
R> dev.off()
```

The results are shown in Figure 3.

We can compare various approaches visually by writing a small script and putting the graphs in one frame; this repeats the examples of §2.2 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`.

⁷on a Windows system; on Mac OS X use `quartz()` and on Linux `x11()`

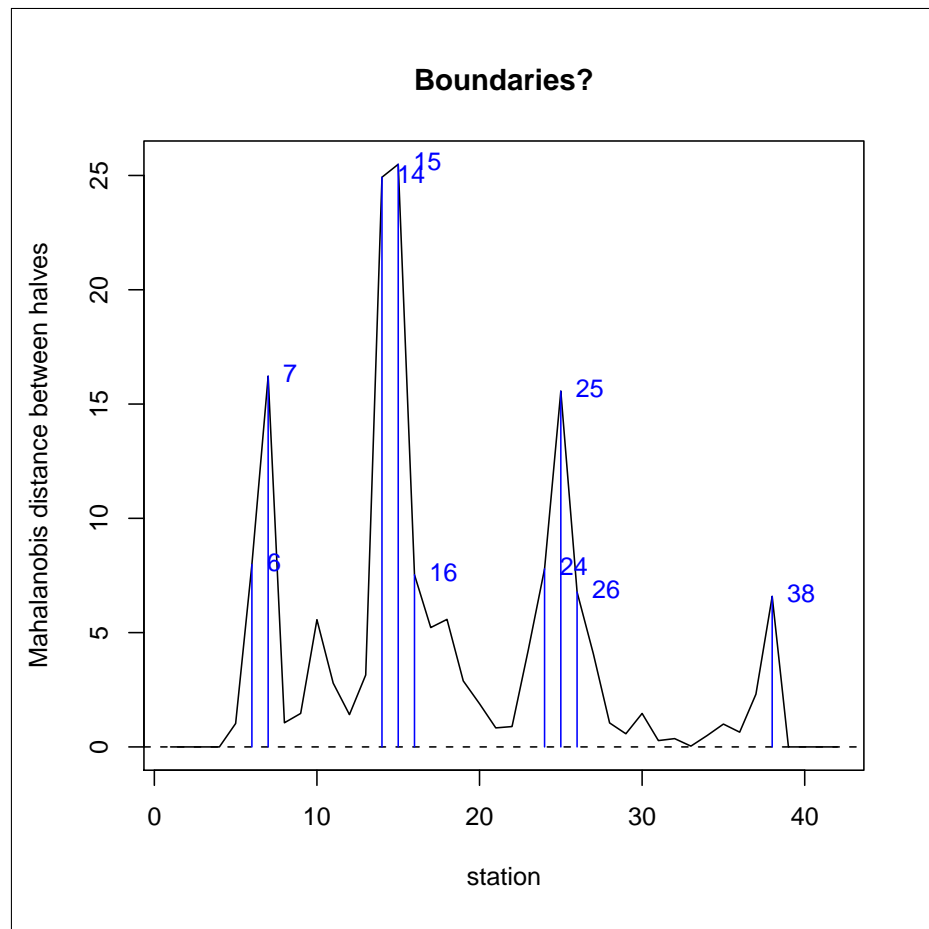


Figure 3: Boundaries identified by the SMW algorithm, 2 PCs, window width = 8, no mullion

R code:

```
pdf("SMW_Examples.pdf", height=12, width=12)
par(mfrow=c(3,3))
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")
# original vars
d <- smw.dw(transect, wid=9)
```

The results are shown in Figure 4.

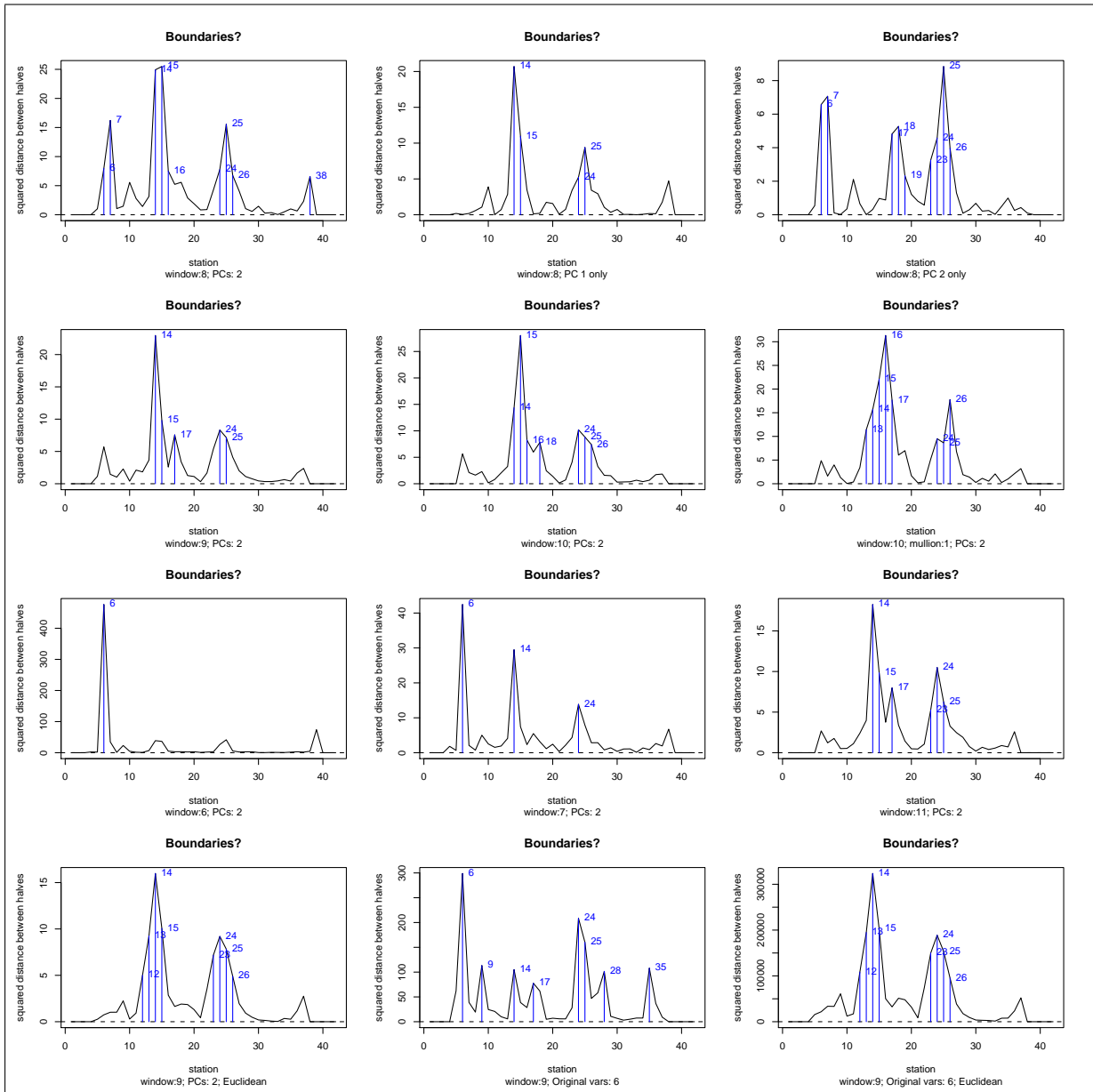


Figure 4: Boundaries identified by the SMW algorithm with various parameters

3 Maximum Level Variance

The Maximum Level Variance (MLV) approach 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. It 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.

References

- [1] John C. Davis. *Statistics and data analysis in geology*. John Wiley & Sons, New York, 3rd edition, 2002.
- [2] R Ihaka and R Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
- [3] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL <http://www.R-project.org>. ISBN 3-900051-07-0.
- [4] R. Webster. Automatic soil-boundary location from transect data. *Mathematical Geology*, 5:27–37, 1973.
- [5] R. Webster. Optimally partitioning soil transects. *Journal of Soil Science*, 29:388–402, 1978.
- [6] R. Webster. DIVIDE, a FORTRAN IV program for segmenting multivariate one-dimensional spatial series. *Computers & Geosciences*, 6(1):61–68, 1980.
- [7] R Webster and I F T Wong. A numerical procedure for testing soil boundaries interpreted from air photographs. *Photogrammetria*, 24:59–72, 1969.

A Sample data frame

	sand.A	clay.A	sand.B	clay.B	sand.C	clay.C
1	438.24	283.10	349.36	371.99	482.69	238.66
2	449.36	316.43	349.36	405.32	616.02	205.33
3	560.46	216.44	449.36	305.32	616.02	171.99
4	549.35	260.88	516.02	271.99	516.02	238.66
5	428.02	471.98	404.02	429.32	270.70	462.65
6	498.69	315.54	482.69	405.32	382.69	371.99
7	427.13	371.99	482.69	371.99	616.02	305.32
8	460.47	338.65	549.35	238.66	582.68	238.66
9	449.36	316.43	549.35	238.66	582.68	238.66
10	271.58	483.09	349.36	405.32	282.70	438.65
11	282.70	416.43	249.36	371.99	282.70	405.32
12	360.47	360.87	249.36	371.99	282.70	405.32
13	404.91	338.65	582.68	338.65	449.36	238.66
14	337.36	395.98	337.36	395.98	404.02	362.65
15	616.02	260.88	649.35	205.33	616.02	205.33
16	671.57	205.33	482.69	205.33	682.68	305.32
17	585.35	211.99	749.34	105.33	716.01	138.66
18	749.34	116.44	716.01	138.66	616.02	238.66
19	593.79	216.44	416.02	305.32	416.02	271.99
20	685.35	145.33	585.35	278.66	685.35	145.33
21	385.36	211.99	252.03	345.32	285.36	445.32
22	552.02	178.66	485.35	211.99	385.36	345.32
23	618.68	78.66	552.02	245.32	518.69	278.66
24	637.35	229.32	461.35	271.99	570.68	229.32
25	785.34	78.66	752.01	112.00	785.34	178.66
26	552.02	145.33	685.35	112.00	685.35	112.00
27	685.35	112.00	685.35	78.66	685.35	112.00
28	552.02	211.99	718.68	45.33	718.68	45.33
29	718.68	112.00	718.68	78.66	718.68	78.66
30	752.01	78.66	718.68	78.66	718.68	78.66
31	785.34	45.33	852.01	12.00	852.01	12.00
32	618.68	145.33	652.01	112.00	585.35	145.33
33	652.01	112.00	718.68	112.00	718.68	112.00
34	785.34	45.33	818.67	12.00	818.67	12.00
35	585.35	112.00	618.68	112.00	618.68	112.00
36	764.01	102.66	764.01	102.66	852.01	81.33
37	752.01	78.66	685.35	112.00	685.35	112.00
38	752.01	78.66	718.68	112.00	718.68	112.00
39	685.35	145.33	685.35	112.00	685.35	112.00
40	585.35	245.32	618.68	178.66	585.35	178.66
41	552.02	245.32	518.69	278.66	518.69	278.66
42	585.35	178.66	585.35	211.99	618.68	211.99

B Split moving-window functions

R code:

```
### method smw.pc()
##   extracts standardized PC's from a multivariate transect
## arguments
##   transect: data frame; rows are equally-spaced,
##             columns are variables
##   n.pc     : number of PCs to extract, default 2
## returns
##   data frame with columns as PC scores named "PC.1" etc.
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)
}
```

R code:

```
### method smw.dw()
##      find Mahalanobis distances for one transect and window size
## arguments
##      tsect: data frame; rows are equally-spaced observations,
##            columns are variables
##      wid: points in moving window
##      mull: points to omit at boundary from each half-window ("mullion")
##      p.mmd: proportion of maximum Mahalanobis distance for a
##            boundary to show in the summary table
##      ident: calculate Euclidean (T) or Mahalanobis (F, default) distances?
## returns
##      data frame with station and D2 (distance) in station order;
##      note l. and r. ends will be 0 (not in any window)
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)
}
### continued on next page
```

R code:

```
### continued from previous page
      # 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,]
if (ident==F) {
  # pool halves after subtracting means
  pool <- (rbind(t(t(win.l) - mean(win.l)), t(t(win.r) - mean(win.r))));
  # store D^2 by boundary location
  tmp <- try(mahalanobis(mean(win.l), mean(win.r), cov(pool)), silent=T);
  # singular matrix means zero distance
  d[l + b.offset] <- ifelse (class(tmp) == "try-error", 0, tmp)
}
  # option: Euclidean distance
else
  {
    d[l + b.offset] <- mahalanobis(mean(win.l), mean(win.r), 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("Mahalanobis 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))
}
```

R code:

```
## returns: nothing
smw.graph <- function (ds, p.mmd=.25, text="") {
  p.mmd <- min(1, p.mmd); p.mmd <- max(0.05, p.mmd)
  if (dev.cur() >1) {
    # the transect
    plot(ds, xlab="station", ylab="squared distance between halves",
         type="l", main="Boundaries?", sub=text)
    abline(h=0, lty=2)
    # main boundaries
    d.sig <- ds[ds$D2 > max(ds$D2)*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="blue")
      text(s, d2, s, pos=4, col="blue")
    }
  }
}
```

C Plotting the transect

R code:

```
# plot 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")
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)
#
# compute and plot PCs
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)
```

D Weighting variables for 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 specific variance-covariance structure of a window.

Consider the following two windows from a (scaled and centred) matrix. The distance between the windows is to be computed in bivariate space.⁸ We load the data, compute the covariance for two of variables, and then extract two sequences of four observations as window halves:

R code:

```
R> data(trees)
R> test <- as.data.frame(scale(trees))
R> var(test[,1:2])
      Girth Height
Girth 1.00000 0.51928
Height 0.51928 1.00000
R> left <- test[1:4, 1:2]; right <- test[5:8, 1:2]
R> left;right
      Girth Height
1 -1.576854 -0.941647
2 -1.481256 -1.726353
3 -1.417524 -2.040236
4 -0.875802 -0.627765
      Girth Height
5 -0.812070  0.784706
6 -0.780204  1.098588
7 -0.716472 -1.569412
8 -0.716472 -0.156941
R> mean(left)-mean(right)
      Girth Height
-0.581555 -1.373236
```

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, with height having a greater difference.

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

⁸ The data is from the `trees` dataset which is part of the `stats` library loaded by default

R code:

```
R> sqrt(sum((mean(left)-mean(right))^2))
[1] 1.49130
R> sqrt(mahalanobis(mean(left),mean(right),diag(2)))
[1] 1.49130
```

Euclidean distance does not weight the variables by their discriminating power; 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. We compute this matrix as follows:

R code:

```
R> left.resid <- t(left) - mean(left)
R> # note the transposition to compute column means
R> right.resid <- t(right) - mean(right)
R> (resid <- (rbind(t(left.resid), t(right.resid))))
      Girth  Height
1 -0.2389952  0.392353
2 -0.1433971 -0.392353
3 -0.0796651 -0.706235
4  0.4620573  0.706235
5 -0.0557655  0.745471
6 -0.0238995  1.059353
7  0.0398325 -1.608647
8  0.0398325 -0.196177
R> sd(resid)
      Girth  Height
0.208525  0.895194
R> (W <- cov(resid))
      Girth  Height
Girth  0.0434827  0.0294707
Height  0.0294707  0.8013723
R> var(test[,1:2])
      Girth  Height
Girth  1.00000  0.51928
Height  0.51928  1.00000
```

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 vari-

ance 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 0.04 and 0.80, respectively, for the two variables (instead of 1 for the whole sequence) and the covariance is 0.03 (instead of 0.52 for the whole sequence).

The two residuals are very weakly correlated, so this will hardly affect the distance. We can now examine the contribution of each variable to the distance. First, we compute the distance step-by-step, then with the direct call to `mahalanobis`:

R code:

```
R> (diff <- mean(left) - mean(right))
      Girth      Height
-0.581555 -1.373236
R> (W.inv <- solve(W,diag(2)))
      [,1]      [,2]
Girth 23.585492 -0.867364
Height -0.867364  1.279757
R> W.inv[1,1]/W.inv[2,2]
[1] 18.4297
R> (tmp <- drop(W.inv %*% diff))
      Girth      Height
-12.52516  -1.25299
R> # note: drop removes redundant dimensions from arrays
R> (d.squared <- drop(diff%*%tmp))
[1] 9.00472
R> sqrt(d.squared)
[1] 3.00079
R> sqrt(mahalanobis(mean(left),mean(right),W))
[1] 3.00079
```

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 and 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.

Here's another window from the sequence, this time with more even variances and a slight negative correlation between the residuals. In this case the Euclidean distance, 1.10, is increased dra-

matically to the Mahalanobis distance, 5.24. 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.

R code:

```
R> left <- test[22:25, 1:2]; right <- test[26:29, 1:2]
R> left;right
      Girth  Height
22 0.303241 0.627765
23 0.398839 -0.313882
24 0.876830 -0.627765
25 0.972428 0.156941
      Girth  Height
26 1.29109 0.784706
27 1.35482 0.941647
28 1.48228 0.627765
29 1.51415 0.627765
R> mean(left)-mean(right)
      Girth  Height
-0.772751 -0.784706
R> sqrt(sum((mean(left)-mean(right))^2))
[1] 1.10132
R> left.resid <- t(left) - mean(left); right.resid <- t(right) - mean(right)
R> (resid <- (rbind(t(left.resid), t(right.resid))))
      Girth  Height
22 -0.3345933 0.6670001
23 -0.2389952 -0.2746471
24 0.2389952 -0.5885295
25 0.3345933 0.1961765
26 -0.1194976 0.0392353
27 -0.0557655 0.1961765
28 0.0716986 -0.1177059
29 0.1035646 -0.1177059
R> sd(resid)
      Girth  Height
0.230341 0.372809
R> (W <- cov(resid))
      Girth  Height
Girth 0.0530569 -0.0384012
Height -0.0384012 0.1389866
R> (diff <- mean(left) - mean(right))
      Girth  Height
-0.772751 -0.784706
R> (W.inv <- solve(W,diag(2)))
      [,1] [,2]
Girth 23.55886 6.50918
Height 6.50918 8.99339
R> (tmp <- drop(W.inv %*% diff))
      Girth  Height
-23.3129 -12.0871
R> (d.squared <- drop(diff%*%tmp))
[1] 27.5000
R> sqrt(d.squared)
[1] 5.24404
```