An introduction to the **prospectr** package

Antoine Stevens[1] and Leonardo Ramirez–Lopez[2]

[1]Georges Lemaître Centre for Earth and Climate Research, Earth and Life Institute, UCLouvain, Place Pasteur 3, 1348 Louvain–La–Neuve, Belgium. ✉ antoine.stevens@uclouvain.be
[2]Switzerland. ✉ leonardo.ramirez@wsl.ch

August 12, 2013

Contents

1 Introduction

Visible and Near Infrared diffuse reflectance (vis–NIR) spectroscopy is a high–troughput, non–destructive and cheap sensing method that has a range of applications in agricultural, medical, food and environmental science. A number of `R` packages of interest for the spectroscopist is already available for processing and analysis of spectroscopic data (Table 1). The CRAN task views Multivariate Statistics, Machine Learning, Chemometrics and Computational Physics. The interested reader can also have a look at the special issue "Spectroscopy and Chemometrics in R" of the Journal of Statistical Software [9].

Table 1: Non–exhaustive list of R package useful for vis–NIR spectroscopic analysis

Package Name	Description
chemometrics	functions and scripts for chemometrics
ChemometricsWithR	functions and scripts for chemometrics
ChemoSpec	misc functions for exploratory analysis in Spectroscopy
hyperSpec	processing and visualisation of spectra
cluster	cluster analysis and visualisation
mvoutlier	outlier detection in the multivariate space
pls	partial least square regression
signal	signal filtering
soil.spec	some functions related to soil spectroscopy
caret	training classification and regression models

The prospectr package gathers algorithms commonly–used in spectroscopy for pre–treating spectra and select calibration samples. Some of the algorithms are already available in other package, like the Savitzky–Golay algorithm [12] but our functions works indifferently on `vector`, `data.frame` or `matrix` input.

2 Signal Processing

The aim of signal pre–treatment is to improve data quality before modeling and remove physical information from the spectra. Applying a pre–treatment can increase the repeatability/reproducibility of the method, model robustness and accuracy, although there are no guarantees this will actually work .... The pre–processing functions that are currently available in the package are listed in Table 2.

We show below how they can be used, using the NIRsoil dataset included in the package [6]. Observations should be arranged row–wise.

```r
library(prospectr)
data(NIRsoil)
# NIRsoil is a data.frame with 825 obs and 5 variables: Nt (Total
# Nitrogen), Ciso (Carbon), CEC (Cation Exchange Capacity), train (vector
# of 0,1 indicating training (1) and validation (0) samples), spc
# (spectral matrix)
str(NIRsoil)
```

Table 2: List of pre–processing functions

Function Name	Description
`movav`	simple moving (or running) average filter
`savitzkyGolay`	Savitzky–Golay smoothing and derivative
`gapDer`	gap–segment derivative
`continuumRemoval`	compute continuum–removed values
`detrend`	detrend normalization
`standardNormalVariate`	Standard Normal Variate (SNV) transformation
`binning`	average a signal in column bins
`resample`	resample a signal to new band positions
`resample2`	resample a signal using FWHM values
`blockScale`	block scaling
`blockNorm`	sum of squares block weighting

```
## 'data.frame': 825 obs. of  5 variables:
##  $ Nt   : num  0.3 0.69 0.71 0.85 NA ...
##  $ Ciso : num  0.22 NA NA NA 0.9 NA NA 0.6 NA 1.28 ...
##  $ CEC  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ train: num  1 1 1 1 1 1 1 1 1 1 ...
##  $ spc  : num [1:825, 1:700] 0.339 0.308 0.328 0.364 0.237 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr  "1" "2" "3" "4" ...
##   .. ..$ : chr  "1100" "1102" "1104" "1106" ...
```

2.1 Noise removal

Noise represents random fluctuations around the signal that can originate from the instrument or environmental laboratory conditions. The simplest solution to remove noise is to perform n repetition of the measurements, and the average individual spectra. The noise will decrease with a factor $\sqrt{n}$. When this is not possible, or if residual noise is still present in the data, the noise can be removed mathematically.

2.1.1 Moving average or runnnig mean

A moving average filter is a column–wise operation which average contiguous wavelengths within a given window size.

```r
noisy <- NIRsoil$spc + rnorm(length(NIRsoil$spc), 0, 0.001)  # adding some noise
# Plot the first spectrum
plot(as.numeric(colnames(NIRsoil$spc)), noisy[1, ], type = "l", xlab = "Wavelength",
    ylab = "Absorbance")
X <- movav(noisy, w = 11)  # window size of 11 bands
# Note that the 5 first and last bands are lost in the process
```

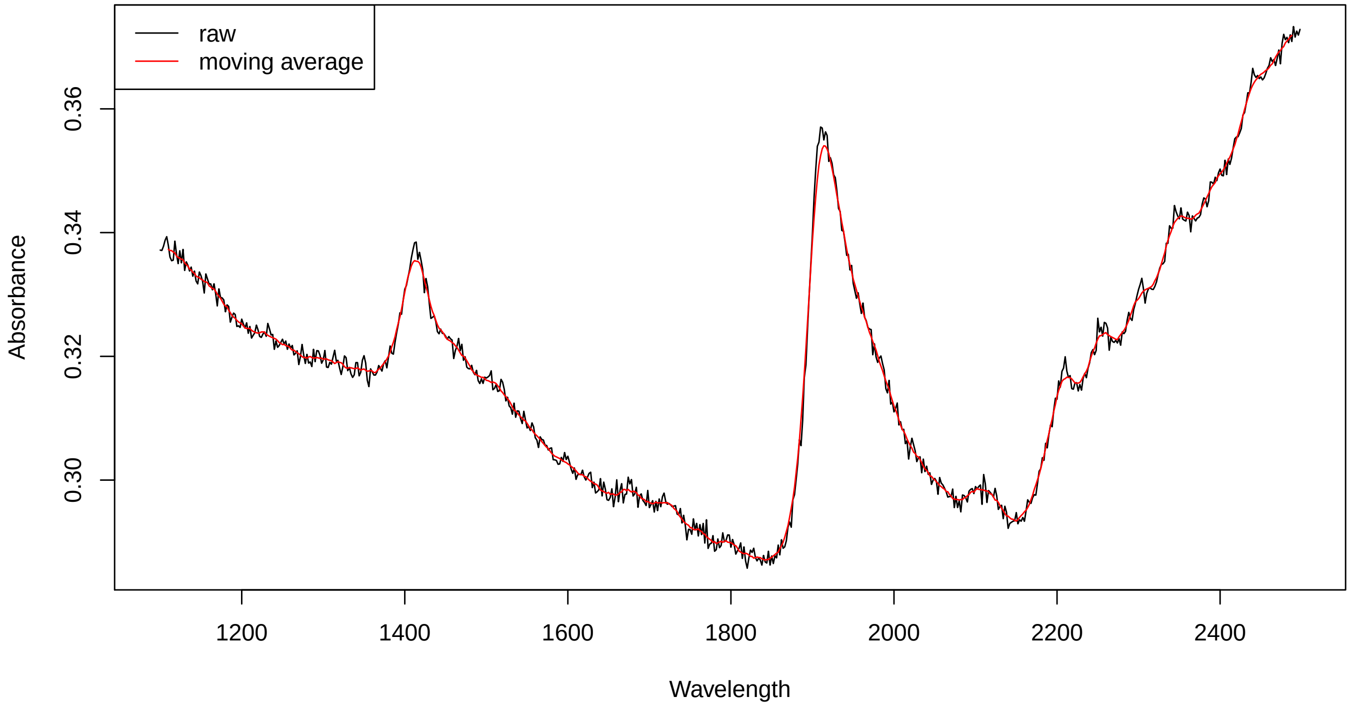

Figure 1: Effect of a moving average with window size of 10 bands on a raw spectrum

```
lines(as.numeric(colnames(X)), X[1, ], col = "red")
legend("topleft", legend = c("raw", "moving average"), lty = c(1, 1), col = 1:2)
```

2.1.2 Binning

```
# After averaging, the spectrum can be further resampled (binning) We keep
# here one 1 out every 10 data points
X.bin <- binning(X, bin.size = 10)
# We reduce the spectral matrix to 50 (equally-spaced) data points
X.bin2 <- binning(X, bins = 50)
# Plot the first spectrum
plot(as.numeric(colnames(X)), X[1, ], type = "l", xlab = "Wavelength", ylab = "Absorbance")
# new data points
points(as.numeric(colnames(X.bin)), X.bin[1, ], pch = 2)
points(as.numeric(colnames(X.bin2)), X.bin2[1, ], pch = 1, col = 2)
legend("topleft", legend = c("bin.size = 10", "bins = 50"), pch = 2:1, col = 2:1)
```

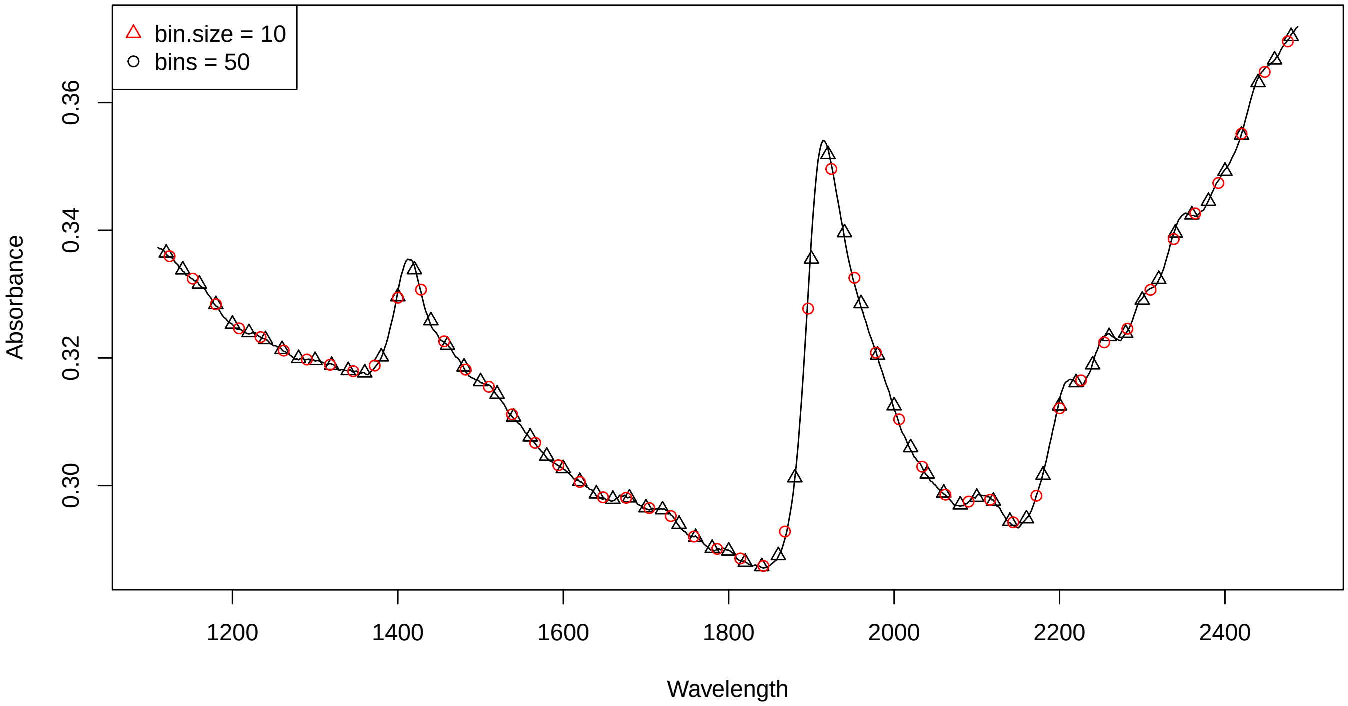

Figure 2: Average in bins

2.1.3 Savitzky-Golay filtering

Savitzky-Golay filtering [12] is a very common preprocessing technique. It fits a local polynomial regression on the signal and requires *equidistant* bandwidth. Mathematically, it operates simply as a weighted sum of neighbouring values:

$$x_j* = \frac{1}{N}\sum_{h=-k}^{k} c_h x_{j+h}$$

where x_j* is the new value, N is a normalizing coefficient, k is the number of neighbour values at each side of j and c_h are pre–computed coefficients, that depends on the chosen polynomial order and degree (smoothing, first and second derivative)

```
# p = polynomial order w = window size (must be odd) m = m-th derivative
# (0 = smoothing) The function accepts vectors, data.frames or matrices.
# For a matrix input, observations should be arranged row-wise
sg.vec <- savitzkyGolay(NIRsoil$spc[1, ], p = 3, w = 11, m = 0)
sg <- savitzkyGolay(NIRsoil$spc, p = 3, w = 11, m = 0)
# note that bands at the edges of the spectral matrix are lost !
dim(NIRsoil$spc)
```

```
## [1] 825 700

dim(sg)

## [1] 825 690
```

2.2 Derivatives

Taking (numerical) derivatives of the spectra can remove both additive and multiplicative effects in the spectra and have other consequences as well (Table 3).

Table 3: Pro's and con's of using derivative spectra

Advantage	Drawback
Reduce of baseline offset	Risk of overfitting the calibration model
Can resolve absorption overlapping	Increase noise, smoothing required
Compensates for instrumental drift	Increase uncertainty in model coefficients
Enhances small spectral absorptions	Complicate spectral interpretation
Often increase predictive accuracy for complex datasets	Remove the baseline !

First and second derivatives of a spectrum can be computed with the finite difference method (difference between to subsequent data points), provided that the band width is constant:

$$x'_i = x_i - x_{i-1}$$

$$x''_i = x_{i-1} - 2 \cdot x_i + x_{i+1}$$

In R , this can be simply achieved with the diff function in base:

```r
# X = wavelength Y = spectral matrix n = order
d1 <- t(diff(t(NIRsoil$spc), differences = 1))  # first derivative
d2 <- t(diff(t(NIRsoil$spc), differences = 2))  # second derivative
plot(as.numeric(colnames(d1)), d1[1, ], type = "l", xlab = "Wavelength", ylab = "")
lines(as.numeric(colnames(d2)), d2[1, ], col = "red")
legend("topleft", legend = c("1st der", "2nd der"), lty = c(1, 1), col = 1:2)
```

One can see that derivatives tend to increase noise. One can use gap derivatives or the Savitzky-Golay algorithm to solve this. The gap derivative is computed simply as:

$$x'_i = x_{i+k} - x_{i-k}$$

$$x''_i = x_{i-k} - 2 \cdot x_i + x_{i+k}$$

where k is the gap size. Again, this can be easily achieved in R using the lag argument of the diff function

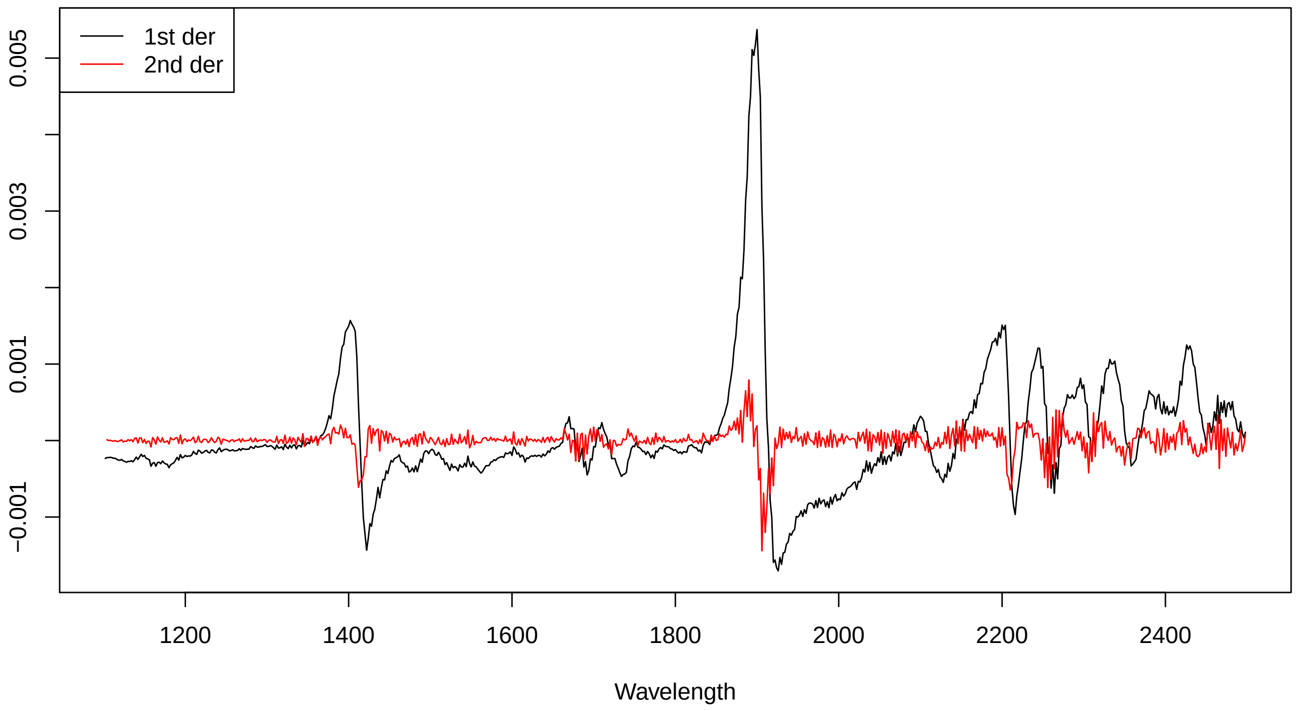

Figure 3: Effect of first derivative and second derivative

```
# first derivative with a gap of 10 bands
gd1 <- t(diff(t(NIRsoil$spc), differences = 1, lag = 10))
```

For more flexibility and control over the degree of smoothing, one could however use the Savitzky-Golay (`savitzkyGolay`) and Gap–segment derivative (`gapDer`) algorithms. The Gap-segment algorithms performs first a smoothing under a given segment size, followed by gap derivative. Here is an exemple of the use of the `gapDer` function.

```
# m = order of the derivative w = window size ( = {2 * gap size} + 1) s =
# segment size first derivative with a gap of 10 bands
gsd1 <- gapDer(X = NIRsoil$spc, m = 1, w = 11, s = 10)
plot(as.numeric(colnames(d1)), d1[1, ], type = "l", xlab = "Wavelength", ylab = "")
lines(as.numeric(colnames(gsd1)), gsd1[1, ], col = "red")
legend("topleft", legend = c("1st der", "gap-segment 1st der"), lty = c(1, 1),
    col = 1:2)
```

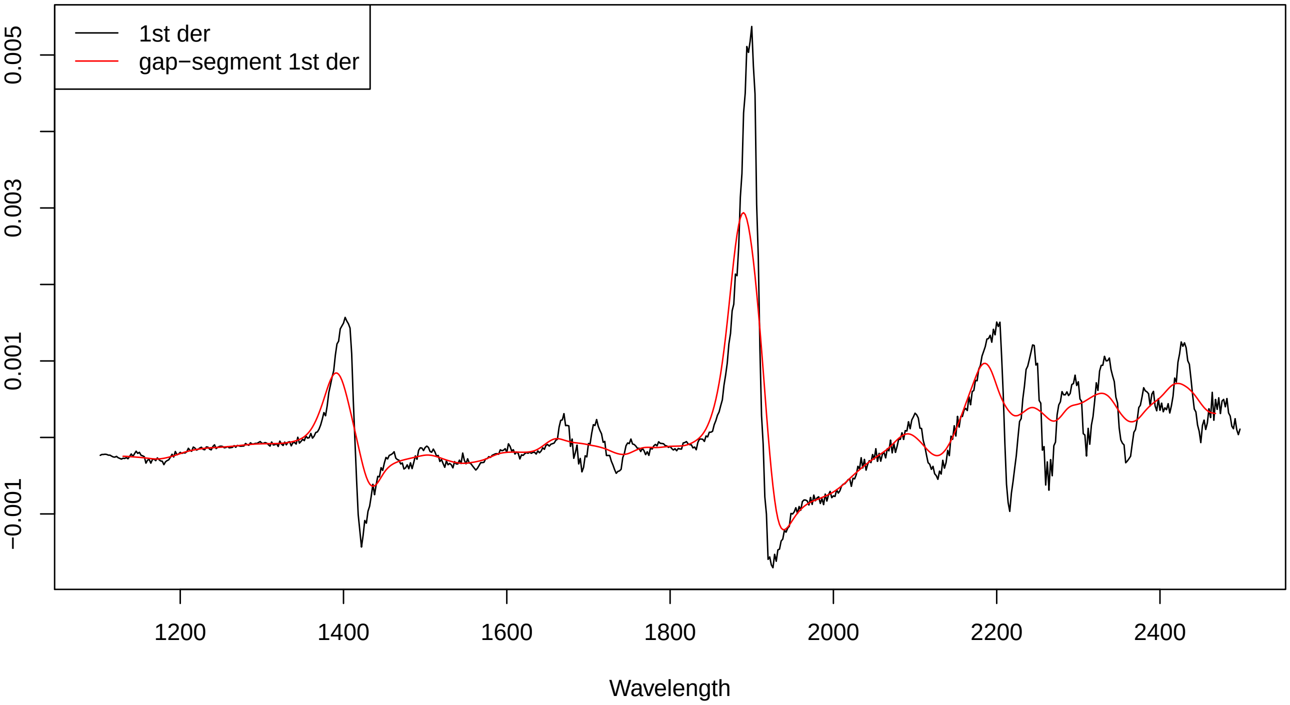

Figure 4: Effect of 1st-order gap-segment derivative

2.3 Scatter corrections

Undesired spectral variations due to light *scatter* effects and variations in effective *path length* can be removed using scatter corrections.

2.3.1 Standard Normal Variate (SNV)

Standard Normal Variate (SNV) is another simple way for normalizing spectra that intends to correct for light scatter. It operates row–wise:

$$SNV_i = \frac{x_i - \bar{x}_i}{s_i}$$

```r
snv <- standardNormalVariate(X = NIRsoil$spc)
```

According to Fearn [4], it is better to perform SNV transformation after filtering (by e.g. Savitzky–Golay) than the reverse.

2.3.2 SNV–Detrend

The SNV–Detrend [1] further accounts for wavelength-dependent scattering effects (variation in curvilinearity between the spectra). After a SNV transformation, a 2^{nd}–order polynomial is fit to the spectrum and subtracted from it

```r
# X = input spectral matrix wav = band centers
dt <- detrend(X = NIRsoil$spc, wav = as.numeric(colnames(NIRsoil$spc)))
plot(NIRsoil$spc[1, ], type = "l", xlab = "Band number", ylab = "")
par(new = T)
plot(dt[1, ], xaxt = "n", yaxt = "n", xlab = "", ylab = "", col = "red", type = "l")
axis(4, col = "red")
legend("topleft", legend = c("raw", "detrend signal"), lty = c(1, 1), col = 1:2)
```

```r
par(new = F)
```

2.4 Centering and scaling

Centering and scaling tranforms a given matrix to a matrix with columns with zero mean (centering), unit variance (scaling) or both (auto–scaling):

$$Xc_{ij} = X_{ij} - \bar{X}_j$$

$$Xs_{ij} = \frac{X_{ij} - \bar{X}_j}{s_j}$$

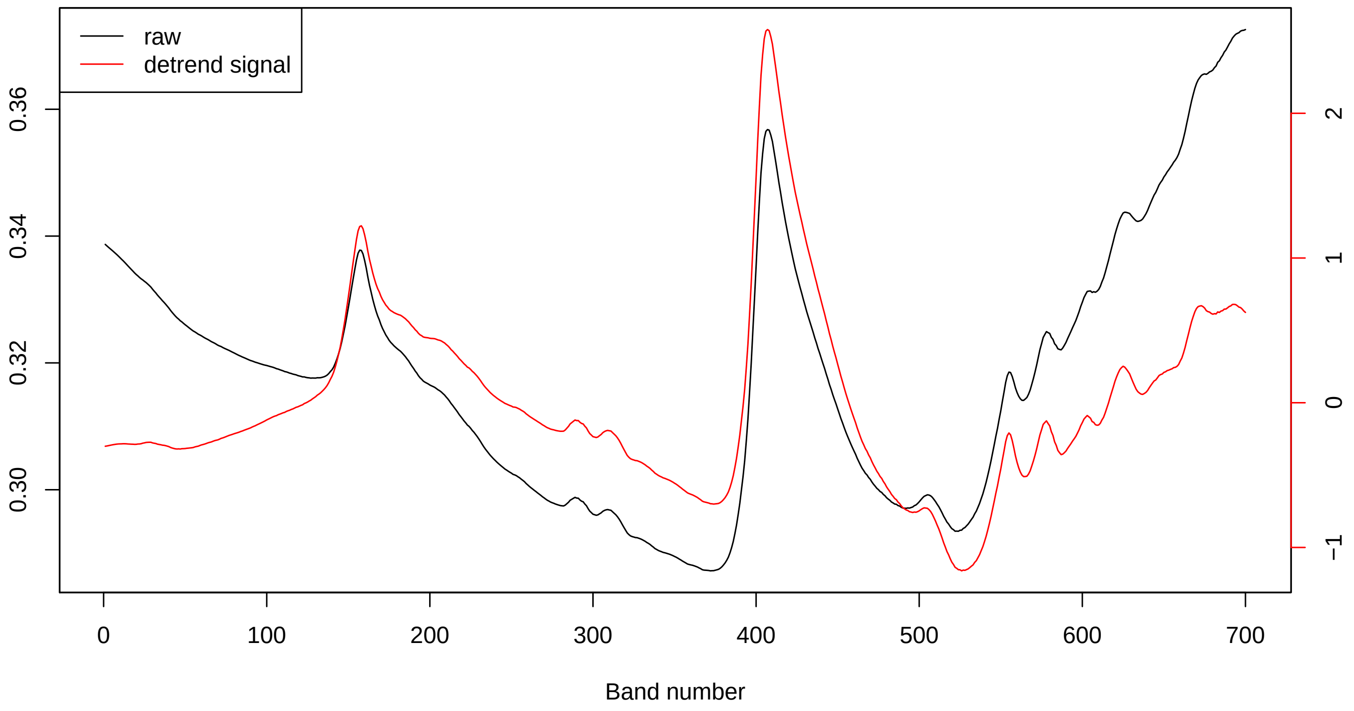

Figure 5: Effect of SNV-Detrend on raw spectra

where Xc and Xs are the mean centered and auto-scaled matrices, X is the input matrix, $\bar{X}_j$ and s_j are the mean and standard deviation of variable j.

In R , these operations are simply obtained with the `scale` function. Other types of scaling can be considered. Spectroscopic models can often be improved by using ancillary data (e.g. temperature, ...) [5]. Due to the nature of spectral data (multivariate), other data would have great chance to be dominated by the spectral matrix and have no chance to contribute significantly to the model due to purely numerical reasons [3]. One can use *block scaling* to overcome this limitation. It basically uses different weights for different block of variables. With *soft block scaling*, each block is scaled (i.e. each column divided by a factor) such that the sum of their variance is equal to the square root of the number of variables in the block. With *hard block scaling*, each block is scaled such that the sum of their variance is equal to 1.

```
# X = spectral matrix
# type = "soft" or "hard"
# The ouptut is a list with the scaled matrix (Xscaled) and the divisor (f)
bs <- blockScale(X=NIRsoil$spc,type="hard")$Xscaled
sum(apply(bs,2,var)) # this works!

## [1] 1
```

The problem with *block scaling* is that it down–scale all the block variables to the same variance. Since sometimes this is not advised, one can alternatively use *sum of squares block weighting* . The spectral matrix is multiplied by a factor to achieve a pre–determined sum of square:

```
# X = spectral matrix targetnorm = desired norm for X
bn <- blockNorm(X = NIRsoil$spc, targetnorm = 1)$Xscaled
sum(bn^2)  # this works!

## [1] 1
```

2.5 Other transformations

2.5.1 Continuum removal

The continuum removal technique was introduced by [2] as an effective method to highlight absorption features of minerals. It can be viewed as an albedo normalization technique. This technique is based on the computation of the continuum (or envelope) of a given spectrum. The continuum-removed spectrum of a given spectrum is computed as follows:

1. The local reflectance spectrum maxima points (in the case of absorbance, local minima points) are identified.

2. Then, these points are connected by linear interpolation to form the continuum c.

3. The continuum-removed spectrum is given by $\phi_i = \frac{x_i}{c_i}; i = \{1, ..., p\}$, where x_i and c_i are the original and the continuum reflectance (or absorbance) values respectively at the i^th wavelength of a set of p wavelengths, and ϕ_i is the respective final reflectance (or absorbance) value after continuum removal.

The `continuumRemoval` function allows to compute the continuum-removed values of either reflectance or absorbance spectra.

```
# type of data: 'R' for reflectance (default), 'A' for absorbance
cr <- continuumRemoval(X = NIRsoil$spc, type = "A")
# plot of the 10 first abs spectra
matplot(as.numeric(colnames(NIRsoil$spc)), t(NIRsoil$spc[1:10, ]), type = "l",
    ylim = c(0, 0.6), xlab = "Wavelength /nm", ylab = "Absorbance")
matlines(as.numeric(colnames(NIRsoil$spc)), t(cr[1:10, ]))
```

2.5.2 Resampling

To match the response of one instrument with another, a signal can be resampled to new band positions by simple interpolation (`resample`) or using full width half maximum (FWHM) values (`resample2`).

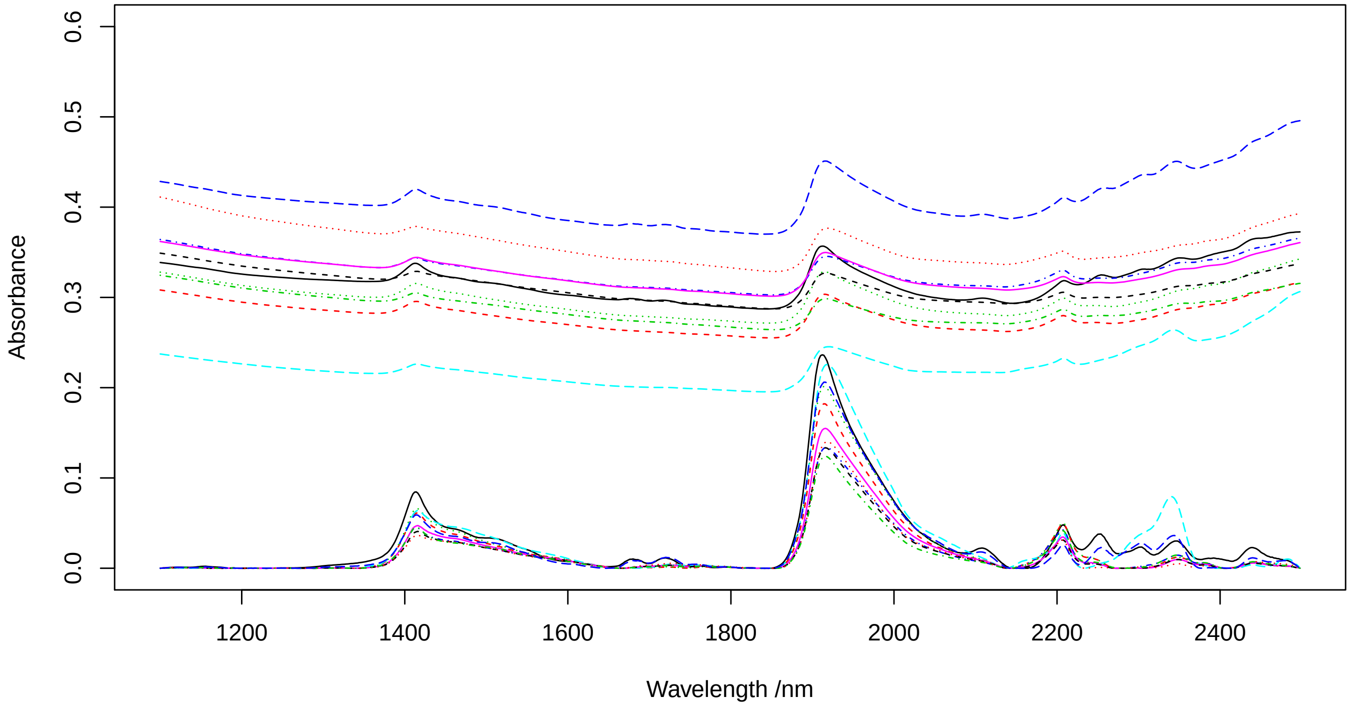

Figure 6: Absorbance and continuum-removed absorbance spectra

3 Calibration sampling algorithms

Calibration models are usually developed on a *representative* portion of the data (training set) and validated on the remaining set of samples (test/validation set). There are several solutions for selecting samples, e.g.:

- random selection (see e.g. `sample` function in `base`)
- stratified random sampling on percentiles of the response y (see e.g. `createDataPartion` in the `caret` package)
- use the spectral data.

For selecting representative samples, the `prospectr` package provides functions that use the third solution. The following functions are available: `kenStone` [8], `duplex` [15], `puchwein` [11], `shenkWest` [13], `naes` [10], `honigs` [7].

3.1 Kennard-Stone sampling (`kenStone`)

To sample a subset of n samples $X_{tr} = \left\{x_{tr j}\right\}_{j=1}^{n}$, from a given set of N samples $X = \{x_i\}_{i=1}^{N}$ (note that $N > n$) the Kennard-Stone (CADEX) sampling algorithm consists in [8]:

1. Find in X the samples x_{tr1} and x_{tr2} that are the farthest apart from each other, allocate them in X_{tr} and remove them from X.

2. Find in X the sample x_{tr3} with the maximum dissimilarity to X_{tr}. Allocate x_{tr3} in X_{tr} and then remove it from X. The dissimilarity between X_{tr} and each x_i is given by the minimum distance of any sample allocated in X_{tr} to each x_i. In other words, the selected sample is one of the nearest neighbours of the points already selected which is characterized by the maximum distance to the other points already selected.

3. Repeat the step 2 n-3 times in order to select the remaining samples $(x_{tr4}, ..., x_{trn})$.

The Kennard–Stone algorithm allows to create a calibration set that has a flat distribution over the spectral space. The metric used to compute the distance between points can be either the Euclidean distance or the Mahalanobis distance. Let's see some examples . . .

```
# Create a dataset for illustrating how the calibration sampling
# algorithms work
X <- data.frame(x1 = rnorm(1000), x2 = rnorm(1000))
plot(X)
# kenStone produces a list with row index of the points selected for calibration
ken <- kenStone(X,k=40)
points(X[ken$model,],col=2,pch=19,cex=1.4) # plot selected points
```

```
# Test with the NIRsoil dataset
# one can use the mahalanobis distance
# The mahalanobis distance is computed by taking the Euclidean distance
# in the normalized principal component score space
# (see Maesschalck et al. 2000, Chemo. Int. Lab. Syst. 50, 1-18)
# If the 'pc' argument is set, the Mahalanobis distance will be used
# The 'pc' argument is the number of pc's retained in the computation of the distance
# if 'pc' < 1, then it corresponds to a treshold in terms of explained variance
ken_mahal <- kenStone(X = NIRsoil$spc, k = 20, pc= .999)
# The pc components in the output list stores the pc scores
plot(ken_mahal$pc[,1],ken_mahal$pc[,2],xlab="PC1",ylab="PC2")
# This is the selected points in the pc space
points(ken_mahal$pc[ken_mahal$model,1],ken_mahal$pc[ken_mahal$model,2],pch=19,col=2)
```

3.2 DUPLEX (duplex)

The Kennard–Stone algorithm selects calibration samples. Often, we need also to select a validation subset. The DUPLEX algorithm [15] is a modification of the Kennard-Stone which allows to select a validation set that have similar properties to the calibration set. DUPLEX, similarly to Kennard–Stone, begins by selecting pairs of points that are the farthest apart from each other, and then assigns points alternatively to the calibration and validation sets.

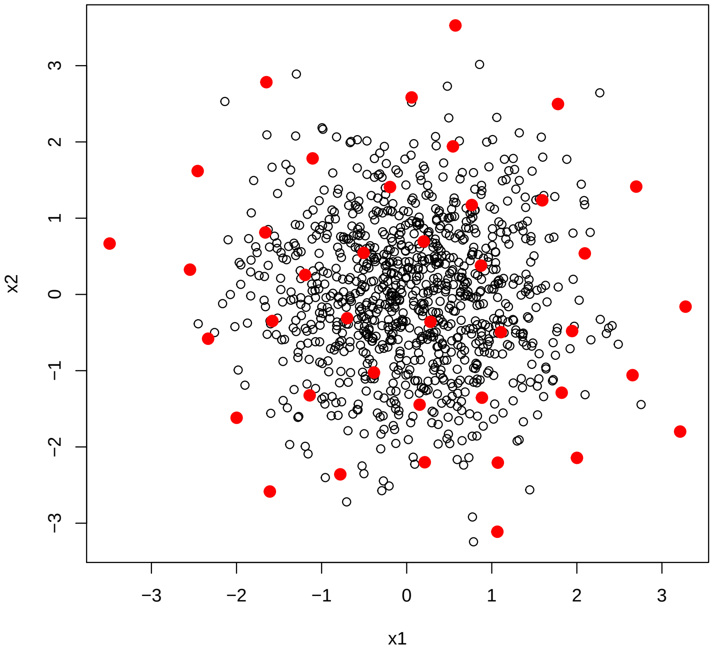

Figure 7: Selection of 40 calibration samples with the Kennard-Stone algorithm

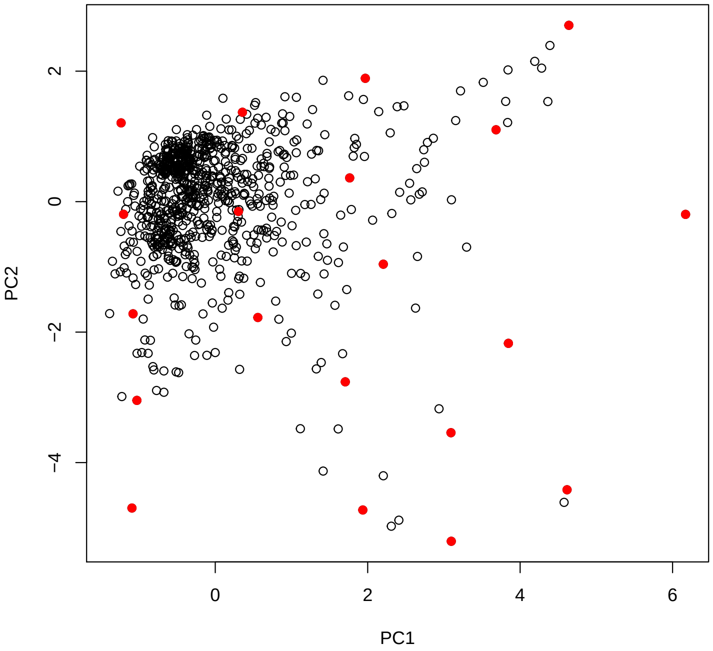

Figure 8: Kennard-Stone sampling on the NIRsoil dataset

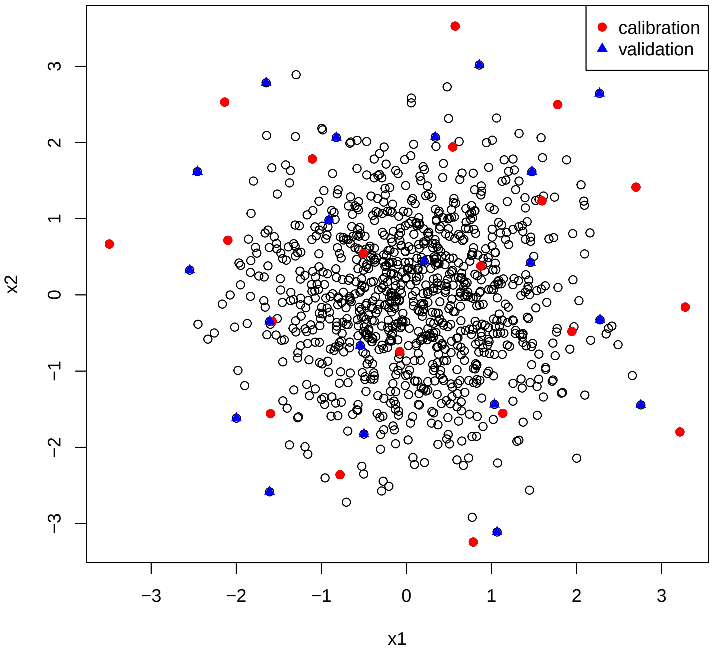

Figure 9: Selection of 15 calibration and validation samples with the DUPLEX algorithm

```r
dup <- duplex(X = X, k = 15)  # k is the number of selected samples
plot(X)
points(X[dup$model, 1], X[dup$model, 2], col = "red", pch = 19)  # calibration samples
points(X[dup$test, 1], X[dup$test, 2], col = "blue", pch = 17)  # validation samples
legend("topright", legend = c("calibration", "validation"), pch = c(19, 17),
    col = c("red", "blue"))
```

3.3 k-means sampling (naes)

The k-means sampling simply uses k-means clustering algorithm. To sample a subset of n samples $X_{tr} = \left\{x_{trj}\right\}_{j=1}^{n}$, from a given set of N samples $X = \{x_i\}_{i=1}^{N}$ (note that $N > n$) the algorithm works as follows:

1. Perform a k-means clustering of X using n clusters.

2. Extract the n centroids (c, or prototypes). This can be also the sample that is the farthest away from the centre of the data, or a random selection. See the `method` argument in `naes`.

3. Calculate the distance of each sample to each c.

4. For each c allocate in X_{tr} its closest sample found in X.

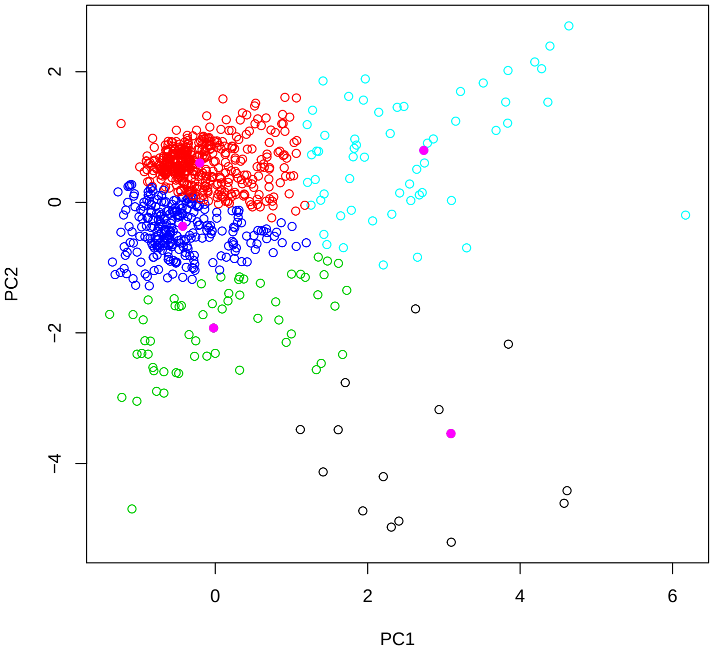

Figure 10: Selection of 5 samples by k-means sampling

```
# X = the input matrix
# k = number of calibration samples to be selected
# pc = if pc is specified, k-mean is performed in the pc space
# (here we will use only the two 1st pcs)
# iter.max =  maximum number of iterations allowed for the k-means clustering.
kms <- naes(X = NIRsoil$spc, k = 5, pc = 2, iter.max = 100)
# Plot the pcs scores and clusters
plot(kms$pc,col=kms$cluster)
# Add the selected points
points(kms$pc[kms$model,],col=6,pch=19)
```

3.4 SELECT algorithm (shenkWest)

The SELECT algorithm [13] is an iterative procedure which selects the sample having the maximum number of neighbour samples within a given distance (`d.min` argument) and remove the neighbour samples of the selected sample from the list of points. The number of selected samples depends on the chosen treshold (default = 0.6). The distance metric is the Mahalanobis distance divided by the number of dimensions (number of pc components) used to compute the distance. Here is an example of how the `shenkWest` function might work:

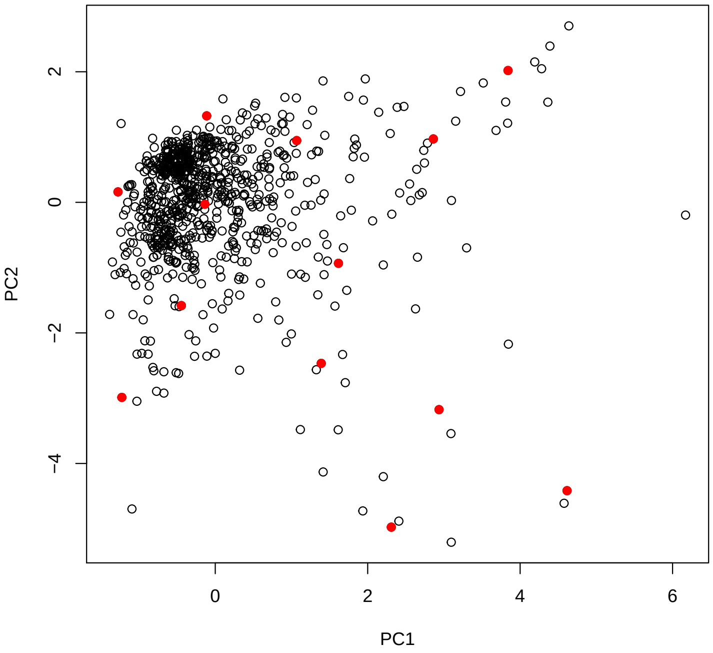

Figure 11: Selection of samples with the SELECT algorithm

```
shenk <- shenkWest(X = NIRsoil$spc, d.min = 0.6, pc = 2)
plot(shenk$pc)
points(shenk$pc[shenk$model, ], col = 2, pch = 19)
```

3.5 Puchwein algorithm (puchwein)

The Puchwein algorithm is yet another algorithm for calibration sampling [11] that creates a calibration set with a flat distribution. A nice feature of the algorithm is that it allows an objective selection of the number of required calibration samples with the help of plots. First the data is usually reduced through PCA and the most significant PCs are retained. Then the mahalanobis distance (H) to the center of the matrix is computed and samples are sorted decreasingly. The distances betwwen samples in the PC space are then computed.

Here is a *pseudo-code* of the algorithm:

1. Definition of a limiting distance
2. Find the sample with max(H)
3. Remove all the samples which are within the limiting distance away from the sample selected in step 2.
4. Go back in step 2 and find the sample with max(H) within the remaining samples

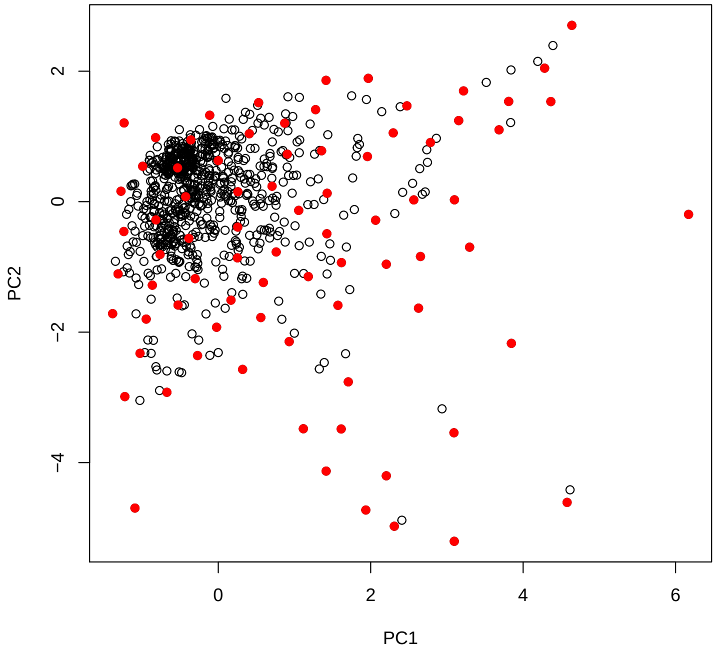

Figure 12: Samples selected by the Puchwein algorithm

5. When there is no sample anymore, go back to step 1 and increase the limiting distance.

```
pu <- puchwein(X = NIRsoil$spc, k = 0.2, pc = 2)
plot(pu$pc)
points(pu$pc[pu$model, ], col = 2, pch = 19)  # selected samples
```

The number of sample selected depends on the limiting distance. To help choosing the appropriate number of samples, two plots are used [14]:

- a plot showing the number of samples that are removed in each loop and the total number of samples left
- a plot showing the theoretical sum of leverages (each sample has the same leverage) together with the true sum of leverages. The number of samples removed that reach the maximum difference between the two can be considered to be optimal

```
# Optimal loop
par(mfrow = c(2, 1))
plot(pu$leverage$removed, pu$leverage$diff, type = "l", xlab = "# samples removed",
    ylab = "Difference between th. and obs sum of leverages")
```

```
# This basically shows that the first loop is optimal
plot(pu$leverage$loop, nrow(NIRsoil) - pu$leverage$removed, xlab = "# loops",
    ylab = "# samples kept", type = "l")
```

```
par(mfrow = c(1, 1))
```

3.6 Honigs (honigs)

The Honigs algorithm selects samples based on the size of their absorption features [7]. It can works both on absorbance and continuum-removed spectra. The sample having the highest absorption feature is selected first. Then this absorption is substracted from other spectra and the algorithm iteratively select samples with the highest absorption (in absolute value) until the desired number of samples is reached.

```
ho <- honigs(X = NIRsoil$spc, k = 10, type = "A")
# plot calibration spectra
matplot(as.numeric(colnames(NIRsoil$spc)), t(NIRsoil$spc[ho$model, ]), type = "l",
    xlab = "Wavelength", ylab = "Absorbance")
# add bands used during the selection process
abline(v = as.numeric(colnames(NIRsoil$spc))[ho$bands], lty = 2)
```

References

[1] R.J. Barnes, M.S. Dhanoa, and S.J. Lister. Standard normal variate transformation and de-trending of near-infrared diffuse reflectance spectra. *Applied spectroscopy*, 43(5):772–777, 1989.

[2] Roger N. Clark and Ted L. Roush. Reflectance spectroscopy: Quantitative analysis techniques for remote sensing applications. *Journal of Geophysical Research*, 89(B7):PP. 6329–6340, 1984.

[3] Lennart Eriksson, Erik Johansson, Nouna Kettaneh, Johan Trygg, C. Wikstrom, and Svante Wold. *Multi- and Megavariate Data Analysis*. MKS Umetrics AB, 2006.

[4] Tom Fearn. The interaction between standard normal variate and derivatives. *NIR news*, 19(7):16, 2008.

[5] Tom Fearn. Combining other predictors with NIR spectra. *NIR news*, 21(2):13, 2010.

[6] J. A. Fernandez Pierna and P. Dardenne. Soil parameter quantification by NIRS as a chemometric challenge at chimiometrie 2006. *Chemometrics and Intelligent Laboratory Systems*, 91(1):94–98, 2008.

[7] D. E. Honigs, Gary M. Hieftje, H. L. Mark, and T. B. Hirschfeld. Unique-sample selection via near-infrared spectral subtraction. *Analytical Chemistry*, 57(12):2299–2303, October 1985.

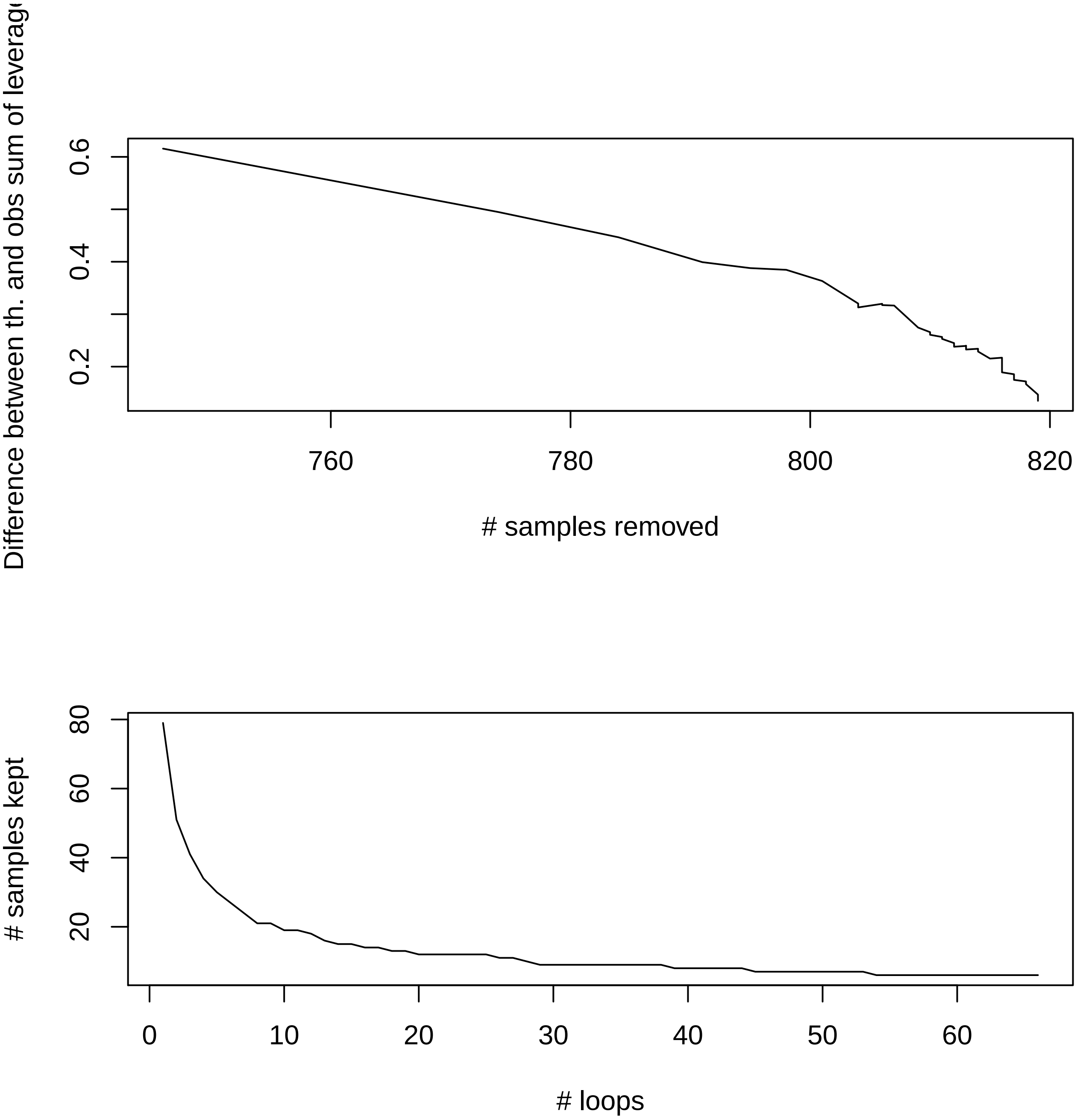

Figure 13: How to find the optimal loop

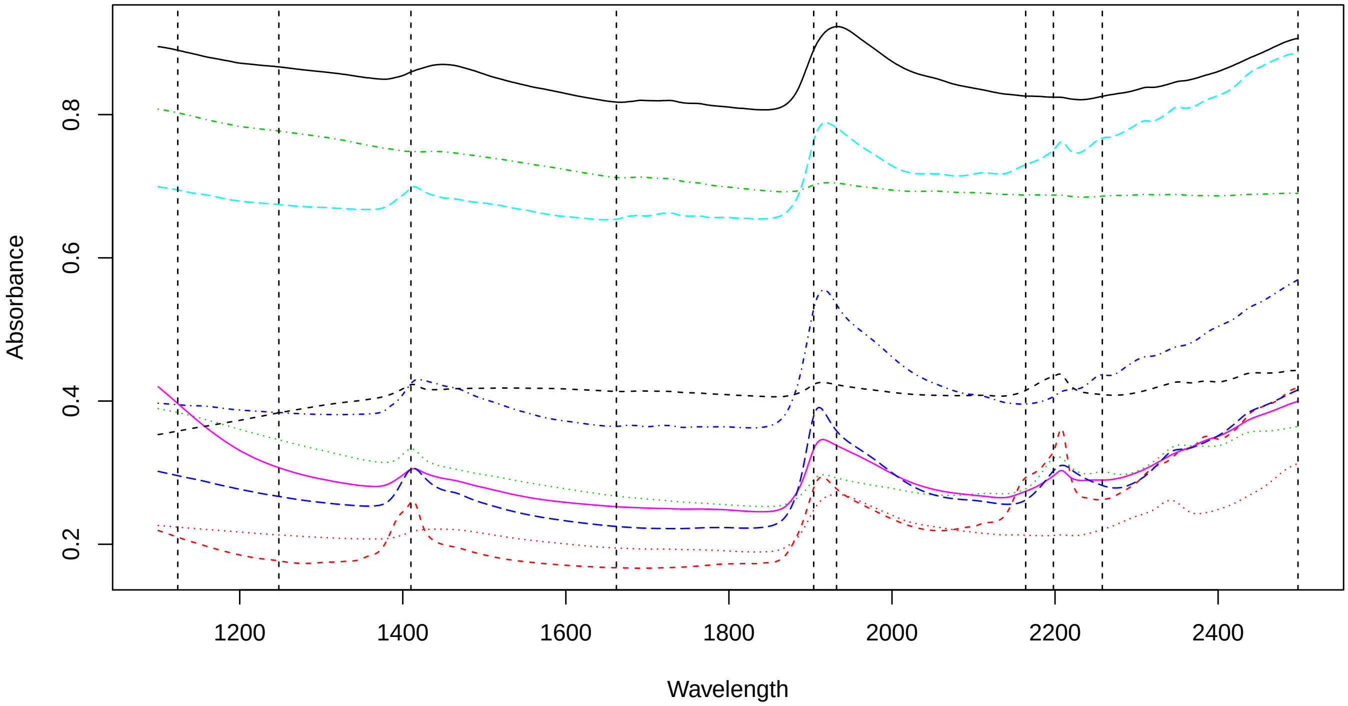

Figure 14: Spectra selected with the Honigs algorithm and bands used

[8] R. W. Kennard and L. A. Stone. Computer aided design of experiments. *Technometrics*, 11(1):137–148, February 1969.

[9] Katharine M. Mullen and Ivo HM van Stokkum. An introduction to the special volume spectroscopy and chemometrics in r. *Journal of Statistical Software*, 18(1):1–5, 2007.

[10] T. Naes, T. Isaksson, T. Fearn, and T. Davies. *A user friendly guide to multivariate calibration and classification*. NIR Publications, Chichester, United Kingdom, 2002.

[11] Gerd Puchwein. Selection of calibration samples for near-infrared spectrometry by factor analysis of spectra. *Analytical Chemistry*, 60(6):569–573, 1988.

[12] Abraham. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Anal. Chem.*, 36(8):1627–1639, 1964.

[13] J. S. Shenk and M. O. Westerhaus. Population definition, sample selection, and calibration procedures for near infrared reflectance spectroscopy. *Crop Science*, 31(2):469–474, 1991.

[14] Nisha Shetty, Asmund Rinnan, and Rene Gislum. Selection of representative calibration sample sets for near-infrared reflectance spectroscopy to predict nitrogen concentration in grasses. *Chemometrics and Intelligent Laboratory Systems*, 111(1):59–65, February 2012.

[15] R. D. Snee. Validation of regression models: methods and examples. *Technometrics*, 19(4):415–428, 1977.