SAISIR

A COMPREHENSIVE CHEMOMETRICS TOOLBOX UNDER MATLAB ®

Getting started

Package CD-Rom de SAISIR©

Révision Octave, Scilab
Dominique Bertrand
INRAe Nantes
dataframe@free.fr

Révision Matlab
Christophe B.Y. Cordella
INRAe Paris / AgroParisTech
christophe.cordella@agroparistech.fr
christophe.cordella@fsaa.ulaval.ca

Révision 2024

<u>Dear chemometrician</u>
Welcome to the SAISIR toolbox!

- It is not possible to start using SAISIR without understanding the data structure (which is simple). Please read at least pargraphs 1 and 2

*- The use of **SAISIR** requires a basic knowledge of the standard MATLAB environment and programming.*

*- All the functions in **Saisir** have a help which can be obtained from*
help <function name>.
The list of the available main functions can be obtained from the command help **Saisir**

- <u>A trick</u>: if the output of a function (which is very often a structure) is not very clear, it is always possible to enter the name of the output argument. The names of the fields are often easy to understand.
For example:
```
>>myresult=pca(wheat) ; %compute a PCA on the SAISIR file "wheat"
```
But what contains myresult?
```
>>myresult
   score: [1x1 struct]
   eigenvec: [1x1 struct]
   eigenval: [1x1 struct]
   average: [1x1 struct]
   info: 'PCA 08-Jan-2024 16:06:46'
```

One can imagine that myresult.score *gives the scores,* myresul.eigenvec *the eigenvectors and so on!*

<u>Installing SAISIR</u>

Build a folder and copy all the SAISIR functions in it.
In MATLAB, use the "*File/set path*" menu to add this folder in your path.
Verify that everything is OK, under MATLAB by typing help saisir or what saisir.

Introduction:

In most of the chemometric works, it is needed to manipulate large data files and to have the capability to make complex chain of processing steps. For example, a simple data processing may consist in loading data such as spectra, numeric images, sensory data …, pre-treating them by specific relevant methods, sampling the data, and eventually processing them using PCA, PLS, or any adapted methods (multiblock table methods, discrimination …). Moreover, it is essential for the user to keep the trace of the identifiers of the rows and the columns of the matrix data. For example, when processing spectra, it is important to identify the particular role of given variables (labeled, for example, by a wavenumber or a wavelength values). In the same way, the rows of the data matrices are generally logically identified by *names making sense to the chemometrician*. When the chain of procedures is complex, it is not easy to make such a work in graphically interfaced environments. One may want to keep the trace of the applied procedures and to be able to re-use them on other data sets, or in cross-validation procedures.
The SAISIR environment has been developed in order to cope with such situations. The (French) acronym of SAISIR is *Statistiques Appliquées à l'Interprétation des Spectres Infrarouge,* which can be (tortuously) translated by *Statistics Applied to the Interpretation of Spectra in the InfraRed*. The French verb "**SAISIR**" means "to hold" but also "to understand". SAISIR contains many functions for loading, saving, manipulating or displaying data. It is well equipped with chemometric functions for regression, multidimensional analysis, discrimination and multiblock analysis … A module devoted to image analysis will be also available, with many useful functions for manipulating multivariate images or for image texture analysis.
The use of SAISIR requires a basic knowledge of MATLAB programming.

The functions are (alas!) given with no guarantee of any kind on their accuracy in any situation. But, dear Chemometrician, if you identify a bug, please help our community by reporting it to Dominique Bertrand, (*bertrand@enitiaa-nantes.fr*)

1. General principle

The SAISIR environment mainly lies on the processing of data tables, as matrices of real (double) number. The fundamental notion is that <u>the rows and the columns of the matrices in SAISIR have identifiers</u> (names) which must be kept in any function.

Let us, for example, examine the simple data set below. It is the result of the assessment of three apples by a given panelist. The apples are identified by their cultivar names " GALA1", "FUJI1", " FUJI2 ".

The panelist has assessed three sensory features of the fruits, namely "global odor" coded by "GLO", earth odor ("EARTH"), fungus odor ("FUNG").

	GLO	EARTH	FUNG
GALA1	2.8	1.2	0.3
FUJI1	2.6	0.5	0.4
FUJI2	7.5	0.3	0

(Important: the decimal separator is the point)

This file includes 3 rows and 3 columns. The rows (representing the observations or *individuals*) are characterized by the identifiers (« GALA1 », « FUJI1 » and « FUJI2 »). In the same way, the columns (*variables*) are named "GLO", "EARTH", "FUNG". At last, the numeric data build up the matrix:

2.8	1.2	0.3
2.6	0.5	0.4
7.5	0.3	0

It is in general necessary to keep these three pieces of information.

Under SAISIR, the matrix with the identifiers is kept in a MATLAB structure with fields "d" for the numerical data, "i" for the identifiers and "v" for the variables.

Let us suppose that the previous data are kept in a structure named "APPLE». We will have:

```
>> APPLE
APPLE =
d: [3x3 double]
i: [3x5 char]
v: [3x20 char]
```

It is of course easy to extract the field using the MATLAB extraction command « . ». For example:

```
>> APPLE.i
GALA1
FUJI1
FUJI2
```

If (who knows?) you are bored with SAISIR, you can easily go out and retrieve the good old MATLAB matrices. For example:

```
>> X=APPLE.d

X =

        2.80        1.20        0.30
        2.60        0.50        0.40
        7.50        0.30          0
```

It is important to notice that the field « i » and « v » are matrices of characters and not *cell array*. Thus, for example, APPLE.i(:,1) is meaningful and gives the first character of all the row identifiers:

```
                    G
                    F
                    F
```

The first dimension of these matrices of characters is important and must correspond to the number of rows or columns (3 for both, in the example). However, the second dimension is not important. In operations such as merging files, SAISIR automatically adds the necessary space character.

A particularly important case deals with the data which are digitized curves such as spectra or chromatograms. In this case, the data are generally under the form of matrices with the rows corresponding to the curves. The identifiers of variables are then numbers, transformed in character strings, and normally represent the X-graduation of the curve (wavelength, retention time, shift …). Here is a simplified example of a file of Near Infrared data:

	1100	1102	1104	1106	1108
1br01	0.20541	0.20723	0.20908	0.21099	0.21293
1br51	0.21421	0.21611	0.21805	0.22002	0.22201
1fu21	0.17093	0.1725	0.1741	0.17574	0.17741
1fu71	0.17365	0.17514	0.17667	0.17823	0.17981

As previously « 1br01 », « 1br51 », « 1fu21 », « 1fu71 » are the identifiers of individuals, whereas « 1100 », « 1102 », « 1104 », « 1106 », « 1108 » are character strings identifying the variables. If the data are in a SAISIR structure spectra, we will have:

```
>>spectra

   d: [4x5 double]
   i: [4x31 char]
   v: [5x4 char].
```

and

```
>> spectra.v
1100
1102
1104
1106
1108
```

The numerical nature of the descriptors is exploited in the function devoted to displaying curves such as (curve, curves, tcurve, tcurves …).

2. Use of the SAISIR environment

One may have a list of the main functions in SAISIR by the command

```
>>help saisir
```

For each function, the command help "function name" gives a (short!) explanation of the function. You will find a complete list of function classified alphabetically or by themes in the document saisir_documentation.html, which can be read using a Web explorer.

2.1 Getting started

For working in the SAISIR environment, the user must have the data under the form of vectors or rectangular tables (matrices). In its current state, SAISIR is not able to cope with missing data (NaN values). There only simple procedures to eliminate rows or columns which contain NaN data (see eliminante_nan).

2.1.1 Loading, creating and saving SAISIR structures

Creation from matrices currently in the MATLAB workspace

From matrices currently in the MATLAB workspace, one can "manually" build up a SAISIR structure. If only a MATLAB matrix is available, the function matrix2saisir can be used.

For example:

```
>>a=rand(5,2) ;% matrix of random number  5x2
>>b=matrix2saisir(a) ;
b =
   d: [5x2 double]
   i: [5x1 char]
   v: [2x1 char]
```

In this case, the identifiers of rows and columns are simply character strings representing the index of row or column.

Thus; we have:

```
>> b.i

1
2
3
4
5
```

It is possible to add a prefix in the identifiers, for example:

```
>> b=matrix2saisir(a,'sample','column')
b =
   d: [5x2 double]
   i: [5x6 char]
   v: [2x4 char]
```

We then have, for example :

```
>> b.v
column1
column2
```

If some identifiers of observations are available, it is still possible to build up the whole SAISIR structure, for example:

```
>>identifiers={'thing' 'object' 'element' 'entity'}
>>b.i=char(identifiers) ;
>> b.i
thing
object
element
entity
```

It must noticed that the *array of cells* (with accolades "{ }") can be transformed in a matrix of characters by the MATLAB function char. The matrix is then correctly filled with space characters.

Loading from excel files

It is possible to load data from excel files using a very simple format in Excel.
The single format compatible with SAISIR is the following:

	GLO	EARTH	FUNG
GALA1	2.8	1.2	0.3
FUJI1	2.6	0.5	0.4
FUJI2	7.5	0.3	0

The first row contains the identifiers of the columns. The first column contains the identifiers of rows. The other cells are numerical values. The other cells contain *numerical values* with the point as decimal separator.
The (eventual) content of the cell in position (1,1) is not kept by SAISIR.
The Excel files must be saved in the Excel format ".CSV" (it is an ASCII format, with semicolon as separator). In Excel, choose save as and CSV (separator semicolon).
Let us suppose that the previous example has been saves under the name 'apple.csv'.
Then in MATLAB, you must set the directory where this file has been saved as the current directory and use the command (for example):

```
apple=excel2saisir('apple.csv');
```

The general command is:

```
[res] = excel2saisir(filename) ;
```

filename is a string of characters and res is the SAISIR structure.
The function excel2saisir suppresses the rows or the columns which are formed with completely empty cells. Otherwise, the empty cells in Excel, or the ones containing non numerical data are replaced by NaN (*not a number*).
It is possible to build a SAISIR structure with no missing data using eliminate_nan.
Examples:

```
reduced = eliminate_nan1(table,0) ;% maximum of rows, suppressing columns
reduced = eliminate_nan1(tableau,1) ;% maximum de columns, suppressing the row
```

Of course, this way of eliminating missing data is rather brutal!!

Saving SAISIR structure

The function saisir2excel makes it possible to save data in a format readable by Excel.

Syntax:

```
saisir2excel(saisir,filename)
```

Example:

```
saisir2excel(apple,'apple.csv')
```

Save the SAISIR structure under the name essai.CSV

The first argument is a MATLAB variable identifying a SAISIR structure, whereas the second one is a character string (It is thus necessary to use apostrophe ').

The file is saved with the identifiers of rows and columns at the Excel format ".CSV".

If the number of columns is greater than 255, Excel is not able to read the whole .CSV file. If the number of rows is less than 255, it is still possible to save the transposed matrix, using saisir_transpose.

2.1.2 Elementary data manipulations

Note: in the following, we will call "matrices" the SAISIR structure.

The elementary manipulations deal with concatenation and suppression of rows and columns, The concatenation is designated by append and the suppression by delete.

We thus have the commands:

appendrow, deleterow

appendcol and deletecol

Example with concatenation

Let us consider two matrices with the *same number of columns*.

The command saisir3=appendrow(saisir1, saisir2) concatenate the matrices saisir1 and saisir2 with increasing the number of rows.

For example, let table1 have 10 rows and 25 columns, and table2 with 30 rows and 25 columns. The command:

```
whole=appendrow(table1, table2)
```

builds a new matrix called whole with 40 rows and 25 columns. The variables and observation names are obviously actually copied in whole.v and whole.i .

It is possible to concatenate an arbitrary number of matrices with the command appendrow1. For example:

```
whole=appendrow1(x1, x2, x3, x4).
```

appendcol and appendcol1 work in the same way, on the columns. In this case, the number of rows of the matrices must be identical.

For example if A is a matrix with 25 rows and 3 columns, and B a matrix with 25 rows and 12 columns, the command

```
C=appendcol(A,B)
```

builds a matrix C with 25 rows and 15 columns.

Suppression of rows and columns

We have the commands
deleterow, deletecol, selectrow and selectcol.
Example:
Let A be a matrix 15 x 20.
The command
C=deletecol(A,1)
builds a matrix C from A suppressing the first column of A.
Thus C is dimensioned 15 x 19.
The same principle exists for the rows, using deleterow.
It is also possible to identify the columns which are kept. For example; the command C=selectcol(A,1:2:8) selects the columns 1, 3, 5, 7 of A which are placed in C. The same principle applies with the rows (command selectrow).
Warning! It must be noticed that in the case of spectral data and other digitized curves, the variables are measurements at given time of wavenumber. However, there is no immediate correspondence between the identifier of the variable (character string representing a number) and its index (rank).
For example, in the following matrix of NIR spectra, beginning with:

	1100	1102	1104	1106	1108
1br01	0.20541	0.20723	0.20908	0.21099	0.21293
1br51	0.21421	0.21611	0.21805	0.22002	0.22201
1fu21	0.17093	0.1725	0.1741	0.17574	0.17741
1fu71	0.17365	0.17514	0.17667	0.17823	0.17981

The first variable represents the wavelength « 1100 » .
The variable at the 4th position represents the wavelength « 1106 » .
This, for selecting the row associated with wavelength « 1104 », one must write:

lo1104=selectcol(spectra, 3) ; and not lo1104=selectcol(spectra, 1104).

It must be tedious in large matrices, to find the rank of an observation or a variable, when knowing its identifier. For finding the rank, one can use the command seekstring, for example:
index = seekstring (spectre.v, ‘1104’)
gives the ranks of the variables identified by “ 1104 ».
In the same way,
index = seekstring (spectre.i, ‘golden’)
returns the rank of all the observations which have “golden” in their name. Lower and uppercase characters are considered as different.

In the case of numerical curve, the command find_index can be used for finding the index of a given wavelength.
For example:
index=find_index(spectra.v,1940)
will give the index of the variable the value of which is close to 1940.

Use of identifiers as extraction key
When dealing with large matrices, it is almost compulsory to make use of identifiers able to build up an extraction key system. Many SAISIR procedures (discriminant analysis, principal component analysis, graphical display) are strongly simplified if the user has respected this principle.
It is easy to understand the system by taking an example.
Let us imagine that a given experiment deals with 3 wheat cultivars (*Camp Rémi*, *Talent* and *Arminda*), grown at two locations (*Paris* and *Montpellier*). Twenty replicates are made for each cultivar and each location.
In this case, a correct sample descriptor can be, for example:
CRPA09 which means: *Camp Rémi*, grown at *Paris*, replicate 9.
We thus have decided that two letters (here CR) are used for designating the cultivar, two characters for the location (PA), two letters for the replicate. The 9th replicate is designated by '09' and not '9' for avoiding problems when the number of replicates is higher than 9!
In the same way, we will have TAMO19 : *Talent*, grown at *Montpellier*, replicate *19*.
We must, in this example, avoid changing the number of characters devoted to a given field. For example, CRMO12 and ARMPA1 are not compatible, because the code "CR" includes 2 characters, whereas the code "ARMS "includes 3 ones!
One can make use of the sample codes for extracting sub-files, with the command select_from_identifier. The syntax is:
[saisir1] = select_from_identifier(saisir,startpos,str)
Where startpos (starting position) indicates the place in the row identifiers where the character chain str (*string*) is researched. For example, the command:
Paris=select_from_identifier (wheat, 3 ,'PA') selects in the matrix wheat the samples whose identifiers contains PA in 3th and 4th position.

2.1.3 Developed examples: principal component analysis
Principal component analysis (PCA) can be used for giving a first glance to data, prior the applications of other methods.

A first example deals with the data set described in the article:
M. Forina and C. Armanino, Eigenvector Projection and Simplified Non-Linear Mapping of Fatty Acid Content of Italian Olive Oils, Annali di Chimica 72:127-141, (1987).

In order to characterize oils obtained from olive cultivated in different Italian regions, the authors have analyzed 8 fatty acids (*Palmitic*, *Palmitoleic*, *Stearic*, *Oleic*, *Linoleic*, *Eicosanoic*, *Linolenic*, *Eicosenoic*) of 572 samples of olive oils, picked up in 9 regions. The analytical data are eventually stored in an Excel file. The rows represent the 572 samples. The matrix includes 9 rows corresponding to a qualitative grouping indicating the region by an integer, followed by 8 fatty acid analyses. We thus have a data matrix dimensioned 572 x 9.
The sample identifiers include two characters indicating the growing location.

For example:

```
olive.i(5,:)
>>Ca005
```

indicates that the 5th observation has been collected in region Ca.
The Excel file is first saved at .CSV format, under the name olive.csv. The matrix is loaded in the MATLAB workspace using the command:

```
olive=excel2saisir('olive')
>>olive =
>>   v: [9x20 char]
>>   d: [572x9 double]
>>   i: [572x20 char]
```

For computing PCA, we must have a data matrix *without missing data* containing only the rows and the columns of active observations. *In this particular Excel file*, the first column of numerical values contains the qualitative group indicating the growing regions. Before computing PCA, we must first delete this column, which is not used in this analysis.

```
olive1=deletecol(olive,1);
```

As the data have different importance, it is necessary to *reduce*, *standardize* or *norm* the columns, *i.e.* dividing each column by its own standard-deviation. Warning! The PCA in SAISIR is not systematically carried out on reduced data!! This is done by using the command standardize.

```
olive2=standardize(olive1);
```

PCA is then computed by the command:

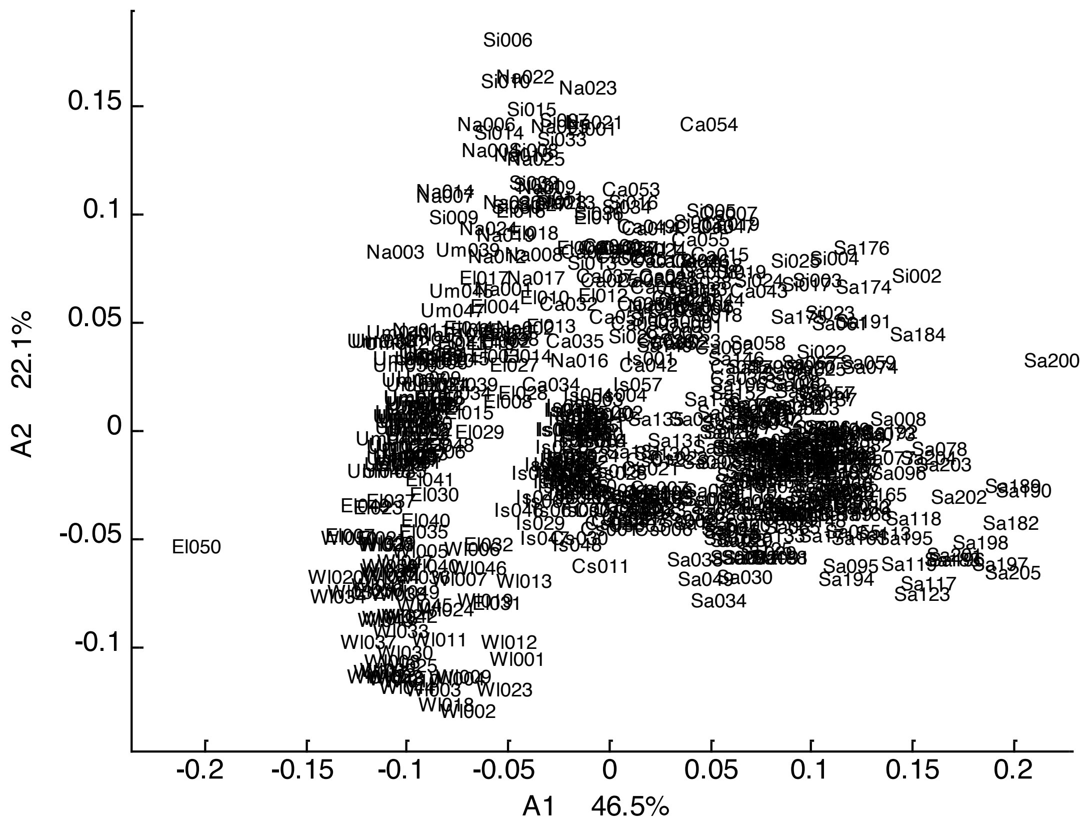
```
res=pca(olive2)
>>res =

>>      score: [1x1 struct]
>>   eigenvec: [1x1 struct]
>>    eigenval: [1x1 struct]
>>     average: [1x1 struct]
        info: 'PCA 06-Dec-2002 08:44:24'
```

Note: the function normed_pca standardizes the data before the PCA computation.
In this example, res is a structure giving the main results of PCA. Each field of this structure (score, eigenvec, eigenval, average) is itself a complete SAISIR structure. One can of course examine individually each result. The more important field is res.score. Each row of res.score represents an observation (or individual) whereas each column is a given component in the decreasing order of the eigenvalues. The identifiers of the rows of res .score, namely res .score.i are (logically) the copy of the identifiers in olive1.i. The identifiers of the columns of res .score are created by the command pca , and have the form:

```
A1   46.5%
A2   22.1%
A3   12.7%
A4    9.9%
A5    4.2%
A6    3.1%
A7    1.5%
A8      0%
```

The first characters represent the component rank (axis); the number indicates the percentage of inertia (percentage of explained variance) for the corresponding component. The factorial maps (or scatter plots) are representation of pairs of columns (chosen by the user) of res.score under the form of graphical X-Y display. One can obtained such representation using the command map and its variants.

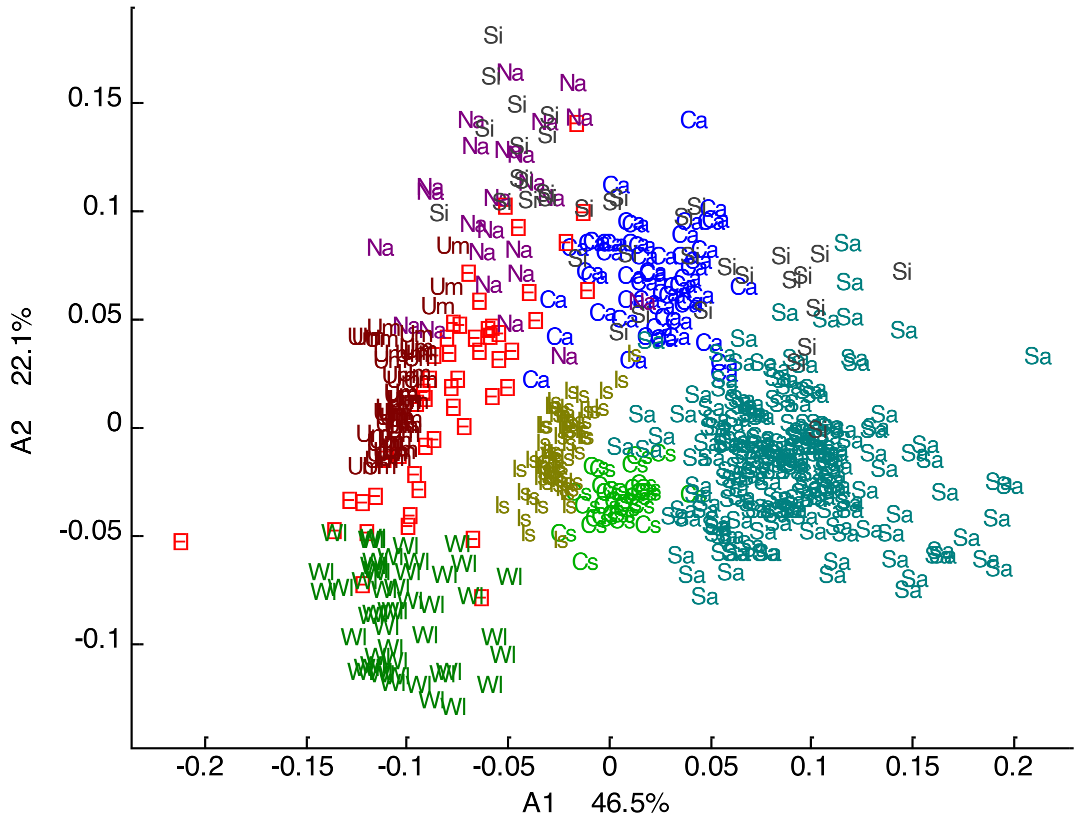
```
map(res.score , 1, 2)
```



This map is not very readable in this format. It is interesting to emphasize the influence of the growing locations. It is very easy to obtain a colored map from the command colored_map1:

```
colored_map1(res.score, 1, 2, 1, 2) ;
```

This command shows that we want to display the scatterplot of the columns #1 and # 2 of the matrix res.score, with colors obtained from the key represented by the two last arguments of the function. These arguments (1 and 2) indicate the location in the character string of each identifier of the codes which determine the coloring. A same color is attributed to all the identical strings.



We can now see clearly that the growing locations have a marked influence on the fatty acid composition of olive oils.

The function colored_map2 does about the same work, but the complete identifiers are displayed.

The correlation circle can be obtained from the function correlation_circle or correlation_plot.

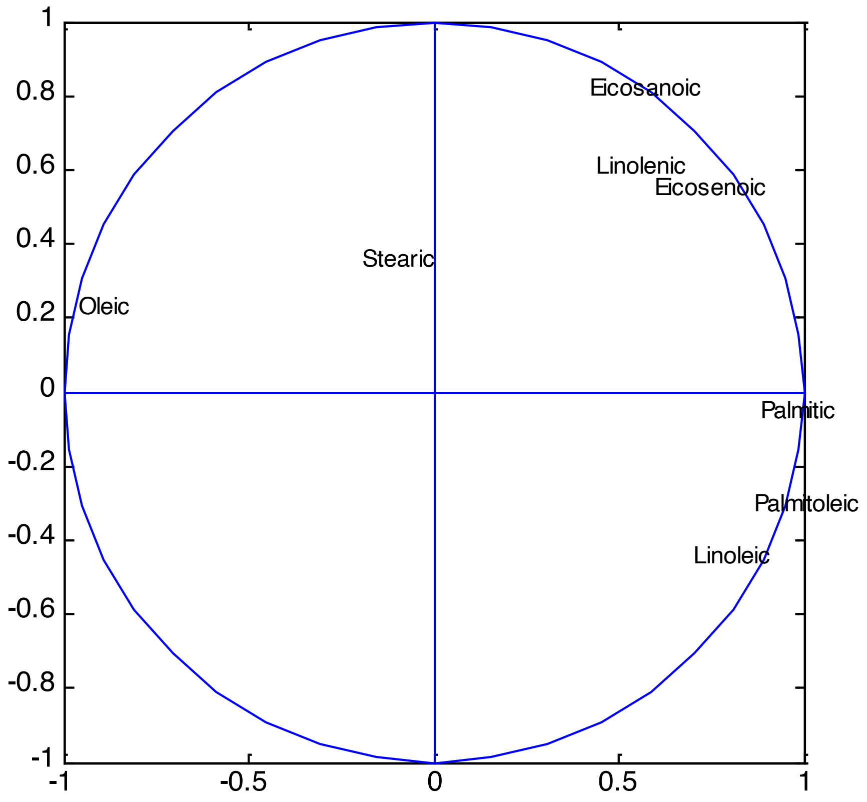
correlation_circle(pcatype,X,col1,col2) ;

The first input argument of this function is a pcatype , which is the output of the function pca. X is the matrix on which the PCA was computed. col1 and col2 gives the numbers of the components to be displayed.

For example, the command:

```
correlation_circle(res,olive2, 1,2) ;
```

gives the correlation circle of the two first components.



Second example: PCA on digitized curves

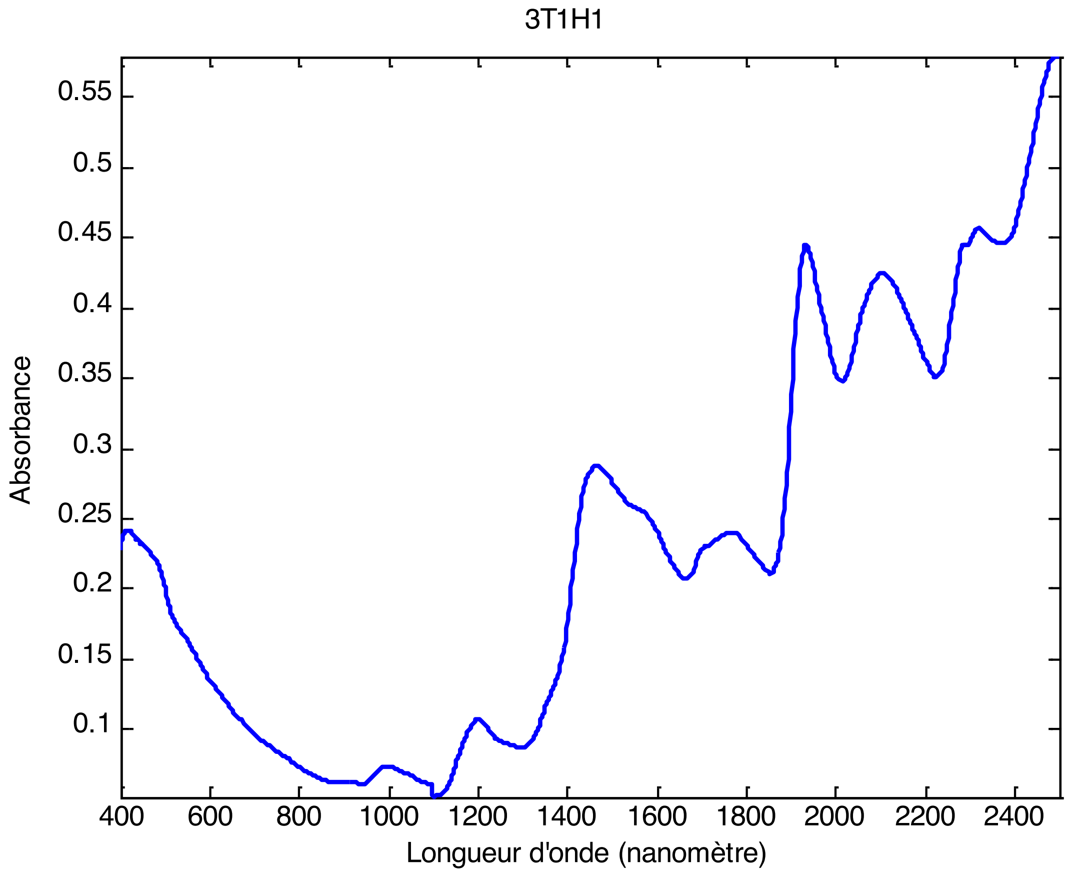
This second example deals with a collection of 140 spectra in the Visible-Near Infrared spectral region. The descriptors of the observations are coded by a character indicating the year of cultivation (3 : 1993 ; 4 : 1994), followed by the code of the nature *durum* (hard (D)) or *tendre* (soft, T) of the studied wheat, the number of the cultivar and the agronomic conditions of cultivation H1, H2, A1, A2.

The spectra were acquired at wavelengths ranging from 400 to 2500 nanometers, at intervals of 2 nanometers. A spectrum is thus represented by a vector of 1050 data points.

The matrix of spectral data is named ble. ("Wheat" in French)

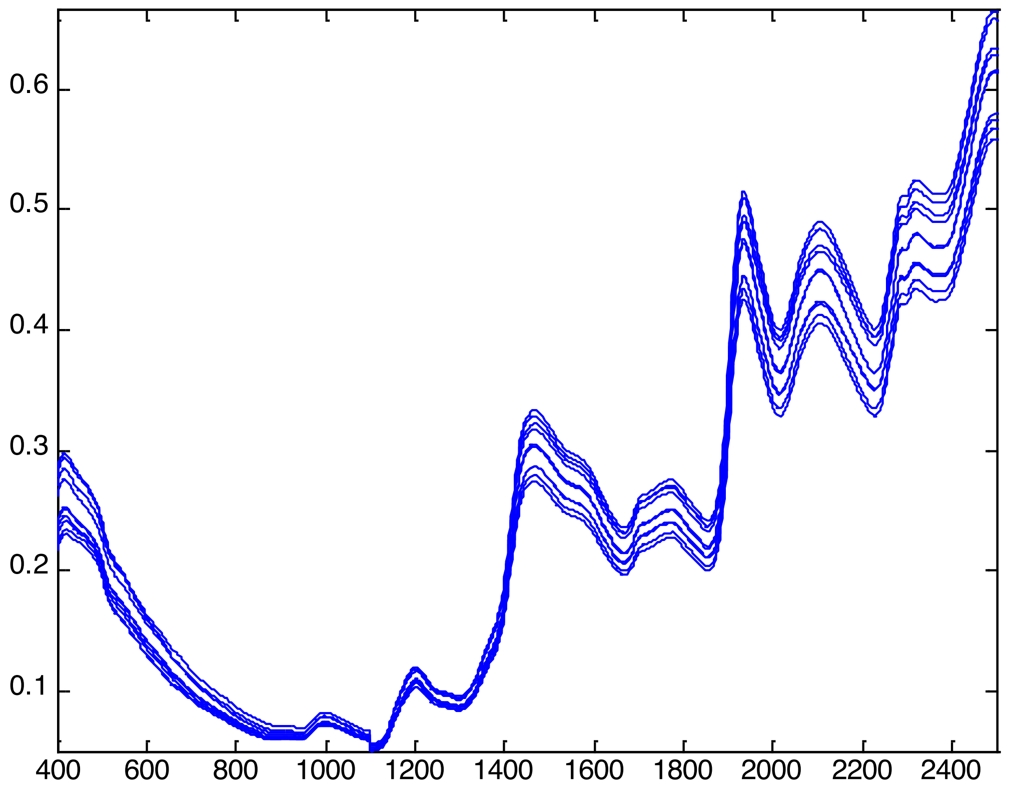
```
ble =
      d: [140x1050 double]
      i: [140x10 char]
      v: [1050x10 char]
```

One can examine the spectra with the function, for example
curve(ble,3,'Wavelength (nanometer)','Absorbance')
displays the third spectrum.



Note that the X-axis is rightly scaled according to the wavelengths. The simple command (MATLAB) plot(ble.d(3,:)) builds up a similar graph, but the X-scale is simply ranging from 1 to 1050 (number of data points).
The function curve interprets the identifiers of the variables as number, in order to have a correct graduation of the X-axis. If this is not possible (the variable identifiers are not numbers), the X-axis is simply graduated using the rank of the variables.

The function curves makes it possible to represent several curves on the same graph. For example, curves(ble,1:10) shows the first 10 spectra.

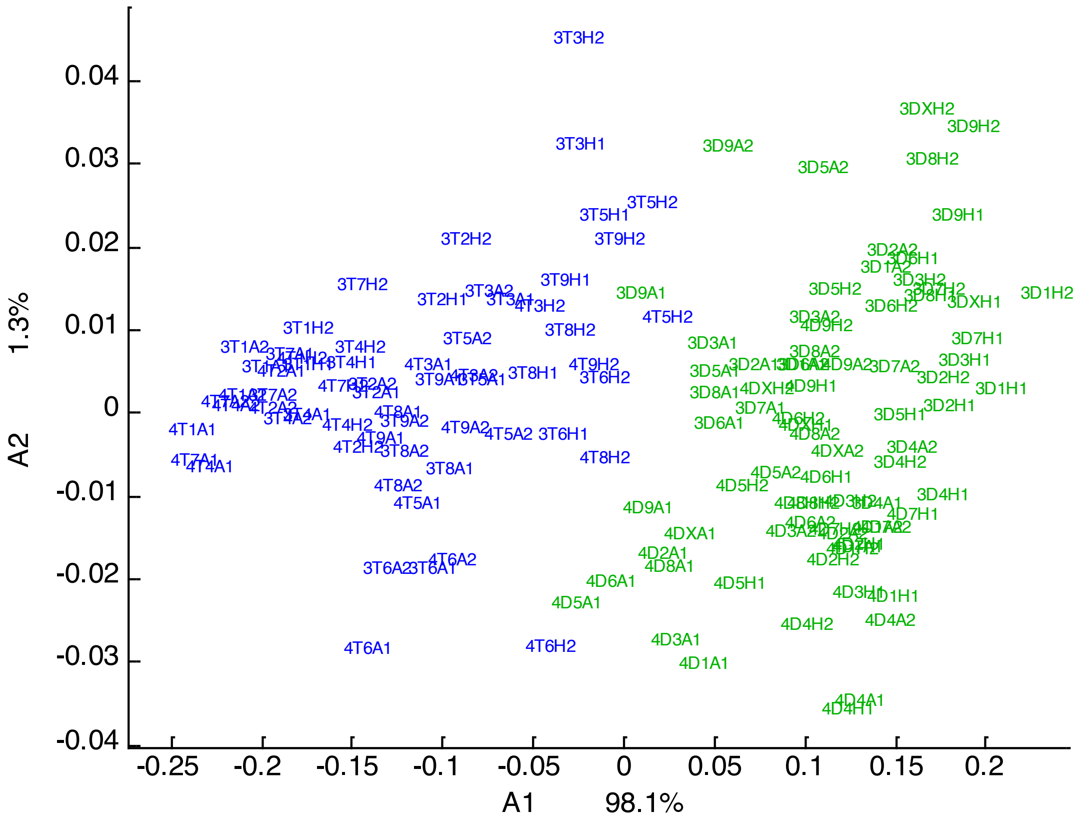


With the function curve, one can use the MATLAB command GINPUT, which gives the X-Y coordinates on the curve. PCA on digitized curves is very close to the one performed on discrete variables. However, it is important to note that, in general, the data matrix must not be standardized.

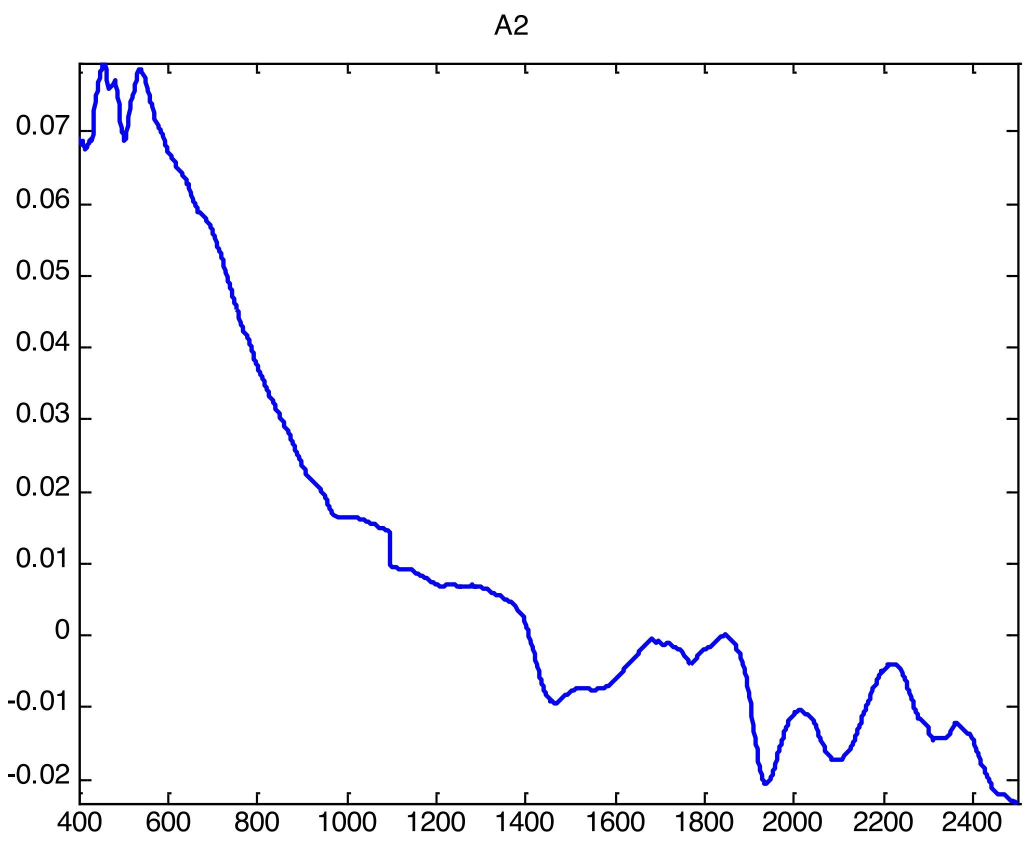
```
acp_ble=pca(ble)

acp_ble =

      score: [1x1 struct]
   eigenvec: [1x1 struct]
   eigenval: [1x1 struct]
    average: [1x1 struct]
```

The function computes all the components (here 140). The factorial maps can be created as previously:

colored_map2(acp_ble.score,1,2,2,2)



The nature *soft/hard* is clearly visible.
When PCA is carried out on digitized signals, it is not very efficient to display the correlation circle. It is wiser to show the eigenvectors as “patterns” or “loadings”. It is easy to show that PCA actually decompose each spectrum as a sum of eigenvectors weighted by the corresponding scores. The eigenvectors (*in columns)* are gathered in the field eigenvec of the structure pcatype.
It is possible to examine the eigenvectors (as profiles) with the function tcurve (curve on transposed matrix). Thus tcurve(acp_ble.eigenvec,2) displays the second eigenvector as a profile.



We can see here that the absorbances in the region 400-800 nanometers (visible light) play an important role in the creation of the second component. According to this component, the wheat samples are mainly separated by their color (in the Visible).

3 Example of basic statistical processing

In order to show the rationale of the functions in SAISIR, some regression and discrimination methods are shown. For the many other functions, see the HTML documentation, the tutorials and the example scripts.

3.1 Regressions

Several regression methods are available in SAISIR: Partial least square (PLS), Principal component regression (PCR), ridge regression, latent root regression and stepwise regression.
Several functions are also available for the validation. Only PLS will be presented here. But have a look on other methods! (Ridge regression is the best in terms of robustness and accuracy).
For building up a regression method, we must have a matrix **X** with *n* rows and *p* columns (predictive variables, and a vector **y** with *n* rows (variable to be predicted) at the SAISIR format. The rows of **X** and y must obviously correspond one with each other (see the function reorder in case of initially scrambled data).

3.1.1 PLS (Partial least square)

In the following, ndim is an integer which indicates the maximal dimension of the model.

BASIC_PLS

Example:

res = basic_pls(X, y, ndim)

Here is the "help" of the function:

```
Input arguments:
 ---------------
 X: SAISIR matrix of predictive variables (n x p)
 y: SAISIR vector of variables to be predicted
 ndim: maximum number of dimension asked

 Output arguments:
 ----------------
 res with fields:
  T: PLS scores (n x ndim)
  P: PLS loadings such as X = TP + residuals (ndim x p)
  beta:  final regression coefficients with ndim dimensions (p x 1)
  beta0: final intercept value (number)
  meanx: mean of X (1 x p)
  meany: mean of y (number)
  predy: predicted y with ndim dimensions (n x 1)
  error: Root mean square error of the model with ndim dimensions (number)
  corcoef: correlation coefficient r between observed and predicted values
  in the model with ndim dimensions
  BETA: regression coefficients for the ndim models (p x ndim)
  BETA0: intercepts of the ndim models (ndim x 1)
  loadings: pls loadings such as T=X*loadings (with X centered) (p x ndim)
  Q: Q value such as y = TQ + residual (ndim x 1)
  PREDY: predicted y values up to ndim dimensions (n x ndim)
  RMSEC: Root mean square error of prediction up to ndim dimensions (1 x
  ndim)
  r2: determination coefficient between observed and predicted values (1 x
  ndim)
```

The field `predy` contains all the predicted values of y for all the models from 1 to ndim dimensions.

APPLYPLS

res=applypls(X,plsmodel, (knowny))

```
Input arguments:
 ---------------
 X : SAISIR matrix (n x p)
 plsmodel  :output  argument  of  functions  "saisirpls",  "basic_pls"  or
"basic_pls2"
 knowny (optional): actual value of y (if it is known, this allows the
 computation of r2 and RMSEV

 Ouput arguments:
 ---------------
 res with fields
 PREDY: predicted values for the all the models given by "plsmodel" (n x
 ndim)
 RMSEV: root mean square error of validation for all the models (only if
 "knowny" is given) (1 x ndim)
 r2: determination coefficient between predicted and observed y values
 (only if "knowny" is given) (1 x ndim)
 T: PLS scores of the set (n x ndim)
```

CROSSVALPLS

crossvalpls - validation of pls with up to ndim dimensions.
 [res]=crossvalpls(X,y,ndim,selected)

```
  Input args:
 ===========
 X: predictive data n x p
 y: Variable to be predicted n x 1
 ndim: maximal number of dimensions in the PLS model
 selected: MATLAB vector n x 1 (0= obs in calibration; 1 in validation)

  Output args
 ============
  res with fields
 ---calibration: calibration results with fields
    BETA: regression coefficients (p x ndim)
    BETA0: intercept (p x 1)
    PREDY: predicted y in calibration (n x ndim)
    T: scores PLS  (n x ndim)
    RMSEC: Root mean square of calibration (1 x ndim)
    r2: determination coefficient (yobs/ypred) (1 x ndim)
 ---validation : validation  results with fields
    PREDY: predicted y in validation (n x ndim)
    RMSEV: Root mean square error of validation (1 x ndim )
    r2: determination coefficient (yobs/ypred) (1 x ndim)
    OBSY: observed y in validation (number of rows=number of obs in
    validation)
```

The input argument selected is a MATLAB *vector* and *not a SAISIR structure*, including n elements taking the value 0 or 1. It indicates how the data in **X** and **y** are separated in a calibration and a validation set. The observations for which the corresponding element of selected is equal to 0 are placed in the calibration set. The other observations are placed in the validation set. The PLS model is established on the calibration set and then applied on the validation set.
obsy and predy are the observed and predicted values in the validation set.

The elements of the vector selected can be created randomly using the function random_select.
selected = random_select(nel, nselect)
nel is the number of elements forming the vector selected (taking the value 0 or1).
nselect is he number of elements set to 1.

3.2 Discriminations

All the discrimination methods require a data matrix **X** and a SAISIR vector indicating the group, called "gr" in the following. If **X** is dimensioned *n x p*, gr is a vector *n x 1*. This vector is formed by integers, taking values ranging from 1 to maxgroup, where maxgroup is the number of qualitative groups. It is necessary that each group be represented at least once in the calibration set. In SAISIR, gr must be a structure. gr.i gives the row identifiers (normally identical to X.i). gr.d is a column-vector indicating the group numbers and gr.v must of course contain a single identifier, such as « group ». Only PLS discriminant analysis is presented here, but there are many powerful functions for discrimination in SAISIR. Look for example at the" MAHA " series or at QUADDIS (quadratic discriminant analysis) which is very powerful.

CREATE_GROUP1

If the row descriptors are keys able to indicate the group, it is very easy to create the vector gr. For example, in the previous example related to olive oil the growing locations are indicated by the two first letters of the names.
The function create_group1 can be applied.

```
gr=create_group1(oil1,1,2);

>>gr =
>>    d: [572x1 double]
>>    i: [572x20 char]
>>    v: 'group'
>>    g: [1x1 struct]
```

The groups are built by exploiting oil.i and researching the character chains starting in position 1 and finishing in position 2. The observations having the same extracted chains are placed in the same group.
The structure gr.g gives the number of observations in each group and the corresponding letters.

gr.g.i	*gr.g.d*
Ca	56
Cs	33
El	50
Is	65
Na	25
Sa	206
Si	36
Um	51
Wl	50

Thus, for example, 36 observations belong to the 7th group, the growing location Si.

PLSDA, CROSSPLSDA

PLSDA basically consists in replacing the vector of group by the matrix of indicators, and then applying PLS2 on this indicator matrix.
Suppose that the group numbers of the 3 first observations are "2", "1" and "4" and the number of group is 5.
The beginning of the indicator matrix is:
0 1 0 0 0
1 0 0 0 0
0 0 0 1 0
PLS2 is able to predict several y-variables, which are here the indicators. PLS2 gives us a matrix **T** of PLS scores and a matrix **ypred** of predicted indicators. Two strategies can be used to predict the groups from **T** or **ypred**. The first one consists in putting the observation in the group for which the indicator is the largest. For example, if an observation gives predicted indicators equal to [-0.1 1.5 0.5 0.2 0.01] then the predicted group of this observation is "2" .
The second strategy (which is probably the better) is to use the PLS scores **T** as the input of an ordinary linear discriminant analysis. These two strategies are proposed in the function PLSDA.

function[type]=plsda(x, group, ndim)

Discrimination using PLS2, with ndim dimensions.

```
Input arguments:
================
 X : SAISIR matrix of predictive variables (n x p)
 group: SAISIR (n x 1)  vector of integers of groups. Observations with
 the  same  number  in  group  belong  to  the  same  group  (missing  group  not
allowed)
 ndim: number of dimensions in the PLS model

 Output arguments
 ================
 plsdatype with fields
 beta       :coeff for predicting the indicator matrix
 beta0            :intercept for predicting the indicator matrix
 t                      :PLS latent variable
 predy            :predicted indicator matrix
 classed          :predicted groups according to method #0
 ncorrect   :number of rightly classified samples according to method #0
 (attribution to index of max of predicted Y)
 confusion  :confusion matrix according to method #0
 ncorrect1  :number of rightly classified samples according to method #1
 (mahalanobis distance on latent variable t)
 confusion1 :confusion matrix according to method #1
 tbeta            :coeff for predicting the latent variables t
 tbeta0           :intercept for predicting the indicator matrix
 linear           :linear form for direct prediction of group
 linear0    :%the min of x'*linear + linear0 gives the predicted group
 (this is equivalent with considering Mahalanobis distances)
```

Example of use:

```
>>pl=plsda(oil1,gr,8)
>>pl =
>>         beta: [1x1 struct]
>>        beta0: [1x1 struct]
>>            t: [1x1 struct]
>>        predy: [1x1 struct]
>>      classed: [1x1 struct]
>>     ncorrect: 478
>>    confusion: [1x1 struct]
>>     classed1: [1x1 struct]
>>    ncorrect1: 533
>>   confusion1: [1x1 struct]
>>        tbeta: [1x1 struct]
>>       tbeta0: [1x1 struct]
>>       linear: [1x1 struct]
>>      linear0: [1x1 struct]
>>         info: [1x76 char]
```

When using the classification rule #1 (Mahalanobis distance on PLS scores), an unknown observation can be quickly classified by computing the vector

test = x'*linear + linear0 .

This vector has as many elements as the number of groups. The classification rule says that the observation must be classified in the group for which the element of test is the minimum.

The function crossplsda allows the validation of the model. For example: res=crossplsda(oil1,gr,5,selected). The MATLAB vector selected is defined as previously.

```
res =

      confusion1: [1x1 struct]
      ncorrect1: 328
      nscorrect1: 171
      sconfusion1: [9x9 double]
      confusion: [1x1 struct]
      ncorrect: 289
      nscorrect: 150
      sconfusion: [9x9 double]
      info: [1x76 char]
```

The letter "s" (for example in nscorrect1) indicates that it is the validation (supplementary) set.

4 Recommendations to the programmer

Dear chemometrician, we hope that you will have the ambition to improve the SAISIR environment! If it is the case, please send your proposal contribution to myself (Dr Dominique Bertrand (bertrand@nantes.inra.fr). Anyway, even if you write functions for your own use, you will spare a lot of time by respecting some rules.

Here are some simple recommendations which, if respected, will make it possible to enrich the environment and facilitate the implementation of new functions.
The SAISIR environment does not include script, but only functions such as
function [result, result1]=name(param1,param2, ...)
In general one must not mix graphical and computational works in a single function.

Name of the functions
The functions must be written in US-English (for example"color, "analyze" and not "colour", analyse)
The name of the function may respect, if possible the logic of the MATLAB function, with a systematic use of the usual lexemes (for example: str pour *string*, num for *number*, nan pour *not a number* etc.).

Input arguments
The input arguments must be given according to their importance, with first the matrices then the other parameters. The input matrices must be at the SAISIR format with the three fields .d, .i, .v .

The optional parameters must be given at last.

Help
The first lines of the function are remarks which can be consulted through the command help <*function name*>. With MATLAB, the command help <*directory*>) displays the first line of these remarks. It is thus important that this first (short) line was meaningful.
The second line is a remark copying the header of the function. This helps the user to correctly fill the input arguments.
It must be also noticed that the SAISIR environment contains a way to automatically create HTML documentation from a list of function. See the function build_documentation for more information on this possibility. The function build_documentation works properly only if the help remarks are correctly written.

Here is an example of a correct help header:

```
function res=myfunction(arg1, arg2)
%myfunction  - does something very smart
%function res=myfunction(arg1, (arg2))
%
%Input arguments
%============
%arg1       : its role
%arg2       : (optional) its role. (default: <value>)
%
%Output argument
%============
%res with fields
%     struct1: explanation
%     struct2: explanation
…
```

If some input arguments are optional, they are indicated in bracket.

In the more recent MATLAB versions, the name of the function must be identical to the name of the « .m » file, respecting the letter case. For example a function demo, will be actually recognized with a file DeMo.m but will causes the display of the warning « `inexact Matching` ». For this reason, we will recommend that every function name is in low case, with an exact matching with the ".m" file.
Moreover, the first line must be written exactly in the following way:
%demo[space][space]-[space]demonstration
The chain <function name> [space(s)], [hyphen], [space], text activate in MATLAB the hypertext interactive link.

Output arguments
When the outputs arguments are vectors or matrices, they must compulsorily be in the SAISIR format. The correct order is .d, .i, .v with, in principle, no other fields. That means that somewhere in the source there are lines such as res.d= …, res.i=…, res.v=…
It is useful to respect this order, because MATLAB allows the creation of vectors of SAISIR structure only in this conditions (such vectors are the input of function dealing with multiblock tables such as statis or comdim.)
When a function returns several pieces of information, it is a good idea to gather them in a few numbers of structures (in general, only one!) in which each field is a matrix at the SAISIR format.

For example, the function PCA (principal component analysis) has only one output argument.

```
p=pca(x) ;
p =

       score: [1x1 struct]
    eigenvec: [1x1 struct]
    eigenval: [1x1 struct]
     average: [1x1 struct]
```

Each field is a SAISIR structure. When the matrices are created by the function (like the scores in PCA) the programmer must create correct identifiers for the rows or the columns. For example:

```
>>p.score.v

A1    81.5%
A2    16.9%
A3     0.6%
A4     0.3%
A5     0.2%

>>size(p.score.v)
5 11
```

In this example, p.score.v is a matrix of characters.

Transmission of developed functions and report of bugs

The interesting developed function can be sent by e-mail to bertrand@nantes.inra.fr.
It is useful that the function includes a remark giving the coordinates of the author and possibly the related scientific publications. *People making this beneficial work of course admit that this function will be freely distributed, without any remuneration of any kind. In scientific papers, the users are asked to quote the authors of the functions they have used.*

Some lexemes of the functions in SAISIR

(To be used in the input arguments)
2: 'to' indicates a transformation (for example num2str)
apply: apply a model build elsewhere. (For example pca, applypca). The output of the building function is often called a "type" (for example pcatype)
append: concatenate
bag: SAISIR structure in which the field '.d' is made of a matrix of char
col: number (rank) of a column or collection of tables (vector of SAISIR structure)
delete: suppress
dim: dimension
group: qualitative groups (in general: SAISIR vector of integers)
load: load
map: plot with the name displayed
range: MATLAB vector of value giving a choice of numbers
read: read file
row: row of a table
startpos endpos : start and end of a zone to be selected (generally in strings)
saisir : SAISIR matrix
sel: MATLAB vector the elements of which are 0 or 1
x: SAISIR matrix
y: SAISIR vector or matrix of values to be predicted (normally in regression methods)