FACT
A CHEMOMETRIC TOOLBOX
UNDER SCILAB ®

GETTING STARTED

For FACT_0.3 and FACT 0.4

Updated : 2014, May, the 13th

For users

- It is strongly advised to read this document before using the Fact functions, it is not very long!

• *The use of Fact implies to know a bit about* Scilab.

• *Almost all functions are associated to an help using :*

help <fonction>

which opens a html window with a browser.
To get a list of the main functions :

help abc_fact
help fact

• *If the result yielded by a function is not clear, it is possible to type the name of the output argument, then check its fields. For example, a principal component analysis is obtained by :*

```
-->p=pcana(ble) ;
```

But what is there into p *?*

```
-->p
  scores: [1x1 struct]
  eigenvec: [1x1 struct]
  var_scores: [1x1 struct]
  eigenval: [1x1 struct]
  x_mean: [1x1 struct]
  centred: 0
  std : 0
```

Then we can guess that p.scores *contains the scores,* p.eigenvec *the eigenvectors, and so on.*

How to install/remove Fact for Scilab :

Scilab is installed from: www.scilab.org

1- open Scilab, choose the console, open « Applications » then « Atoms modules »
sort them by alphabetical with « all modules » ; choose Fact
click on « install » :
1bis : alternatively : open the Fact page (http://atoms.scilab.org/toolboxes/FACT)
download the binary file (ex : FACT_0.4-3.bin.zip) in a directory
open the same directory with Scialb browser
verify the internet connection !
type in the console: atomsInstall('FACT_0.4-3.bin.zip')

2- restart Scilab *Fact has been installed*

- remove Fact with the « remove» button

FACT

A CHEMOMETRIC TOOLBOX
UNDER SCILAB®

GETTING STARTED

Contents

Introduction :

Issued for example from chromatography, spectrometry or images, data suited for chemometric applications have very often hundreds or thousands of lines and/or columns, and thus are very difficult to handle with the usual softwares, Excell or OpenOffice. To fix this problem, we have developed toolboxes under Scilab et Matlab, easing the work of users and giving a possibility for the chemometric community to contribute. Only the Scilab version is presented here, for Matlab see at the last page of this document.

A first toolbox developed by Dominique Bertrand (INRA) was called « SAISIR » (*Statistiques Appliquées à l'Exploitation des Spectres Infrarouge*). It contains tools for loading, handling and saving data, graphical interfaces and statistical applications: multivariate analysis, discrimination, regression, …Saisir has been built since 1998 for Matlab. One of its particular features is to use a Div structure introduced before 1985 in BASIC in order to keep informations about the observations and the variables along with the data.

The rewriting of Saisir for Scilab led to update the functions, some added, other removed or replaced. These modifications were important enough for the resulting toolbox to be considered different from Saisir. In order to avoid confusion, the name was changed in : « FACT » (*Free-Access Chemometric Toolbox*). Fact keeps many functions as well as the spirit of Saisir. The use of Fact implies a basic knowledge of Scilab environment and of the chemometric tools that are used.

1. General principles

Both Saisir and Fact use structures. The basic principle is that the lines and columns contain identifiers or labels which « follow » the processings. For example, the following table contains notes of three apple cultivars named « GALA1 », « FUJI1 », « FUJI2 », for three sensorial descriptors: global « OGLO », earth «OTER », cellar « OCAV » :

	OGLO	OTER	OCAV
GALA1	2.8	1.2	0.3
FUJI1	2.6	0.5	0.4
FUJI2	7.5	0.3	0

(in this example the decimal separator is a dot)

The data contains 3 rows and 3 columns. The rows (or *observations*) are identified by the names of the cultivars : « GALA1 », « FUJI1 » and « FUJI2 ») ; as well as the columns (or *variables*) are identified by « OGLO », « OTER » et « OCAV ». Then the data form the matrix:

```
2.8  1.2  0.3
2.6  0.5  0.4
7.5  0.3  0
```

This table will be processed with the three informations. The corresponding Fact format is a *DIV structure* which contains the fields :

<> .d for « data/données »
<> .i for « individuals/individus » (rows) ;
<> .v for « variables » (columns).

The Div structure is necessarily built by the function div or other functions into which this function is embedded, according to two possibilities :

a) build manually a structure with the fields .d,.i and .v, then apply div :

```
-->Pomme.d=[2.8 1.2 0.3;2.6 0.5 0.4;7.5 0.3 0];
-->Pomme.i=['gala1';'Fuji1';'Fuji2'];
-->Pomme.v=['oglo';'oter';'ocav'];
-->Pomme
 Pomme  =
   d: [3x3 constant]
   i: [3x1 string]
   v: [3x1 string]
-->Pomme=div(Pomme)
 Pomme  =
   d: [3x3 constant]
   i: [3x1 string]
   v: [3x1 string]
```

b) apply div by giving the three fields .d, .i and .v in this order:

```
-->Pomme=div([2.8 1.2 0.3;2.6 0.5 0.4;7.5 0.3 0],['Gala1';'Fuji1';'Fuji2'],
['oglo';'oter';'ocav'])
 Pomme  =
  d: [3x3 constant]
   i: [3x1 string]
   v: [3x1 string]
```

The Div structure Div is identified by the command typeof :

```
-->typeof(Pomme)
 ans  =
 div
```

On the screen, the baselinestrech between the fields .d, .v and .i is simple for usual structures, double for Div structures.

Each field can be extracted, e.g.:

```
--> Pomme.i
ans=
! Gala1           !
! Fuji1           !
! Fuji2           !

--> x=Pomme.d
x =
        2.80        1.20        0.30
        2.60        0.50        0.40
        7.50        0.30          0
```

Note that the fields « i » and « v » are *strings ;* Scilab does not use *cell arrays* as Matlab.

An interesting case concerns data represented by curves (spectra, chromatograms,..). They form matrices with *rows* as observations. The identifiers of the variables are thus numbers converted inti strings and representing the scale marks of the curve. Here is an example from infrared spectra :

	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 » identify the observations ; and « 1100 », « 1102 », « 1104 », « 1106 », « 1108 » identify the variables. Suppose that the Div structure of these data is called spectra, then:

```
-->spectra
spectra=
d: [4x5 constant]
i:  [4x1 string]
v: [5x1 string]

--> spectra.v
ans=
! 1100          !
! 1102          !
! 1104          !
! 1106          !
! 1108          !
```

The numerical origin of the variable labels is used for the representation of the curves (functions curves and tcurves).

2. Using the Fact environment

The help command opens a html window explaining the syntax of the function. The list of the Fact commands classified by thematic is obtained by :

```
-->help abc_fact
```

It allows to find quickly the searched command. In the upper left hand side of the window, is printed:

fact >> fact > abc_fact

Click onto one of the fact and it opens an alphabetical list of the functions. Click onto the desired function to get all the details.

2.1 Getting started

The data compatible with Fact form matrices. Missing values (NaN) are not handled and generate an error message.

2.1.1 Loading, building and saving Div structures

Loading from Excel or OpenOffice
The function csv2div imports data from Excel and OpenOffice if they are organized as follow :
<>the first row contains the identifiers of the columns;
<>the first column contains the identifiers of the rows;
<>the other cells contain *numerical values*, with *a dot « . » or a comma « , »* as decimal separator.
The cell in position (1,1) is dropped during the importation process.
Example of the data « fruits » with Excel or OpenOffice:

	OGLO	OTER	OCAV	
GALA1		2,8	1,2	0,3
FUJI1		2,6	0,5	0,4
FUJI2		7,5	0,3	0

This file must be saved under the « .csv » format:
<> with Excel, click onto save under then CSV (separator = « ; ») ;
<> with OpenOffice : click onto save under then csv / edit the parameters of the filter then filed separator: « ; » and clear the default text separator.

The loading into Scilab is obtained by the command: res = csv2div('filename') ;
where filename is a string, the name of the file to be loaded (and eventually with the path), and res is the Div structure. For example :

```
--> essai=csv2div('fruits.csv')
```

loads the file fruits.csv and puts the result into the Div structure essai.

The csv2div function imports the missing values replaced by NaN in the original file .csv. But if missing values are represented by an empty field, then an error message is generated by csv2div.

Export of Div structures towards a .csv file (for re-use with Excel or OpenOffice) :
The command div2csv exports a Div structure into a .csv file, with the identifiers of the rows and the columns. The file is saved in the .csv format (field separator: ";") with its row and column identifiers. For example:

```
-->div2csv(essai, 'tableau' , ',' )
```

saves the Div structure essai into the file TABLEAU.csv. Note that the last argument is the decimal separator. Csv files are easily imported by Excel or OpenOffice.

2.1.2 Handling the data

Simple operations are possible with Div structures because the basic functions have been overloaded for Div; they are summerized into the following table. Of course the dimensions must fit.

Operation	Scalars	Matrices	Div Structures	Syntax	Detail of the calculation
Transposition			a,c	c=a'	c.d=a.d' c.i=a.i c.v=a.v
Addition			a,b,c	c=a+b	c.d=a.d+b.d c.i=a.i c.v=a.v
Subtraction			a,b,c	c=a-b	c.d=a.d-b.d c.i=a.i c.v=a.v
Multiplication by a scalar	s		a,c	c=s*a	c.d=n*a.d c.i=a.i c.v=a.v
Division by a scalar	s		a,c	c=a/s	c.d=a.d/n c.i=a.i c.v=a.v
Multiplication of Div structures			a,b,c	c=a*b	c.d=(a.d)*(b.d) c.i=a.i c.v=a.v
Element-wise multiplication			a,b,c	c=a.*b	c.d=(a.d).*(b.d) c.i=a.i c.v=b.v
Element-wise division			a,b,c	c=a./b	c.d=a.d./b.d c.i=a.i c.v=a.v
Merging the row			a,b,c	c=[a;b]	c.d=[a.d;b.d] c.i=[a.i;b.i] ; c.v=a.v
Merging the columns			a,b,c	c=[a b] ou c=[a,b]	c.d=[a.d b.d] c.i=a.i c.v=[a.v;b.v]
Extracting data	p,q,r,s		a,c	c=a(p :q,r:s)	c.d=a.d(p:q,r:s) c.i=a.i(p:q) c.v=a.v(r:s)
Insertion of data	p,q,r,s	m	c	c(p:q,r:s)=m	c.d(p:q,r:s)=m

Tableau 1: operations made possible for Div structures with the current operators (' + - * / .* ./ [ ] [;])

For the extraction or the insertion of data, the ranges p:q or r:s cannot be replaced by p or r. Thus c=a(p,r) is valid but not c=a(p) even in the case of vectors.

The following commands apply onto the indices and not onto the descriptors. So, the row corresponding to Fuji2 is selected by:

```
--> Fuji2=spectre(3,:) ;   // and not: Fuji2=spectre(Fuji2,:)
```

It is sometimes boring with very big matrices to find the index of an observation or a variable from its identifier. The command strseek can help:

```
--> index = strseek (Pomme.i, 'Fuji')
index=
2.
3.
```

It returns the indexes of all the observations of Pomme.i which contain the string « FUJI » within their names. Uppercase and lowercase characters are considered as different.

With numerical data, the command indexseek returns a single value, the index whose value is the closest of a given value. For example:

```
-->index=indexseek(spectre.v,1104);
```

Note that an exact fit is not necessary; this command will find the appropriate index even if the variable « 1104 » is coded by « 1103.9996 ».

Using identifiers as an extraction key

It is ofen useful and sometimes mandatory for big matrices to use the names as extraction key. Numerous procedures (discriminant analysis, principal component analysis, analysis of variance, graphics) are simplified if the user abided to this principle. Let explain it with an example.

Suppose that an experiment involves three cultivars of wheat flour (*Camp Rémi*, *Talent* and *Arminda*), cultivated in two location (*Paris* and *Montpellier*), with 20 répétitions. A correct identifier for an observation could be: CRPA09 which means : *Camp Rémi*, cultivated in *Paris*, 9th répétition. Two letters are used for the cultivr (CR), two letters for the location (PA), and *two* letters for the repetition: ('09' and not '9') which adress to the 9th observation between 1 and 99 répétitions !

Similarily TAMO19 means *Talent,* cultivated in *Montpellier*, 19th repetition.

The dimensions of the identifiers must remain constant. For instance CRMO12 and ARMPA10 are not compatible because the CR code contains 2 characters while the ARM code contains 3 characters! Also the same combinations of letters cannot be used for several codes: si PA is used for the location, it cannot be remployed for a wheat cultivar.

Such identifiers can be used to extract data. The extraction from the Div structure wheat of all the flours from wheats cultivated in Paris is obtained with the command:

```
[indexofobs] = strseek(wheat.i,'PA')
[sel_obs] = wheat(indexofobs,:)
```

The first line determines the indexes of the observations containing PA, the second line extracts them and builds the corresponding Div structure.

2.1.3 Principal component analysis

The principal component analysis is useful for a global observation of the data, before other processings. Moreover this method presents several of the elements and the graphics used in the Div environment.

A first example concerns the data *Olive oil* 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).

For a characterization of the olive oils from several regions of Italy, the authors have quantified 8 fatty acids (*Palmitic*, *Palmitoleic*, *Stearic*, *Oleic*, *Linoleic*, *Eicosanoic*, *Linolenic*, *Eicosenoic*) into 572 samples of olive oils issued from 9 Italian regions. The file olives.csv contains the data. The rows correspond to the 572 obervations. The columns correspond to the concentrations of each of the 8 fat acids. Thus we get a matrix of dimensions (572 x 8).

The identifiers of the observations contain 2 characters for the region. For instance Ca005 means that the 5th sample was obtained in the region Ca for Calabria.

The data is loaded into Scilab :

```
--> olive1=csv2div('olives.csv')
olive1=
   d: [572x8 constant]
   v: [8x20 string]
   i: [572x20 string]
```

The PCA is obtained with the commands pcana or cspcana onto a dataset *without missing or NaN values, containing only the row and columns to be processed.*
The difference between pcana and cspcana is that cspcana always centers and standardizes the data, while by default pcana does not center nor standardize. Nevertheless centering and standardization can be obtained manually with :
centering: centering the columns
standardize: normalisation or standardiaation of the columns (variance=1)
So the following options are equivalent:

option 1 :

```
--> olive2=centering(olive1);
--> olive3=standardize(olive2);
→ res=pcana(olive3)
 res  =
   scores: div
   var_scores: div
   eigenvec: div
   eigenval: div
   ev_pcent : div
   x_mean: div
   x_stdev: div
   centred: 0
   std : 0
```

option2 :

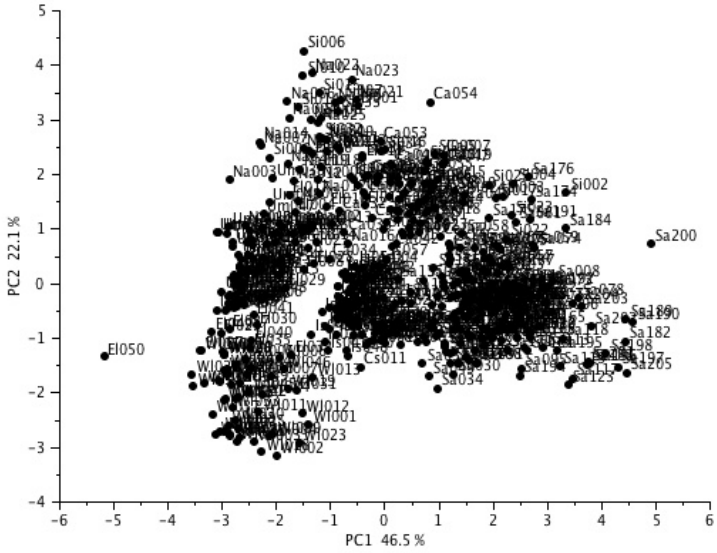
```
-->res=cspcana(olive1)
 res  =
   scores: div
   var_scores: div
   eigenvec: div
   eigenval: div
   ev_pcent : div
   x_mean: div
   x_stdev: div
   centred: 1
   std: 1
```

The output are the same, except for the options centred and std. They report the pretreatments processed by the functions pcana or cspcana, not the pretreatments processed before.
In those examples, res is a structure containing all the results of the PCA. The fields scores, eigenvec, var_scores, eigenval et x_mean are Div structures.
A useful field is res.scores. Each row represents an observation and each column an eigenvector ranked in the decreasing order of the eigenvalues. The identifiers of the rows of res.scores are thus a copy of those of olive1.i or olive3. The identifiers of the columns of res.scores are built by the functions pcana or cspcana :

```
-->res.scores.v
 ans  =
!PC1  46.5 %  !
!PC2  22.1 %  !
!PC3  12.7 %  !
!PC4  9.9 %   !
!PC5  4.2 %   !
!PC6  3.1 %   !
!PC7  1.5 %   !
!PC8  0 %     !
```

The *factorial maps* represent pairs of columns chosen by the operator into res.scores as graphics X-Y. The command map and related commands are used.

```
--> map(res.scores, 1, 2)
```

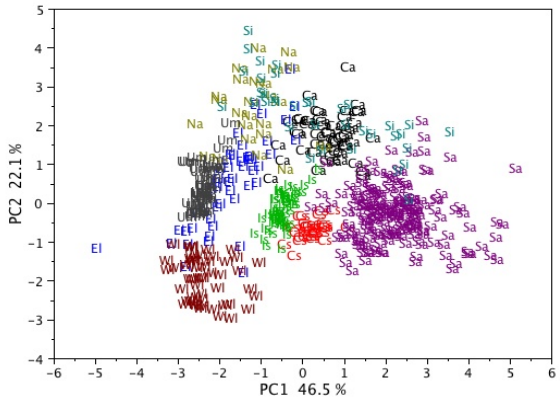


The previous figure represents the columns 1 and 2 of res.scores. More information is added to the captions of the abscissa and ordinate: the eigenvalues of axes 1 and 2, in percent.

This figure is difficult to read. And we can wonder about the influence of the origin of the oils. A colored map is easily obtained with coloredmap.
For example :

--> coloredmap(res.scores, 1, 2, 1, 2)

asks for a representation of the columns 1 and 2 (arguments 2 and 3) of the matrix res.scores using a color according to the key given by the arguments 4 and 5. These arguments determine the position (start/stop) of the selected region within res.scores.i which will be used to gather observations and to represent them with the same color. In our example, the function extracts the strings of res.scores.i which begin in the first position and end in the second position : Ca, Cs, El, Is, Na, Sa, Si, Um and Wl, and sets a color for each group. Each observation is represented by these colored strings.



It is clear that the regions have an influence onto the composition in fatty acids of the olive oils.The eigenvalues in percent, '46,5%' and '22,1%' are added automatically to PC1 and PC2 respectively.

Other options are possible to represent the classes:

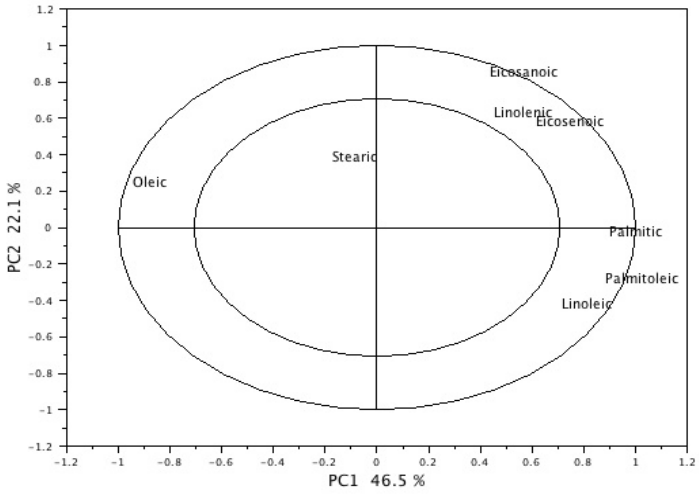
each class with a different color:
scmap : a symbol (<=7 classes) or a number (>7 classes)
diacmap : 'Δ'
dotcmap : '*'
starcmap : '*'

all classes in black and white :
kcmap : the identifier of a class
kscmap : a symbol (<=7 classes) or a number (>7 classes)

The correlation map is obtained with the command corrplot according to the syntax : corrplot(scores,col1,col2, fact1,fact2, …). The first argument is a matrix whose columns are orthogonal together, for example the score of a PCA or a PLS; col1 and col2 are the rank of the components from score to be plotted; the last arguments are Div structures for which we wish to visualize the correlation with the scores. For example:

--> corrplot(res.scores,1,2,olive1) ;

represents the correlation of the variables of olive1with the factors 1 and 2.
Supplementary variables (not involved in the process of calculation of the scores) can be added as supplementary arguments.



Second exemple : PCA onto spectra
This second example is based on a collection of 140 visible and infrared spectra of wheat flours. Observations are described by a code which gives successively the year of harvest (3 : 1993 ; 4 : 1994 ; the type durum (D) or soft (T) of the wheat, the number of the cultivar and the agronomical conditions H1, H2, A1, A2. Spectra have been recorded between 400 and

2500 nanometers by 2 nanometers, yielding 1050 spectral variables for each spectra. The Div structure is the field nir.x from the file nir.dat:

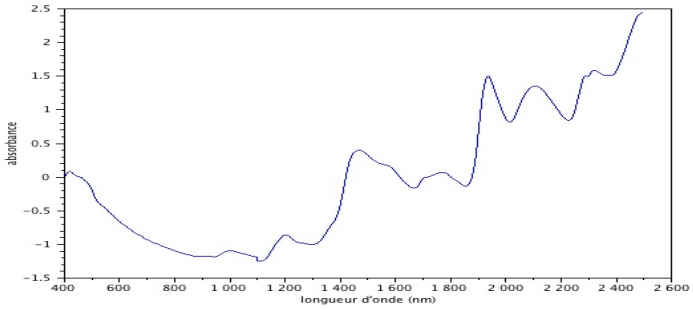
```
--> nir.x
ans =
      d: [140x1050 constant]
      i: [140x1 string]
      v: [1050x1 string]
```

Spectra can be plotted with the command curves which was designed for column vectors. For row spectra, a transposition is necessary. For exemple :

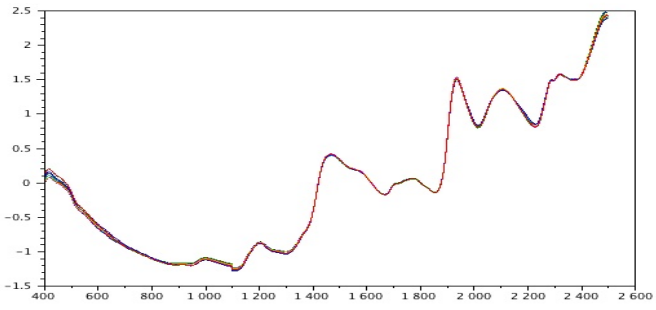
```
--> curves(nir.x(3,:),'','Longueur d''onde (nanomètre)','Absorbance')
```

represents the 3rd spectra. Note that nir.x(3,:) is a vector, so it is represneted by a column; for nir.x(3:4,:) which is a matrix the spectra form the rows and the transposed form is used:

```
--> curves(nir.x(3:4,:)','','Longueur d''onde (nanomètre)','Absorbance')
```



The second argument allows to choose how the curves are represented. It was set here to the default value '' but it is also possible to choose the color or the style of the curve, e.g. 'r*' represents the observations as red stars.
The ticks of the abscissa correspond to the values of the wavelengths. The Scilab command plot(nir.x.d(3,:)') yields almost the same figure, but the abscissa ticks are in the range 1 to 1050 , the number of variables. And the captions are not printed directly, contrary to curves which tries to interpret the variables labels as numbers, in order to correctly set the abscissa ticks. And when it is not possible, e.g. the variable labels are not numbers, the X scale is graduated according to the rank of the variables.
Several curves can be represented in the same figure with curves. For instance, the command curves(nir.x(1:10,:)') represents the 10 first spectra, overlapping.



and curves(nir.x ([3;6;9] , :)') represents the spectra of the rows 3, 6 and 9.

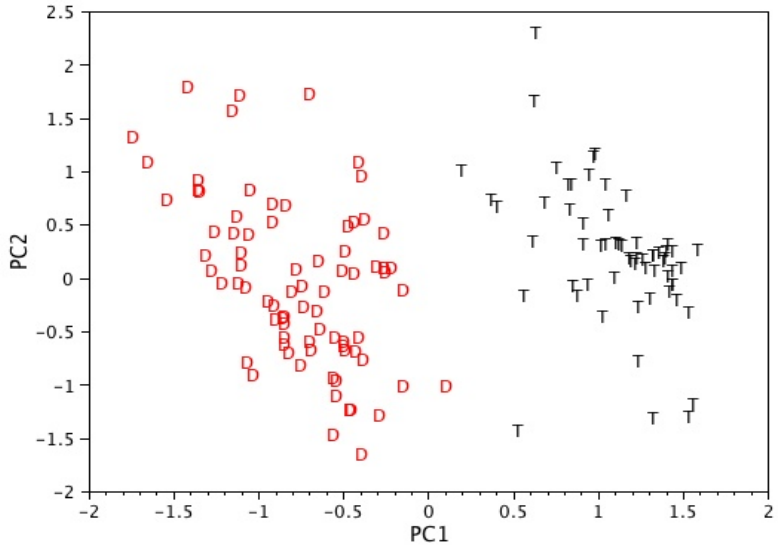
The PCA onto spectra is similar to PCA obtained with other data. Nevertheless the intensity of each wavelength is meaningful and thus the data should never be normalized. But centering remains possible, it can be obtained with the centering and pcana commands:

```
--> x2=centering(nir.x);
--> resacp=pcana(x2);
```

All the possible components are calculated, here from 1 to 140. The factorial maps are obtained as previously;

```
--> coloredmap(resacp.scores,1,2,2,2)
```

represents the factorial plan determined by the two first eigenvectors, using the characters in second position within resacp.scores.i (start and stop at 2) to build the map. The nature durum/*soft* of the wheats is represented by the letters **T** and **D** respectively; their influence is very clear.



3 Supervised multivariate analysis.

Regression and discrimination methods are presented in 3.1 and 3.2 respectively.

3.1. Regressions

The regression methods.

Several regression methods are available : partial least square (PLSR), principal component regression (PCR), Ridge regression, multiple linear regression (MLR).
The simplest regression is the MLR, also named least squares. Unfortunately it cannot apply to highly correlated variables, so these variables are replaced by orthogonal scores in PLSR and PCR.

The data for building a regression.
To build a regression, it is necessary to have a matrix **X** of *n* rows and *q* columns, and a vector **y** of *n* rows at the Div format; **y** is predicted using **X** . The row in **X** and **y** must correspond to the same observations (if necessary, see the reorder command to achieve it). The regression methods in Fact only allow the prediction of a single variable at once, that is why **y** is always a vector.
The PLS regression is detailed. The other methods apply the same way.

The partial least squares regression or PLSR
The PLS regression is a very famous and powerful regression method. It applies even for highly colinear variables in **X**. The « complexity » of the model depends on a parameter called « dimension ». The more a model is complex, the more it fits but the less it is stable. So a compromise between stability and complexity is necessary. The choice of the number of dimensions is possible using cross-validation. Hereafter, ndim is the number of dimensions of the model.
Back to the previous example, about wheat. The reference values of the protein concentration in nir.y correspond to the spectra. The first step is to build a model, the second step is to apply it to an unknown dataset and to validate (or not!) the quality of prediction. In order to get those two datasets, the original dataset was split into: a calibration dataset (xcal,ycal) containing the first 100 obervations ; and a test dataset (xtest,ytest) containing the last 40 observations.

```
-->xcal=nir.x(1:100,:);
-->ycal=nir.y(1:100,:);
-->xtest=nir.x(101:140,:);
-->ytest=nir.y(101:140,:);
```

The PLS is called by the command pls for a standard calculation, or by the command ikpls for a calculation with the improved-kernel pls algorithm (quicker). For instance:

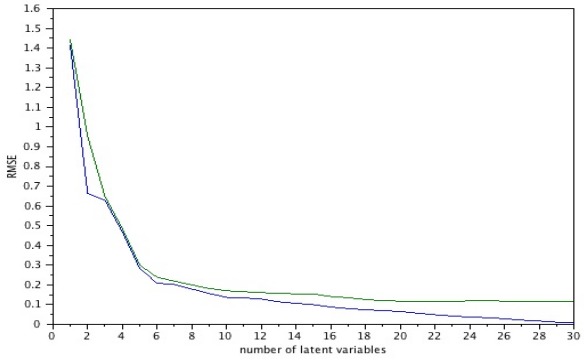
```
--> model=pls(xcal,ycal,10,30)
 model  =
   err:  div
   ypredcv:  div
   b:  div
   scores: div
   loadings: div
   x_mean: div
   y_mean: div
   center: 1
```

The PLSR has been calculated with a cross validation of 10 consecutive block and 30 latent variables (LV). Centering is the default option. The field model.err.d contains two vectors:

the root mean square error of calibration (RMSEC) and the root mean square error of cross-validation (RMSECV). These curves are printed with curves:

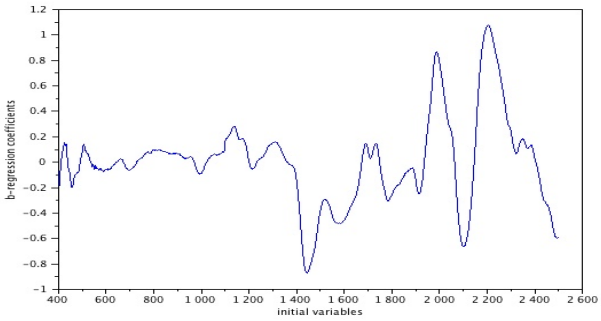
```
--> curves(model.err);
```

The RMSEC and the RMSECV are printed in blue and green respectively. According to this figure, we choose a model with 6 LV.



The b-coefficients of the model with 6 VL are also plotted:

```
--> curves(model.b(:,6))
```



Then the model is applied to the test dataset using the regapply command, the same for all regression methods:

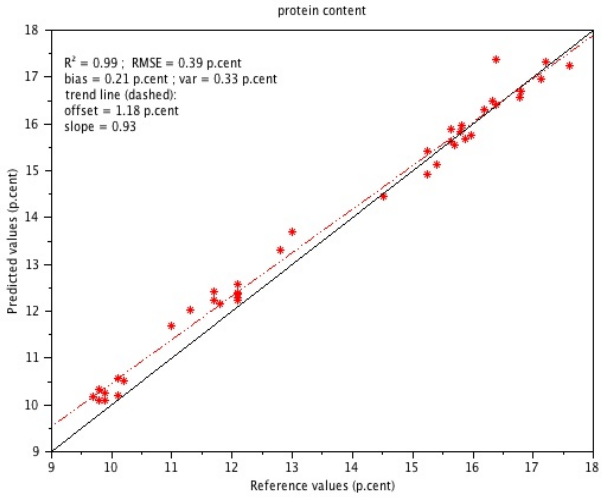
```
-->pred=regapply(model,xtest,ytest)
 pred  =
   ypred:  div
   rmsep:  div
   r2:  div
```

The standard error of prediction RMSEP can also be plotted.

Note that all the models calculated with the regression method are evaluated simultaneously. Thus pred.ypred.d is a matrix of dimensions (40 x 30), 40 observations and 1 to 30 latent variables.

The regplot command compares the prediction of the model with 6 latent variables to the reference values:

```
--> y6=pred.ypredtest(:,6);  // selection of the 6th variable of ypredtest
--> h=regplot(ytest,y6,'r*','t','p.cent','protein content') ;
```



The options chosen for regplot in this example are:

'r*' :	observations are represented by red stars;
't':	representation of the trend curve;
'p.cent':	units of measurement;
'protein content':	title of the figure.

Note that “biais” and “var” are the mean and variance respectively of the residuals (differences) between the predicted and the reference values; and $RMSE^2 = biais^2 + var^2$.

3.2. Pretreatments.

Pretreatments aim at removing a spectral information which is detrimental for building calibration models.

Some pretraitments are very popular in near infra-red spectroscopy where spectral deformations are often very important:
snv normalizes the spectra; veru useful in case of a multiplicative effect;
detrending corrects simple deformations of the baseline as vertical shift and slope.

These methods are vey easy to use, see the syntax using the corresponding hep. But other pretraitments are a bit more difficut to apply because they need supplementary data; among them we will focus on orthogonal projections. Two of them are detailed below: external parameter orthogonalization (EPO) and error removal by orthogonal subtraction (EROS).

The principle of orthogonal projections.

The detrimental information is represented by a matrix **D** obtained from data usually issued from an experimental design. Then this information is represented by the eigenvectors of a PCA onto **D**. The tuning of the model consists in determining the number A of these eigenvectors representing at best the detrimental information. The correction is obtained by a projection of the spectra orthogonally to these A first eigenvectors.

Data dedicated to orthogonal projection.

The data are issued from an experimental design targeting a negative influence to be corrected. Let's take the temperature as an example.

A possibility consists in the acquisition of spectra onto one ore more samples at different temperatures (not necessary the same temperature for each sample; the sampes can vary); the EROS method can be applied.

Another possibility consists in the acquisition of spectra onto a same set of samples at different levels of temperature (all the samples have the same temperature); the EPO and EROS methods can be applied.

The example illustrates both EPO and EROS. The data are in the file epo_apples.dat which contains the following fields:

```
-->apples
apples  =

  x1: [1x1 struct]
  xcal: [1x1 struct]
  ycal: [1x1 struct]
  xtest: [1x1 struct]
  ytest: [1x1 struct]
```

The calibration and test datasets are (xcal,ycal) and (xtest,ytest) respectively. x1 contains the spectra of 10 apples acquired at 8 different temperatures; x1.i identifies simultaneously the apples (1st character, letters A to H) and the temperatures (2nd to 3rd character, 5/10/15/20/25/30/35/40°C). In the following these data are supposed to be extracted, yielding the variables: x1,x1_obs,x1_temp, xcal, ycal, xtest et ytest. It simplifies the notations, x1 is easier to type than apples.x1. For instance:

```
-->x1=div(apples.x1);
-->x1
 x1  =
  d: [80x256 constant]
  i: [80x1 string]
  v: [256x1 constant]
```

We will need to use the codes of the samples c_ech (the apples) and the codes of the detrimental influences c_gi (the temperatures). They are obtained with str2conj which extracts the identifiers of the groups withing the strings:

```
-->c_ech=str2conj(x1.i,1,1);
-->c_gi=str2conj(x1.i,2,3);
```

Calculation of external parameter orthogonalization (EPO)
EPO is called by the command epo according to the following example:

```
-->[res_epo]=epo(x1,c_ech,c_gi,xcal,ycal,10,8)
res_epo  =
  d_matrix: div
  eigenvec: div
  ev_pcent: div
  wilks: div
  rmsecv: div
  pls_models: div
```

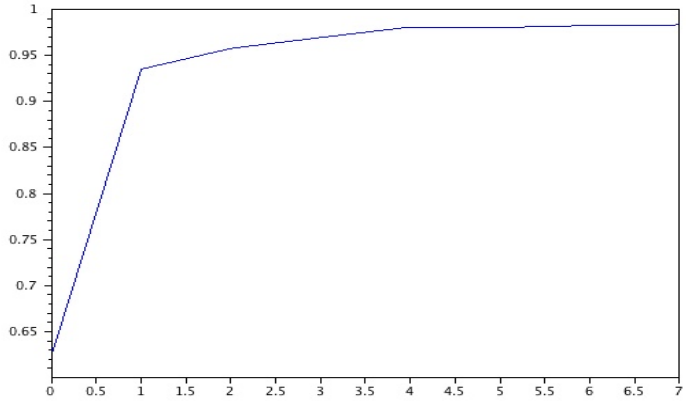
The numbers 10 and 8 indicate a cross validation with 10 groups and a PLS regression with up to 8 latent variables.

We can notice that c_ech (the code of the samples) appears among the input arguments. It is not used by the EPO, but it is necessary for the calculation of the Wilks lambda.

The fields d_matrix and eigenvec are the ones of the matrix of the detrimental influence and its first eigenvectors. The dimension of the correction is determined using the helps: the eigenvalues in ev_pcent, the Wilks lambda in wilks and the RMSECV in rmsecv. For example:

```
-->curves(res_epo.wilks)
```

yields the following figure. The scale between 0 and 7 recalls that the first model is not corrected by EPO.
The Wilks lambda is a multivariate criteria to discriminate groups: 0 = no differences between groups; 1 = groups well separated. The EPO correction is expected to increase Wilks lambda while increasing the number of dimensions. Effectively it increases greatly up to 5 principal components and after low. Thus the dimension of the correction is set to 5.



The field pls_models is a list of structures which contains all the PLSR models between 1 and 8 latent variables. Each element of the list is associated to a correction by orthogonal projection involving from 0 to (8-1 =7) principal components:

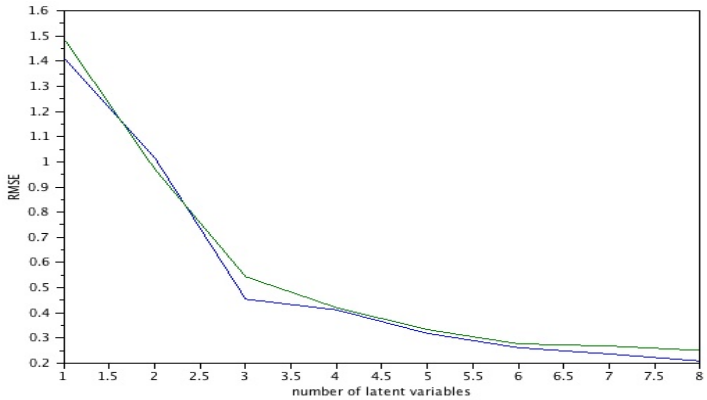
pls_models(1)	no correction by orthogonal projection
pls_models(2)	EPO with 1 componant

… ...

pls_models(8) EPO with 7 componants

An EPO correcting 7 dimensions corresponds to the 8th and last model, for which the values of the RMSECV and the RMSECV are plotted with the following figure:

-->curves(res_epo.pls_models(8).err)



The RMSECV of the PLSR dicreases with the increase of the number of latent variables and suggests 6 latent variables.

Calculation of error removal by orthogonal subtraction (EROS) :

EROS is called by the following command eros:

```
-->res_eros=eros(x1,c_ech,xcal,ycal,10,8)
 res_eros  =
   d_matrix: div
   eigenvec: div
   ev_pcent: div
   wilks: div
   rmsecv: div
   pls_models: list
```

EROS is based on the samples (the apples) and does not take into account the levels of the detrimental influence (the températures) to which the samples are exposed, so compared to EPO there is one parameter less to fill. Nevertheless the outputs (res_epo et res_eros) contain the same fields.

Application of EPO and EROS :

The models obtained by EPO and EROS were res_epo and res_eros respectively. These models are applied to the test dataset (xtest,ytest) with the commande popapply according to the following example:

```
-->res_test=popapply(res_epo,5,xtest,ytest)
```

```
res_test  =
  ypred: div
  rmsep: div
  r2: div
```

We had previously chosen 5 dimensions for EPO and 6 latent variables latentes for PLSR. The graph of the prediction errors shows that this choice was correct, the corresponding RMSEP being 1,3706 among the lowest values and close to the best *a posteriori* choice (5VL, RMSEP=1,2811).

The 6th coloumn of res_test.ypred.d contains the predictions for 6 latent variables.

Another possibility:
The functions pop_dextract then pop_dtune can also perform pretreatments by orthogonal projections (POP). They are used when the correction involves two or more detrimental influences, each represented by a different data or model.
Suppose that a Detrend correction (also an orthogonal projection) is also expected in the previous example. EPO and EROS can be performed by epo, eros and pop_dextract and yield **D** the matrix of detrimental information, into the field dmatrix. Detrend is obtained by the function detrending which yields a Vandermonde matrix named **L**. The two matrices **L** and **D** are merged then pop_dtune performs the correction by orthogonal projection.

3.2 Discrimination

The principle of discrimination methods.
Three discrimination methods are available in Fact: factor discriminant analysis (FDA), PLS discriminant analysis (PLS-DA) and step-wise discriminant analysis. Each of these methods has its own algorithm to determine the directions into the space (associated to a metric) that discriminate at best the different groups. Then the coordinates or scores of the observations into these spaces are obtained. The probabilities that an observation belongs to a group are defined with an Euclidian or a Mahalanobis distance (by default). Each observation is attributed to the group to which it is the closest, provided that the distance is upper to a given threshold. The confusion matrices represent the numbers of observations attributed to each class (in row) compared to the real belonging of the observations (in columns). The diagonal contains the good classifications, their number is compared to the total number of observations. Calculations are processed with all the calibration dataset or with cross-validation.

The data to process a discrimination.
Discrimination methods need a matrix e.g. **X** of variables acquired for several observations and a vector e.g. gr identifying the group of each observation. If **X** is of dimensions (*n x q)*, gr is a vector of dimensions (n x 1). This vector contains integers with values between 1 and maxgroup, where maxgroup is the number of groups. Each group must be represented at least one time in the calibration dataset.
When the labels of the observations are keys that can explain the groups, these labels can be easily used to obtain gr . For the example concerning the olive oils at paragraph 2.1.3, the region of origin is identified by the 2 first letters of the labels.
Thus gr is obtained directly by the command str2conj:

```
-->[gr,labels_regions, nbr_obs]=str2conj(olive1.i,1,2);
```

gr is a vector containing integers between 1 and 9 which represent each of the 9 groups.
labels_regions and nbr_obs give the codes of the groups/regions and the number of observations respectively:

```
-->labels_regions
 labels_regions  =
!Ca  !
!Cs  !
!El  !
!Is  !
!Na  !
!Sa  !
!Si  !
!Um!
!Wl  !
```

Factorial discriminant analysis (FDA)

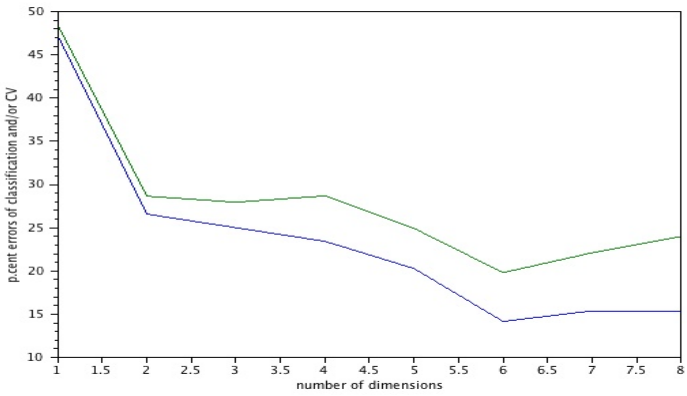
The factorial discriminant analysis is designed for data containing highly correlated variables such as chromatograms or spectra. Similarily to the principal component regression, it processes in two steps: a PCA onto the data followed by a linear discriminant analysis.

The factorial discriminant analysis is called by the command fda :

```
-->res_fda=fda(olive1,code_group,10,8)
 res_fda  =
   conf_cal_nobs: list
   conf_cal: list
   conf_cv: list
   err: div
   errbycl_cal: div
   errbycl_cv: div
   notclassed: div
   notclassed_bycl: div
   method: "fda"
   xcal: div
   ycal: div
   loadings: div
   classif_metric: 0
   scale: "c"
   classif_opt: 0
   threshold: 0.1111111
```

The percents of error of calibration and cross-validation, values between 0 and 100, are in the field res_fda.err and can be plotted:

```
--> curves(res_fda.err);
```



Note that res_fda.conf_cal_nobs, res_fda.conf_cal_nobs and res_fda.conf_cal_cv are lists containing 8 Div structures in this example, one for each dimension. If 7 dimensions are selected, according the figure above, the confusion matrix for 7 dimensions is:

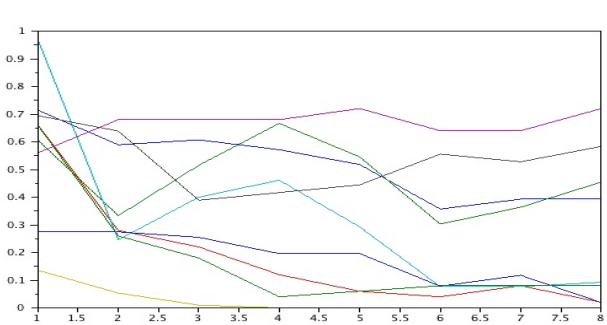
```
-->cm7=res_fda.conf_cal_nobs(7);
-->cm7.d
 cm  =

   34.    0.    0.    0.    0.    0.    1.    0.    0.
    0.   21.    0.    0.    0.    0.    0.    0.    0.
    0.    0.   46.    0.    0.    0.    0.    0.    0.
    0.    0.    0.   60.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    9.    0.    0.    0.    0.
   17.   12.    4.    5.   12.  206.   18.    6.    4.
    5.    0.    0.    0.    4.    0.   17.    0.    0.
    0.    0.    0.    0.    0.    0.    0.   45.    0.
    0.    0.    0.    0.    0.    0.    0.    0.   46.
```

All the 206 observations of the group 6 are well classified into the group 6; but other observations attributed to the group 6 belong in fact to the other groups, e.g. 17 for group 1.

The details of classification errors for each class and each dimension are obtained with res.errbycl_cal and res.errbycl_cv :

```
-->curves(res_fda.errbycl_cal)
```



The PLS-discriminant analysis (PLS-DA)

The PLS-DA first builds a PLS2 model using the VODKA method. The reference values are represented by **Y**, the disjuctive matrix identifying the groups. The outputs are the same than for FDA. For the comparison, we only present the call of plsda and the confusion matrix for 7 dimensions (or latent variables for the PLS-DA) :

```
-->res_plsda=plsda(olive1,code_group,10,8);
-->cm7bis=res_plsda.conf_cal_nobs(7);
--> cm7bis.d
   35.    0.    0.    0.    0.    0.    1.    0.    0.
    0.   20.    0.    0.    0.    0.    0.    0.    0.
    0.    0.   48.    0.    0.    0.    0.    0.    0.
    0.    0.    0.   57.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    7.    0.    0.    0.    0.
   19.   13.    2.    8.   18.  206.   20.    3.    5.
    2.    0.    0.    0.    0.    0.   15.    0.    0.
    0.    0.    0.    0.    0.    0.    0.   48.    0.
    0.    0.    0.    0.    0.    0.    0.    0.   45.
```

The forward discriminant analysis:

The variables are added one after the other by the forwda function:

```
-->res_forwda=forwda(olive1,code_group,10,8);
```

Contrary to the other methods, forwda applies a threshold to add a new discriminant variable. In our example, the maximum is 6 variables and according to the cross-validation the best model is obtained for 3 variables:

```
-->cm7ter=res_forwda.conf_cal_nobs(3);
--> cm7ter.d
 ans  =
   32.    0.    0.    0.    1.    1.    1.    0.    0.
    0.   24.    0.    0.    0.    0.    0.    0.    0.
    0.    0.   43.    0.    0.    0.    0.    6.    4.
    0.    0.    0.   57.    0.    0.    0.    0.    1.
    0.    0.    0.    0.    8.    0.    2.    0.    0.
   10.    9.    0.    7.    3.  205.   10.    0.    3.
   14.    0.    0.    0.   13.    0.   23.    0.    0.
    0.    0.    4.    0.    0.    0.    0.   45.    0.
    0.    0.    3.    1.    0.    0.    0.    0.   42.
```

For all the discriminant methods:

Only for the cross-validation, the confusion matrices represent percentages and not a classification of each observation as for calibration. The reason is that it can happen that all the elements of a same class be in the calibration dataset, or in the validation dataset: therefore their class cannot be estimated. To avoid this problem, the cross-validation is repeated 10 times and that explains the use of the percentages.

3.3 Multi-table analysis.

A selection of multi-table methods is proposed: ccswa, comdim, statis

These methods can be used and an help is provided. However an update with harmonization of their outputs is schedulled. Then a demo will be added.

4 Univariate analysis: ANOVA+SNK

Fact is not statistically oriented but it contains an analysis of variance for one facteur which deals simultaneously with several variables. It is followed by Student-Newman-Keuls which is a test for the classification of the means. The ouptuts are simplified according to the informations needed in chemometrics.

An example is given by the file pph.dat. After loading:

```
-->pph
 pph  =
   d: [21x5 constant]
   v: [5x1 string]
   i: [21x1 string]
-->pph.v
 ans  =
!B1      !
!B2      !
!B3      !
!B4      !
!epicat !
```

The 5 variables are the tanins: the dimers B1, B2, B3, B4 and the monomer epicatéchine.
The 21 observations are represented into pph.i by their class identifier, cl1 à cl7: thus we have 7 classes of 3 repetitions each.

The analysis of variance is called by the command snk:

```
-->res=snk(pph,pph.i)
 res  =
   anova: [7x1 string]
   snk: list
```

The first argument pph is the data to be processed, the second argument pph.i is the identifier of the classes. The output res contains two fields: res.anova for the results of the ANOVA and res.snk for the results of SNK.

```
-->res.anova
```

```
 ans  =
!ANOVA 1 factor, 7 classes of 3 repetitions on average  !
!Variable   SCEA     SCER     F(6,14)     Pr>F            !
!B1         1513.    28.64    123.2       0               !
!B2         1.290    1.244    2.419       0.081           !
!B3         0.229    0.068    7.812       0               !
!B4         0.207    0.077    6.293       0.002           !
!epica      3.932    0.406    22.59       0               !
```

SCEA and SCER are the sums of squares explained respectively by the studied factor and the residual. F is the calculated Fischer value and Pr>F is the probabily that there are no differences between the means.
The ANOVA takes into account different numbers of observations in the classes; only the average is reported.

res.snk is a list; res.snk(i) contains the results of SNK for the variable i. For example the 5th variable is epicat:

```
-->res.snk(5)
 ans  =
!STUDENT_NEWMAN_KEULS 5%:    !
!classes nobs mean    epicat  !
!cl7      3    2.144   A       !
!cl6      3    2.060   A       !
!cl2      3    1.593     B     !
!cl5      3    1.316     B C   !
!cl3      3    1.269     B C   !
!cl4      3    1.092       C   !
!cl1      3    0.933       C   !
```

SNK is designed for classes containing the same number of observations (nobs). If it is not the case, the calculation is still performed with a mean number of observations rounded to the nearest integer; but the more the numbers of observations will be different, the more the errors will increase.

The last column gathers by a same character the means which are not significatively different with an error of 5p.cent. In this example cl6 and cl7 are different from the 5 other classes; cl2 is different from cl1 and cl4 but not from cl3 or cl5.

5 Other useful informations

5.1. More informations about Fact

Fact can be downloaded from the Atoms module of Scilab.
This help file and the data files are available in “getting_started.zip” on the Fact page of Scilab:

http://atoms.scilab.org/toolboxes/FACT

5.2. And for Matlab?

Saisir for Matlab and Fact for Scilab are different. At present Fact has not been designed for Matlab users who can use Saisir 1.0 and download it directly at:

http://www.chimiometrie.fr/saisirdownload.html

5.3. Rights and duties of the users.

Fact has been protected in France at APP. Nevertheless it is freely released under the Cecill-C licence and thus there is absolutely no guarantee about the use of Fact and its consequences.

We would be grateful to users that would mention the use of Fact when they communicate about results obtained with this toolbox.

Correspondance about Fact can be adressed to:

JC Boulet
bouletjc@supagro.inra.fr