FACT
UN ENVIRONNEMENT EN CHIMIOMETRIE
SOUS SCILAB ®

MODE D'EMPLOI

Pour FACT_0.3 et FACT 0.4

mise à jour : 13 Mai 2014

Avis aux l'utilisateurs

- Il est déconseillé de démarrer l'utilisation des fonctions Fact sans lire ce document, ce n'est pas très long!

• *L'utilisation de Fact nécessite de connaître l'environnement* Scilab.

• *Toutes les fonctions ont une aide que l'on peut obtenir par la commande*

help <fonction>

qui ouvre une fenêtre html avec moteur de recherche.
Pour accéder à la liste des fonctions d'usage:

help abc_fact
help fact

• *Si le résultat donné par une fonction n'est pas clair, on peut toujours taper le nom de l'argument de sortie, puis les champ de ce résultat pour savoir la nature des paramètres de sortie. Par exemple une analyse en composantes principales est obtenue par:*

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

Mais que contient donc p *?*

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

On peut alors imaginer que p.scores *contient les scores ou coordonnées,* p.eigenvec *les vecteurs propres (eigenvector) et ainsi de suite.*

Comment installer Fact/Scilab :

Scilab est installé depuis : www.scilab.org
1- ouvrir Scilab, se positionner dans la console, ouvrir l'onglet « Applications »
-cliquer sur « gestionnaire de modules-ATOMS »
- l'option « tous les modules » les classe par ordre alphabétique ; choisir Fact
- cliquer sur 'install'
1-autre possibilité : ouvrir la page Fact (http://atoms.scilab.org/toolboxes/FACT)
téélcharger le fichier binaire (ex : FACT_0.4-3.bin.zip) dans un répertoire
aller dans le même répertoire avec le navigateur de fichiers de Scilab
verifier la connection internet !
taper dans la console: atomsInstall('FACT_0.4-3.bin.zip')
2- relancer Scilab *Fact est installé*
- désinstallation possible en cliquant sur « supprimer »

FACT

UN ENVIRONNEMENT EN CHIMIOMETRIE
SOUS SCILAB®

MODE D'EMPLOI

Table des matières

Introduction :

La manipulation des grands fichiers de données provenant d'études chimiométriques, en particulier lorsque certaines des mesures sont obtenues avec des instruments physiques (spectromètres, chromatographes, pénétromètres, images …) n'est pas toujours possible dans des environnements statistiques habituels de type tableurs.

Afin de disposer d'un outil de manipulation de données, adapté à des tableaux comportant plusieurs milliers de lignes ou de colonnes, situation assez courante en spectroscopie, nous avons développé un ensemble ouvert de procédures sous les environnement Scilab et Matlab, susceptible de faciliter le travail des expérimentateurs ainsi que de permettre un enrichissement des possibilités du système par la communauté chimiométrie. Seule la version Scilab est présentée ici (pour Matlab, voir à la fin de ce document).

Une première boite à outils (BAO) concue et réalisée par Dominique Bertrand (INRA) a été dénommée « SAISIR » (*Statistiques Appliquées à l'Exploitation des Spectres Infrarouge*). Elle comprend des fonctions de chargement et sauvegarde des données, de manipulation, des outils graphiques et des procédures statistiques : analyse multidimensionnelles, discrimination, régression, …Saisir a été concue et implémentée dés 1998 pour une utilisation sous Matlab. Une de ses particularités est d'utiliser le format Div développé avant 1985 en BASIC par D.Bertrand pour garder tout au long des calculs les informations concernant les observations.

L'adaptation de Saisir à Scilab a conduit à une mise à jour des fonctions, certaines ajoutées, d'autres supprimées ou remplacées. Les modifications sont suffisamment conséquentes pour que la nouvelle BAO puisse être considérée comme différente de Saisir. Afin d'éviter toute confusion, le nom a été changé en : « FACT » (*Free-Access Chemometric Toolbox*). Fact a gardé de nombreuses fonctions ainsi que l'esprit de Saisir. L'utilisation de Fact suppose un minimum de connaissance de base de l'environnement et de la programmation sous Scilab ainsi que des outils chimiométriques utilisés.

1. Principes généraux

Les environnement Saisir et Fact reposent sur la manipulation de tableaux de données présentés sous forme de structures. Le principe fondamental, respecté dans toutes les procédures, est que les colonnes et les lignes des matrices de données comportent des identifiants ou labels, qui « suivent » l'ensemble des manipulations. Examinons, par exemple, le fichier suivant reportant les notes d'analyse sensorielle de 3 pommes nommées « GALA1 », « FUJI1 », « FUJI2 », pour les trois caractéristiques odeur globale : « OGLO », odeur de terre «OTER », odeur de cave « OCAV » :

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

(dans cet exemple le séparateur décimal est le point et non la virgule)

Ce fichier comporte 3 lignes et 3 colonnes. Les lignes (ou *individus*) possèdent des identificateurs (« GALA1 », « FUJI1 » et « FUJI2 ») de même que les colonnes (ou *variables*) sont identifiées par « OGLO », « OTER » et « OCAV ». Enfin les données forment la matrice :

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

Pour importer complètement le tableau, il est nécessaire de garder ces trois informations.
Le format utilisé par les environnements Saisir et Fact pour conserver un tableau de données est *une structure DIV* qui va contenir au moins trois informations :
→ l'information sur les données ;
→ l'information sur les observation ;
→ l'information sur les variables.

Dans les cas simples de spectroscopie où une observation est représentée par un spectre, le format Div contient trois champs :
<> .d pour « data/données »
<> .i pour « individuals/individus » (lignes) ;
<> .v pour « variables » (colonnes).

Le format Div est une structure correspondant à une liste typée. Elle est obligatoirement construite par la fonction div ou d'autres fonctions utilisant div, selon deux possibilités :

a) créer manuellement une structure contenant les champs .d,.i et .v, puis appliquer 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) appliquer div en fournissant les champs .d, .i et .v dans cet ordre:

```
-->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]
```

La structure Div se reconnaît avec la commande typeof :

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

A l'écran l'interligne entre les champs .d, .v et .i est simple pour les structures normales, double pour les structures Div.

Chaque champ peut être extrait, par exemple :

```
--> 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
```

Il faut noter que les champs « i » et « v » sont des vecteurs de nombres (*constant*) ou des vecteurs de chaines de caractères (*string*) ; c'est une particularité de Scilab qui n'utilise pas des vecteurs de cellules (*cell array*) comme Matlab.
Un cas particulièrement important concerne les données qui sont sous la forme de courbes numérisées (spectres, chromatogrammes, électrophoregrammes …). Dans ce cas, les données sont le plus souvent représentées par des matrices avec *en ligne* chaque courbe. Les identificateurs des variables sont alors des *nombres mis sous la forme de chaînes de caractères* représentant la graduation en x de la courbe (longueur d'onde, temps de rétention, longueur de déplacement …). Voici par exemple, un fichier (simplifié) de données de spectres proche infrarouge :

	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

Comme précédemment « 1br01 », « 1br51 », « 1fu21 », « 1fu71 » sont des identifiants d'individus ; et « 1100 », « 1102 », « 1104 », « 1106 », « 1108 » sont des chaînes de caractères représentant les identifiants des variables. Supposons que la structure Div de ces données s'appelle spectre, on a:

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

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

La nature numérique des descripteurs de variables est exploitée dans les représentations de courbe (fonctions curves et tcurves).

2. Utilisation de l'environnement Fact

La commande help ouvre une fenêtre au format html expliquant la syntaxe de la fonction. La liste des commandes de Fact s'obtient par :
-->help abc_fact
Les commandes sont classées par thème ce qui permet de trouver rapidement le nom de la commande recherchée. En haut de la fenêtre il s'affiche :

fact >> fact > abc_fact

Cliquer sur l'un des fact et la liste alphabétique des fonctions s'affiche. Cliquer sur la fonction recherchée pour avoir les détails.

2.1 Prise en main

Les données compatibles avec l'environnement Fact sont disposées sous la forme de tableaux rectangulaires (matrices). Dans son état actuel, l'environnement ne gère pas les données manquantes.

2.1.1 Chargement, création et sauvegarde de structures Div

Chargement à partir de fichiers Excel et OpenOffice

La fonction csv2div importe des tableaux provenant d'Excel ou OpenOffice sous réserve qu'ils aient le format suivant dans le tableur:
<>la première ligne contient les identifiants des colonnes ;
<>la première colonne contient les identifiants des lignes ;
<>les autres cellules contiennent des *valeurs numériques*, avec *le séparateur décimal représenté par le point « . » ou la virgule « , » ;*
Lors de l'importation, la cellule en position (1,1) n'est pas conservée.
Exemple du fichier « fruits » sous Excel ou OpenOffice :

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

Le fichier au format Excel ou OpenOffice doit ensuite être sauvegardé sous le format « .csv » :
<> avec Excel, cliquez sur enregistrer sous puis CSV (Séparateur point virgule) ;
<> avec OpenOffice : cliquez sur enregistrer sous puis csv / editer les paramètres du filtre puis séparateur de champ : « ; » et effacer le séparateur de texte par défaut.

Le chargement sous Scilab s'effectue par l'ordre : res = csv2div('filename') ;
où filename est une chaîne de caractères indiquant le fichier à charger (depuis le répertoire de travail). res est la structure Div. Par exemple :

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

charge le fichier fruits.csv et le place dans la structure Div essai.

La fonction csv2div importe les valeurs manquantes qui ont été remplacées par NaN dans le fichier d'origine .csv. Mais si les valeurs manquantes sont représentées par un champ vide, alors csv2div génère un message d'erreur.

Export de structures Div vers un fichier .csv (pour utilisation par Excel ou OpenOffice) :
L'ordre div2csv permet de sauvegarder les données au format Div dans un fichier au format .csv. Le fichier est sauvegardé avec ses identificateurs de lignes et de colonnes, au format. csv (séparateur : point-virgule). Par exemple:

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

sauve la structure Div essai sous le nom TABLEAU.csv. Noter que le dernier argument est le choix du séparateur décimal, ici la virgule. Les fichiers .csv sont facilement importés par Excel ou OpenOffice.

2.1.2 Manipulation élémentaire des données

Des opérations simples sur des structures Div sont gérées par surcharge des fonctions de base; elles sont résumées dans le tableau ci-dessous. Bien évidemment les dimensions doivent correspondre.

Opération	Scalaires	Matrices	Structures Div	Syntaxe	Detail du calcul réalisé
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
Soustraction			a,b,c	c=a-b	c.d=a.d-b.d c.i=a.i c.v=a.v
Multiplication par un scalaire	s		a,c	c=s*a	c.d=n*a.d c.i=a.i c.v=a.v
Division par un scalaire	s		a,c	c=a/s	c.d=a.d/n c.i=a.i c.v=a.v
Multiplication de structures Div			a,b,c	c=a*b	c.d=(a.d)*(b.d) c.i=a.i c.v=a.v
Multiplication élément par élément			a,b,c	c=a.*b	c.d=(a.d).*(b.d) c.i=a.i c.v=b.v
Division élément par élément			a,b,c	c=a./b	c.d=a.d./b.d c.i=a.i c.v=a.v

Concaténation par lignes			a,b,c	c=[a;b]	c.d=[a.d;b.d] c.i=[a.i;b.i] ; c.v=a.v
Concaténation par colonnes			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]
Extraction de données	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 de données	p,q,r,s	m	c	c(p:q,r:s)=m	c.d(p:q,r:s)=m

Tableau 1: opérations possibles sur les structures Div à l'aide des opérateurs courants (' + - * / .* ./ [ ] [;])

Pour l'extraction ou l'insertion de données, les intervalles p:q ou r:s peuvent être remplacés par p ou r si une seule ligne ou colonne. Ainsi c=a(p,r) est valide mais pas c=a(p) même dans le cas d'un vecteur.

Les ordres précédents s'appliquent sur les indices, et non sur les descripteurs. Ainsi, pour sélectionner la ligne correspondant à Fuji2 il faut écrire, par exemple :

```
--> Fuji2=spectre(3,:) ;   //et non pas : Fuji2=spectre(Fuji2,:)
```

Il est parfois fastidieux (dans des grandes matrices), de retrouver le numéro d'ordre d'un individu ou d'une variable, connaissant son identifiant. Pour trouver ce numéro, on peut utiliser l'ordre strseek, par exemple:

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

renvoie les numéros d'ordre de tous les individus de essai.i qui ont la chaîne de caractères « FUJI » dans leur nom. Les minuscules et les majuscules sont considérées comme différentes.

Si on dispose de données de type numérique, on peut utiliser la commande indexseek qui retourne une et une seule valeur, l'index de spectre.v dont la valeur est la plus proche d'une valeur donnée. Par exemple :

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

Notons qu'il n'y a pas la nécessité d'avoir une correspondance exacte des caractères ; cette commande trouvera l'index approprié même si la variable « 1104 » est en fait codée par la chaine « 1103.9996 ».

Utilisation des identificateurs comme clé d'extraction

Il est très avantageux, et pratiquement indispensable pour de grandes matrices, d'utiliser un système de noms d'individus pouvant servir de clé d'extraction. De nombreuses procédures

(analyse discriminante, analyse en composantes principales, analyse de variance, graphique) sont simplifiées si l'utilisateur a respecté ce principe.
Le principe est simple à expliquer sur un exemple.
Supposons qu'un essai porte sur trois variétés de blé (*Camp Rémi*, *Talent* et *Arminda*), cultivées dans deux lieux de culture (*Paris* et *Montpellier*). On effectue 20 répétitions par essai. Dans ce cas, un descripteur correct d'échantillon peut être, par exemple :
CRPA09 qui signifie : *Camp Rémi*, cultivé à *Paris*, répétition 9.
On a ainsi décidé d'utiliser deux lettres (ici CR) pour désigner la variété, deux lettres pour la région de culture (PA), *deux* lettres pour la répétition. La répétition numéro 9 est désignée par '09' et non '9' pour que l'on n'ait pas de problème entre 10 et 99 répétitions !
De même, on aura TAMO19 : *Talent*, cultivé à *Montpellier*, répétition *19*.
Il faut surtout éviter de changer les dimensions des champs d'identifiants. Par exemple, CRMO12 et ARMPA10 ne sont pas compatibles, car le code CR comprend 2 caractères, tandis que le code ARM en comprend 3 ! Il faut aussi éviter que les mêmes lettres se retrouvent dans plusieurs codes : ainsi PA désignant le lieu de production ne peut plus être utilisé pour désigner une variété de blé.

Les codes des identificants peuvent être utilisés pour extraire des sous-fichiers. Supposons qu'on veuille extraire les blés de Paris de la structure Div wheat contenant les données. Cela est possible avec la séquence suivante :

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

La première ligne détermine les index des observations contenant PA, la deuxième ligne construit la structure Div.

2.1.3 L'analyse en composantes principales

L'analyse en composantes principales permet un premier examen des données, avant d'appliquer d'autres méthodes. De plus, cette méthode présente l'ensemble des éléments et les graphiques de l'environnement Div.

Un premier exemple porte sur le jeu de données *Huiles d'olive* décrit dans l'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).

Afin de caractériser des huiles d'olives provenant de différentes régions de l'Italie, les auteurs ont dosé 8 acides gras (*Palmitic*, *Palmitoleic*, *Stearic*, *Oleic*, *Linoleic*, *Eicosanoic*, *Linolenic*, *Eicosenoic*) de 572 échantillons d'huile d'olive, collectés dans 9 régions. Les données sont dans un fichier .csv nommé olives.csv. Les lignes forment les 572 individus. La matrice comprend 8 colonnes correspondant aux 8 dosages d'acide gras. On a ainsi une matrice de dimensions (572 x 8).

Les identificateurs des individus comprennent deux caractères indiquant la région de culture. Par exemple Ca005 indique que le 5ème individu a été collecté dans la région Ca pour Calabria. La matrice est chargée dans Scilab :

```
--> olive1=csv2div('olives.csv')
```

```
olive1=
  d: [572x8 constant]
  v: [8x20 string]
  i: [572x20 string]
```

L'ACP est réalisée par les commandes pcana ou cspcana sur une matrice *sans données manquantes*, *ne contenant que les lignes et les colonnes qui font partie de l'analyse (individus et variables actifs).*
La différence entre pcana et cspcana est que cspcana centre et réduit systématiquement les données alors que par défaut pcana ne fait ni centrage ni réduction. Toutefois le centrage et la réduction peuvent être effectués manuellement avec :
centering: centrage des colonnes
standardize: normalisation ou standardisation des colonnes (variance=1)
Ainsi les options suivantes sont équivalentes :

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
```

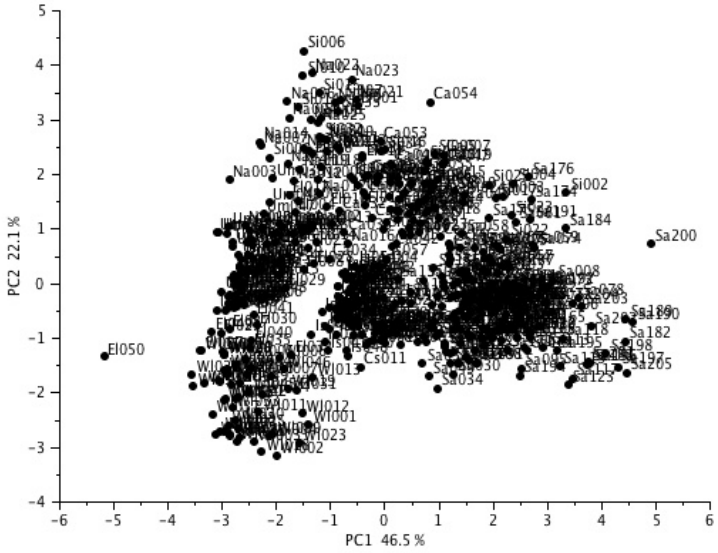
Les sorties sont les mêmes à l'exception des options centred et std. En effet celles-ci rapportent les prétraitements faits par les fonctions pcana ou cspcana lors du calcul, pas des prétraitements faits avant l'ACP.
Dans ces exemples, res est une structure contenant tous les résultats d'une ACP. Les champs scores, eigenvec, var_scores, eigenval et x_mean sont des structures Div complètes.

Un champ très utile est res.scores (matrice des coordonnées factorielles). Chaque ligne représente un individu, et chaque colonne une composante, dans l'ordre décroissant des valeurs propres. Les identificateurs des lignes de res .scores sont donc la recopie des identificateurs de olive1.i ou olive3 Les identificateurs des colonnes de res.scores sont créés par les fonctions pcana ou 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 %     !
```

Les *cartes factorielles* sont des représentations de paires de colonnes choisies par l'utilisateur dans res.scores sous la forme de graphiques X-Y. On peut représenter ces paires par l'ordre map et ses variantes.

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

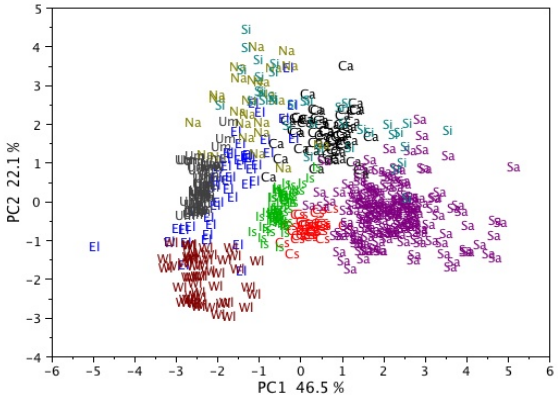


La carte représente les axes 1 et 2 de res.scores. Des informations sont ajoutées aux légendes des abscisses et ordonnées: les valeurs propres en pourcentage des axes 1 et 2.

Cette carte n'est pas très lisible sous cette forme. On peut, sur cette collection, se poser la question de l'influence de la région de culture. Il est très facile d'obtenir une carte coloriée par l'ordre coloredmap.
Par exemple, la commande

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

indique que l'on souhaite représenter les colonnes 1 et 2 de la matrice res.scores en coloriant la carte selon la clé indiquée par les deux arguments suivants. Ces arguments indiquent les positions (début/fin) qui délimitent la portion de chaines de caractères de res.scores.i qui servira de regroupement pour la coloration. Dans notre exemple la fonction extrait les chaînes de caractères de res.scores.i qui commencent en première position et finissent en deuxième position : Ca, Cs, El, Is, Na, Sa, Si, Um et Wl, et attribue une couleur particulière à chaque chaîne différente rencontrée. Chaque individu est représenté par sa chaîne en couleur.



On voit très clairement que les régions de culture ont une influence marquée sur la composition en acide gras des huiles d'olive. Les textes '46,5%' et '22,1%' sont rajoutés automatiquement à PC1 et PC2 respectivement.

D'autres options sont possibles pour représenter les classes :

chaque classe avec une couleur différente:
scmap : un symbole (<=7 classes) ou un nombre (>7 classes)
diacmap : 'Δ'
dotcmap : '.'
starcmap : '*'

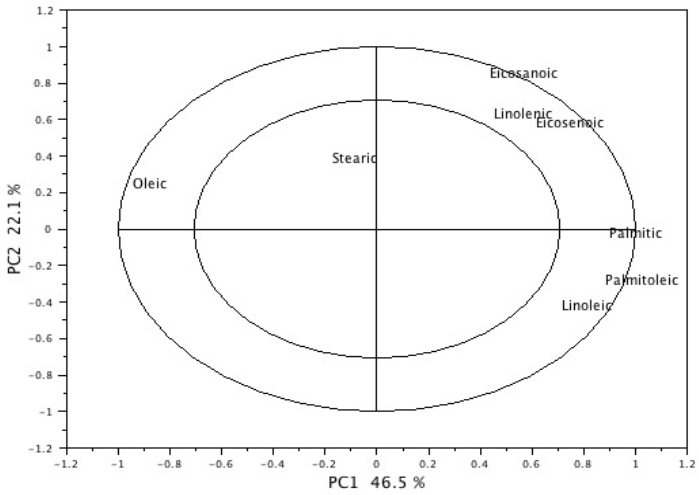
toutes les classes en noir et blanc :
kcmap : l'identifiant de la classe
kscmap : un symbole (<=7 classes) ou un nombre (>7 classes)

Le cercle de corrélation est obtenu par la commande corrplot selon la syntaxe : corrplot(scores,col1,col2, fact1,fact2, …). Le premier argument de cette fonction est une matrice dont les colonnes sont orthogonales entre-elles, par exemple les coordonnées (scores) d'une ACP ou d'une PLS ; col1 et col2 donnent les numéros des deux composantes de score à représenter ; les derniers arguments sont les structures Div pour lesquelles on souhaite visualiser la corrélation avec les deux scores. Par exemple, la commande :

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

représente le cercle des corrélations des variables de olive1 avec les composantes 1 et 2.

On peut aussi représenter des variables supplémentaires qui n'ont pas fait l'objet de l'ACP en rajoutant des tableaux en argument supplémentaire.



Deuxième exemple : ACP sur des spectres (courbes numérisées)
Ce deuxième exemple porte sur une collection de 140 spectres visible-proche infrarouge de farines de blé. Les descripteurs des individus comportent un code indiquant successivement l'année de culture (3 : 1993 ; 4 : 1994 ; la nature *dure* (D) ou *tendre* (T) du blé étudié, le numéro de la variété et les conditions agronomiques H1, H2, A1, A2. Les spectres sont enregistrés à des longueurs d'onde comprises entre 400 et 2500 nanomètres, avec un pas de mesure de 2 nanomètres, soient 1050 variables spectrales par spectre étudié. La structure Div des données spectrales est le champ nir.x du fichier nir.dat:

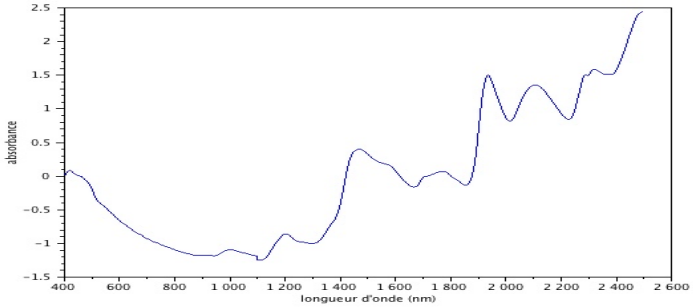
```
--> nir.x
ans =
      d: [140x1050 constant]
      i: [140x1 string]
      v: [1050x1 string]
```

On peut examiner les données spectrales par l'ordre curves, qui est prévue pour des vecteurs-colonne. Si les vecteurs sont en ligne, il suffit de les transposer, par exemple :

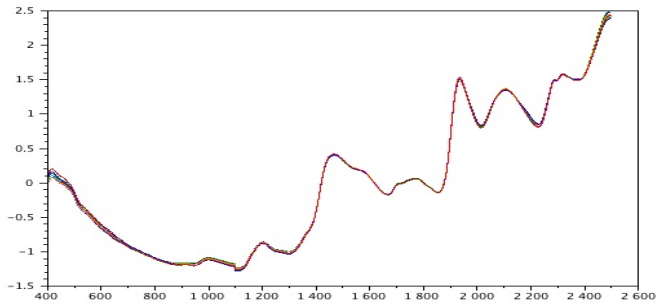
```
--> curves(nir.x(3,:),'','Longueur d''onde (nanomètre)','Absorbance')
```

représente le troisième spectre de la collection. On notera que nir.x(3,:) est un vecteur donc disposé en colonne; pour nir.x(3:4,:) qui est une matrice les spectres sont en ligne et on écrira la forme transposée:

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



Le deuxième argument permet de choisir la représentation de la courbe. Elle a été mise ici à la valeur par défaut " mais il est possible de choisir la couleur et le style de la courbe, p.ex 'r*' représente les observations par une étoile rouge.
L'axe des abscisses est correctement gradué selon les longueurs d'onde. L'ordre de Scilab: plot(nir.x.d(3,:)') effectue presque le même graphique, mais l'échelle des abscisses est simplement graduée de 1 à 1050 (nombre de points de mesures). Et les légendes ne sont pas imprimées simplement, au contraire de curves qui tente d'interpréter les identificateurs de variables comme des nombres, de manière à graduer correctement l'axe des X. Si ce n'est pas possible (les descripteurs de variables ne représentant pas des nombres), l'axe des X est gradué selon les numéros d'ordre des variables.
La fonction curves permet de représenter plusieurs courbes sur le même graphique. Par exemple, la commande curves(nir.x(1:10,:)') représente les 10 premiers spectres superposés.



et curves(nir.x ([3;6;9] , :)') représente les spectres des lignes 3, 6 et 9.

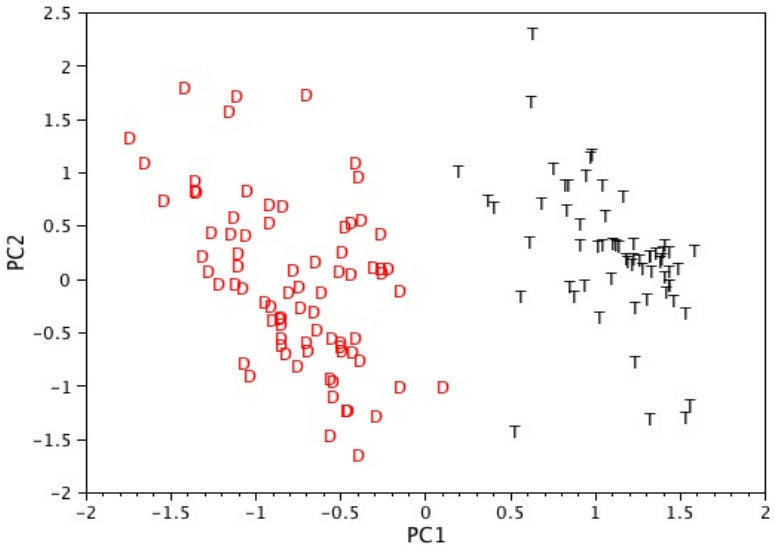
L'ACP sur des courbes numérisées est très voisine de celle effectuée sur des variables discrètes. Il faut cependant remarquer que, dans ce cas, les intensités relatives des variables ont un sens et qu'il ne faut donc pas réduire la matrice des données. Par contre le centrage reste possible, il est obtenu avec la commande pcana et l'option centrage.

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

Le programme calcule la totalité des composantes possibles (ici au nombre de 140). Les cartes factorielles s'obtiennent comme précédemment, par exemple :

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

représente le premier plan factoriel, en utilisant les caractères en deuxième position (commençant et finissant en position 2) pour colorier la carte. La nature *tendre/dure* des blés représentée respectivement par **T** et **D** est très clairement visible.



3 Analyses multivariées supervisées.

Des méthodes de régression et de discrimination sont présentées aux paragraphes 3.1 et 3.2 respectivement.

3.1. Régressions

Les méthodes de régression.
Plusieurs méthodes de régression sont disponibles : p*artial least square* (PLS), régression en composantes principales (PCR), régression *ridge*, régression linéaire multiple (*multiple linear regression*, MLR).
La régression la plus simple est la MLR, aussi dénommée méthode des moindres carrés. Malheureusement elle n'est pas applicable quand les variables sont fortement corrélées, donc celles-ci sont remplacées par des coordonnées ou scores orthogonaux entre eux avec la PLSR et la PCR.

Les données pour effectuer une régression.
Pour développer une méthode de régression, il faut disposer d'une matrice **X** comprenant *n* lignes et *q* colonnes (variables prédictives), et d'un vecteur **y** comprenant *n* lignes (variable à prédire) au format Div; **y** est prédit avec **X** . Les lignes de **X** et de **y** doivent évidemment correspondre aux mêmes observations (voir l'ordre reorder en cas de non-correspondance initiale). Les méthodes de régression de Fact ne permettent la prédiction que d'une seule variable à la fois, **y** est donc toujours un vecteur.
La régression PLS est présentée de manière détaillée. Les autres méthodes s'appliquent à peu près de la même manière.

La régression aux moindres carrés partiels ou PLSR (Partial least square regression)

La régression PLS est probablement la méthode la plus connue des méthodes d'étalonnage indirect ou régressions. Elle est efficace même lorsque les variables de **X** présentent une forte colinéarité. La « complexité » du modèle établi dépend d'un paramètre appelé « dimension ». Plus le modèle est complexe, plus il est précis, mais moins il est stable. Il est donc nécessaire de trouver un compromis entre la complexité et la stabilité. Le choix du nombre de dimensions peut se faire par une méthode de validation croisée (*cross validation*, voir plus loin). Dans la suite, ndim est un nombre indiquant le nombre de dimensions du modèle retenu.

Reprenons l'exemple précédent portant sur le blé. En face des spectres, nous disposons des valeurs de référence sur la concentration en protéines dans nir.y. La première étape est de construire un modèle, la deuxième étape est d'appliquer le modèle à un jeu inconnu et de valider (ou pas!) la qualité de prédiction. Afin d'obtenir ces deux jeux, le jeu initial est coupé en deux : un jeu d'étalonnage (xcal,ycal) contenant les 100 premières observations ; et un jeu (xtest,ytest) contenant les 40 dernières observations.

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

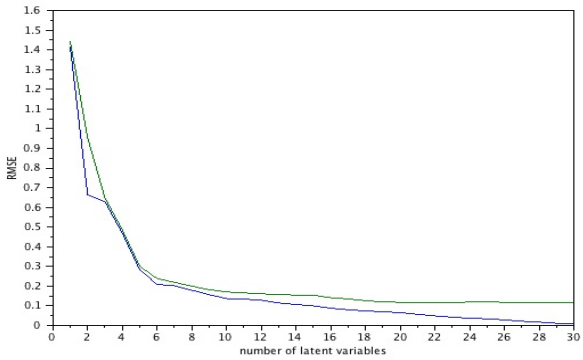
La PLS est appelée par la commande pls pour un calcul avec l'algorithme standard, ou ikpls pour un calcul avec l'algorithme improved-kernel pls. Par exemple :

```
--> 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
```

La PLSR a été calculée avec une validation croisée de 10 blocs consécutifs et 30 variables latentes (VL). On remarque que le centrage est l'option par défaut. Le champ model.err.d contient deux vecteurs: l'erreur d'étalonnage (RMSEC) et l'erreur de validation croisée (RMSECV). Ces courbes sont éditées avec curves:

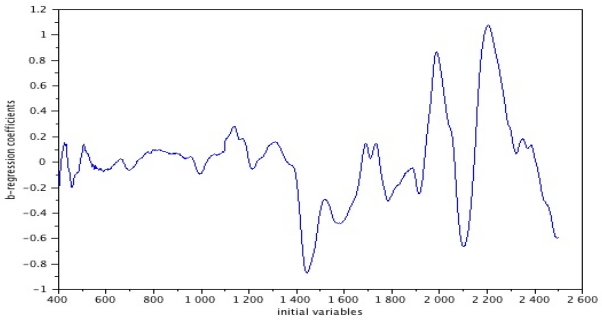
```
--> curves(model.err);
```

Le RMSEC et le RMSECV sont imprimés en bleu et vert respectivement. D'après le graphique précédent, nous choisissons de construire le modèle avec 6 VL.



Les b-coefficients du modèle à 6 VL sont aussi visualisés:

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



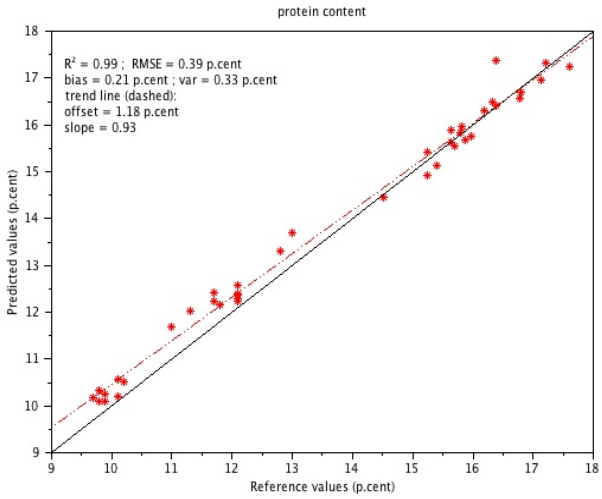
Le modèle est ensuite appliqué sur le jeu test avec la commande regapply qui est commune à tous les modèles issus de régressions :

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

L'erreur standard de prédiction RMSEP (*root mean square error of prediction*) peut aussi être visualisé comme le RMSECV.
A noter que tous les modèles calculés par la méthode de régression sont évalués simultanément. Ainsi pred.ypred.d est une matrice de dimensions (40 x 30), 40 observations et 1 à 30 variables latentes.

La fonction regplot permet de comparer les prédictions du modèle à 6 variables latentes avec les valeurs de référence :

```
--> y6=pred.ypredtest(:,6);  // sélection de la 6° variable de ypredtest
--> h=regplot(ytest,y6,'r*','t','p.cent','protein content') ;
```



Les options facultatives choisies pour regplot dans cet exemple sont:

'r*' : représentation des points avec une étoile rouge;
't': représentation de la courbe de tendance;
'p.cent': unité de mesure
'protein content': titre de la figure.

Pour mémoire, biais et var sont respectivement les moyennes et variances des résidus (différences) entre valeurs prédites et valeurs mesurées ; et $RMSE^2 = biais^2 + var^2$.

3.2. Prétraitements.

Les prétraitements sont destinés à enlever une information spectrale qui est nuisible pour la construction de modèles d'étalonnage.

Certains prétraitements sont très utilisés en proche infra-rouge où les déformations spectrales sont en général importantes :
snv normalise les spectres ; très utile en cas d'un effet multiplicatif;
detrending corrige les déformations simples de ligne de base, dont le décalage vertical ou l'apparition d'une pente.

Ces méthodes sont très simples à utiliser, voir leur syntaxe dans l'aide les concernant. Mais d'autres prétraitements sont les méthodes dites de projection orthogonale, plus complexes à mettre en œuvre car elles nécessitent des données complémentaires. Deux d'entre elles sont détaillées ici : l'*external parameter orthogonalization* (EPO) et *error removal by orthogonal subtraction* (EROS).

Le principe des projections orthogonales.
L'information nuisible est représentée par une matrice **D** obtenue à partir de données généralement issues d'un plan d'expérience. Ensuite les vecteurs-propres d'une ACP condensent l'information de **D**. Le réglage du modèle consiste à déterminer le nombre A de ces vecteurs-propres représentant au mieux l'information nuisible. La correction se fait par projection des spectres orthogonalement aux A premiers vecteurs-propres.

Les données pour effectuer une projection orthogonale.
Les données sont issues d'un plan d'expérience ciblant une grandeur d'influence à éliminer. Prenons l'exemple de la température.
Une possibilité consiste à acquérir des spectres sur les mêmes objets portés à différentes températures (pas nécessairement les mêmes températures pour tous les objets); la méthode EROS peut être appliquée.
Un autre possibilité consiste à acquérir des spectres sur les mêmes objets portés aux mêmes températures ; les méthodes EPO et EROS peuvent être appliquées.

L'exemple choisi permet d'illustrer à la fois l'EPO et EROS. Les données sont dans le fichier epo_apples.dat. Ce fichier contient les champs suivants :

```
-->apples
apples  =

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

Les jeux d'étalonnage et de test sont (xcal,ycal) et (xtest,ytest). Le jeu x1 contient les spectres de 10 pommes acquis à 8 niveaux de température; x1.i identifie simultanément les pommes (1° caractère, lettres A à H) et les températures (2° à 3° caractère, 5/10/15/20/25/30/35/40°C). Par la suite ces données sont supposées avoir été extraites et pouvoir être par leurs noms respectifs : x1,x1_obs,x1_temp, xcal, ycal, xtest et ytest. Cela simplifie l'écriture: il est plus rapide de taper x1 que apples.x1. Par exemple:

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

Nous aurons besoin par la suite d'utiliser les codes d'échantillon c_ech (ici: les pommes) et les codes de grandeur d'influence c_gi (ici: les températures). Ils sont obtenus grâce à str2conj qui va extraire des identifiants de groupes dans des chaines de caractères:

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

Calcul de l'external parameter orthogonalization (EPO)
L'EPO est appellée par la commande epo selon l'exemple ci-dessous :

```
-->[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

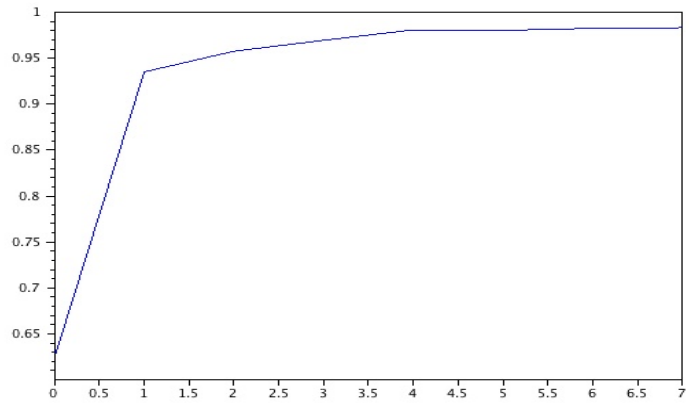
Les nombres 10 et 8 indiquent une validation croisée avec 10 groupes et une régression PLS avec maximum 8 variables latentes.

On constate que c_ech (le code des objets-supports des observations) est présent dans les arguments d'entrée. Il n'est pas utile au calcul de l'EPO elle-même, par contre il est nécessaire pour le calcul du lambda de Wilks.

Les champs d_matrix et eigenvec donnent la matrice de l'information nuisible et ses premières composantes. La dimension de la correction est fixée à l'aide des aides : les valeurs propres dans ev_pcent, le lambda de Wilks dans wilks et le RMSECV dans rmsecv. Par exemple :

-->curves(res_epo.wilks)

donne le graphe suivant. L'échelle est entre 0 et 7 pour rappeler que le premier modèle ne comprend pas de correction par EPO.
Le lambda de Wilks est un critère de discrimination entre groupes: 0 = groupes confondus, 1 = groupes bien séparés. Les corrections par projection orthogonale doivent augmenter le lambda de Wilks. Effectivement il augmente jusqu'à 7 composantes principales mais beaucoup plus faiblement à partir de 5. La dimension retenue pour le modèle peut être fixée à 5.

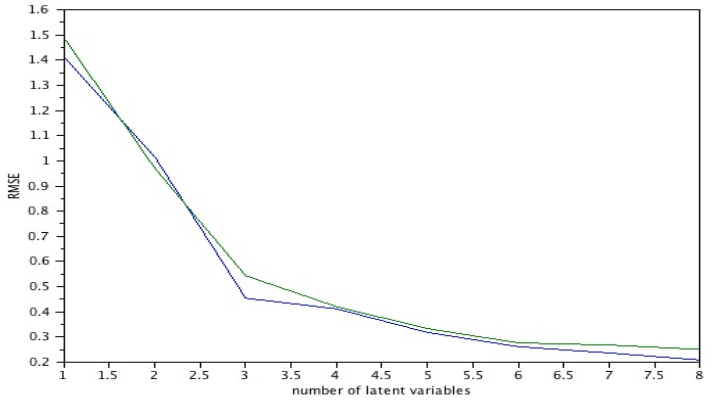


Le champ pls_models est une liste de structures et contient tous les modèles PLS de 1 à 8 variables latentes. Chaque élément de la liste est associé à une correction par projection orthogonale de 0 à (8-1 =7) composantes principales :

pls_models(1)	pas de correction par projection orthogonale
pls_models(2)	EPO avec 1 composante
…	...
pls_models(8)	EPO avec 7 composantes

Une EPO corrigent de 7 dimensions correspond au 8° et dernier modèle, pour lequel les valeurs de RMSECV et du RMSECV sont visualisées sur un graphe :

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



Le RMSECV de la PLSR diminue avec l'augmentation du nombre de variables latentes, et suggère 6 variables latentes.

Calcul de l'error removal by orthogonal subtraction (EROS) :

EROS est appelé par la commande eros selon l'exemple suivant :

```
-->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 est basée sur les objets (ex : pommes) et ne prend pas en compte les niveaux de perturbation (ex : températures) auxquels ils sont soumis, donc par rapport à EPO il y a un paramètre de moins à renseigner. Par contre les sorties (res_epo et res_eros) contiennent les mêmes champs.

Application de EPO et EROS :

Les modèles obtenus par EPO et EROS étaient res_epo et res_eros respectivement. Ces modèles sont appliqués au jeu de test (xtest,ytest) avec la commande popapply selon l'exemple suivant :

```
-->res_test=popapply(res_epo,5,xtest,ytest)
 res_test  =
   ypred: div
   rmsep: div
   r2: div
```

Nous avions précédemment retenu 5 dimensions pour l'EPO et 6 variables latentes pour la PLSR. L'édition des erreurs de prédiction montre que ce choix était correct, le RMSEP

correspondant est de 1,3706 et figure parmi les plus basses valeurs, proche de l'optimum (5VL, RMSEP=1,2811).

Les valeurs prédites pour 6 variables latentes sont à la 6° colonne de res_test.ypred.d.

Autre option :
Les fonctions pop_dextract puis pop_dtune permettent de réaliser aussi les prétraitements par projection orthogonale (POP). Cela est particulièrement indiqué quand la correction porte sur au moins deux grandeurs d'influence représentées par des données différentes.
Supposons que dans l'exemple précédent nous voulons aussi appliquer la correction Detrend qui est aussi une projection orthogonale. L'EPO ou EROS peuvent être réalisés au choix par les fonctions epo, eros ou pop_dextract et produisent **D** la matrice d'information nuisible, dans le champ dmatrix. Detrend est réalisée par la fonction detrending qui produit une matrice de Vandermonde notée **L**. Les deux matrices **L** et **D** sont concaténées puis pop_dtune donne le modèle global de correction par projection orthogonale.

3.2 Discrimination

Le principe des méthodes de discrimination.
Les méthode de discrimination actuellement disponibles dans Fact sont l'analyse factorielle discriminante (FDA) l'analyse discriminante PLS (PLS-DA) et l'analyse discriminante pas à pas. Chacune de ces méthodes a son algorithme propre pour déterminer les directions de l'espace (associées à une métrique) qui discriminent au mieux les différents groupes. Ensuite les scores ou coordonnées des observations dans ces espaces sont obtenus. Les probabilités d'appartenance d'un échantillon à un groupe sont définies à partir de leur distance à chaque groupe évaluée avec une distance Euclidienne usuelle ou une distance de Mahalanobis (par défaut). L'échantillon est attribué au groupe dont il est le plus proche, sous réserve que cette distance ne soit pas inférieure à un seuil donné. Les matrices de confusion représentent le nombre d'observations attribuées à chaque classe (en ligne) comparé à l'appartenance réelle des observations (en colonne). La diagonale représente les bons classements, leur nombre est comparé au nombre total d'échantillons. Les calculs sont fait en étalonnage avec le jeu de données complet, ou avec validation croisée.

Les données pour réaliser une discrimination.
Les méthodes de discrimination demandent une matrice de données explicatives p.ex. **X** et un vecteur p.ex. gr désignant les groupes. Si **X** est de dimensions (*n x q)*, gr est un vecteur de dimension (n x 1). Ce vecteur contient des nombres entiers, qui peuvent prendre des valeurs allant de 1 à maxgroup, où maxgroup est le nombre de groupes. Il est nécessaire que chaque groupe soit représenté au moins une fois dans la collection d'étalonnage.
Lorsque les descripteurs des individus sont des clés pouvant désigner les groupes, il est très facile de créer le vecteur gr en exploitant ces noms. Ainsi dans l'exemple « *huiles d'olive* » donné au paragraphe 2.1.3, les régions de récolte des olives sont désignées par les deux premières lettres. Un identifiant de groupe de spectres pour chaque région de récolte des olives est obtenu avec le commande str2conj:

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

gr est un vecteur contenant des nombres de 1 à 9 pour représenter chacun des 9 groupes.
labels_regions et nbr_obs donnent respectivement les codes des groupes/régions et le nombre d'observations par groupe:

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

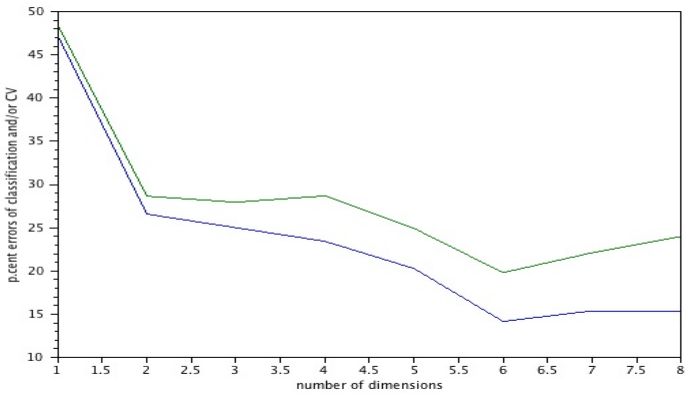
L'analyse factorielle discriminante (*FDA : factorial discriminant analysis*)
L'analyse factorielle discriminante est adaptée aux données possédant des variables fortement corrélées, comme les chromatogrammes ou les spectres. D'une manière similaire à la régression en composantes principales, elle procède en deux étapes : ACP sur les données, suivi d'une analyse discriminante linéaire.

L'analyse discriminante est appellée avec la commande 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
```

Les pourcentages d'erreur en étalonnage et validation croisée, valeurs entre 0 et 100, sont dans le champ res_fda.err et peuvent être visualisés:

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



A noter que res_fda.conf_cal_nobs, res_fda.conf_cal_nobs et res_fda.conf_cal_cv sont des listes contenant ici 8 structures Div, une pour chaque dimensions. Si on retient 7 dimensions comme suggéré par la figure précédente, la matrice de confusion pour 7 dimensions est :

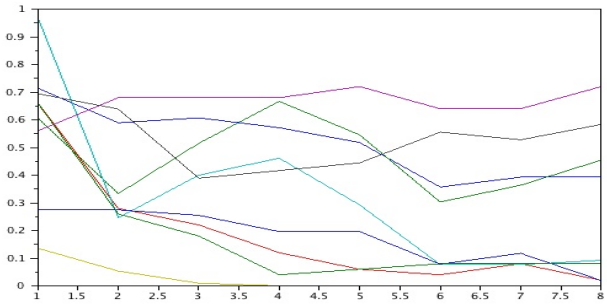
```
-->cm7=res_fda.conf_cal_nobs(7);
-->cm7.d
 cm  =

   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.
```

Toutes les 206 observations du groupe 6 sont bien classées dans le groupe 6 ; mais des observations attribuées au groupe 6 appartiennent à d'autres groupes, p.ex. 17 dans le groupe 1.

Il est possible d'avoir un détail des erreurs de classement pour chacune des classes et chaque nombre de dimensions avec res.errbycl_cal et res.errbycl_cv :

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



L'analyse discriminante PLS (*PLS-DA PLS-discriminant analysis*)
La méthode PLS-DA consiste tout d'abord à établir un modèle linéaire PLS2 sur l'identifiant des groupes toujours représenté par une matrice disjonctive. Les sorties sont exactement les mêmes que celles de FDA. Pour comparaison, nous ne présentons que la séquence d'appel de plsda et la matrice de confusion pour 7 dimensions (ou variables latentes pour 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.
```

L'analyse discriminante pas à pas (*forward discrimimant analysis*):
Les variables sont ajoutées une après l'autre dans le modèle par la fonction forwda:

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

Contrairement aux autres méthodes, forwda applique un seuil pour incorposer une nouvelle variable discriminante. Dans notre exemple, le maximum est de 6 variables et d'après la validation croisée le meilleur modèle est obtenu avec 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.
```

Pour toutes les méthodes d'analyse discriminante:
Seulement pour la validation croisée, les matrices de confusion en validation croisée représentent des pourcentages, non un classement de chaque observtion comme en étalonnage. La raison est qu'il peut arriver que tous les éléments d'une même classe soient dans le jeu d'étalonnage, ou bien dans le jeu de validation: ils ne peuvent alors pas être

classés. Pour éviter ce problème la validation croisée est répétée 10 fois d'où l'utilisation de pourcentages.

3.3 Analyses multi-tableaux.

Une sélection de méthodes multitableaux est proposée: ccswa , comdim, statis

Ces méthodes sont opérationnelles et disposent d'une aide. Toutefois il est prévu que leurs sorties soient harmonisées. Une démo de ces fonctions sera alors disponible.

4 Traitement univarié: ANOVA+SNK

Fact n'est pas orienté statistiques mais malgré tout dispose d'une analyse de variance à 1 facteur qui traite simultanément mais de manière indépendante plusieurs variables. Elle est complétée par un test de classement de moyennes: Student-Newman-Keuls. Fact n'étant pas une BAO de statistiques, les sorties sont réduites à l'essentiel de l'information utile en chimiométrie.

Prenons comme exemple le fichier pph.dat. Après chargement dans la console:

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

Les 5 variables sont des tanins: les dimères B1, B2, B3, B4 et le monomère epicatéchine.
Les 21 observations sont représentées dans pph.i par leur identifiant de classe, cl1 à cl7: il y a donc 7 classes de 3 répétitions chacune.

L'analyse de variance est appelée par la commande snk:

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

Le premier argument pph est le fichier à analyser, le deuxième argument pph.i est l'identifiant des classes. La sortie res contient deux champs: res.anova pour les résultats de l'ANOVA et res.snk pour les résultats de 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 et SCER sont les sommes de carrés expliquées respectivement par le facteur étudié et le résidu. F est la valeur de Fischer calculée, et Pr>F est la probabilité qu'il n'y ait pas de différence entre les moyennes.
L'ANOVA prend en compte des nombres d'observations différents selon les classes; seul est reporté le nombre moyen.

res.snk est une liste; res.snk(i) contient les résultats de SNK pour la variable i. Par exemple pour la 5° variable= 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 est concu et programmé pour des classes ayant toutes le même nombre d'observations (nobs). Si ce n'est pas le cas, le calcul est fait avec le nombre moyen d'observations par classe arrondi à l'entier le plus proche; mais plus les nombres d'observations seront différents et plus le risque d'erreur sera élevé.

La dernière colonne donne le regroupement des moyennes non significativement différentes avec 5% d'erreur, elles sont reliées verticalement par la même lettre. Sur cet exemple cl6 et cl7 sont différentes des 5 autres classes; cl2 est différente de cl1 et cl4 mais pas de cl3 ou cl5.

5 Autres informations utiles

5.1. Informations complètes sur la boite à outils

Fact est téléchargeable directement depuis le module Atoms de Scilab.

Le présente fichier d'aide ainsi les jeux de données qui l'illustrent sont disponibles sur la page Fact de Scilab:

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

5.2. Et pour Matlab?

Les versions Saisir sous Matlab et Fact sous Scilab sont différentes. Il n'y a pas pour l'instant de version Fact sous Matlab. La version Saisir 1.0 sous Matlab peut être téléchargée directement à l'adresse suivante :

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

5.3. Droits et devoirs des utilisateurs.

Fact a été protégé en France auprès de l'APP. Toutefois il est distribué librement sous licence Cecill-C et à ce titre il est rappellé qu'il n'est assorti d'aucune garantie.

Nous serions reconnaissants aux utilisanteurs communiquant sur des résultats obtenus avec Fact de le mentionner.

La correspondance concernant Fact est à adresser à:

JC Boulet
bouletjc@supagro.inra.fr