

SAISIR
UN ENVIRONNEMENT POUR L'EXPERIMENTATION
EN CHIMIOMETRIE
SOUS MATLAB[®]

MODE D'EMPLOI

Package CD-Rom de SAISIR©

Révision Octave, Scilab

Dominique Bertrand
INRAe Nantes
dataframe@free.fr

Révision Matlab

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

Révision 2024

Avis à l'utilisateur

- Il n'est pas possible de démarrer l'utilisation des fonctions Saisir, sans lire ce document jusqu'au paragraphe 3 (ce n'est pas très long !).

- L'utilisation de l'environnement Saisir nécessite de connaître un peu l'environnement Matlab.

- Toutes les fonctions ont une petite aide que l'on peut obtenir par la commande help <fonction>.

La liste des fonctions disponibles est donnée par la commande help saisir

- Si le résultat donné par une fonction n'est pas clair, on peut toujours taper le nom de l'argument de sortie, puis les champs de ce résultat pour savoir la nature des paramètres de sortie.

Par exemple :

```
>>p=pca(ble) ; %effectue une ACP sur le fichier ble
```

Mais que contient donc p ?

```
>>p
```

```
score: [1x1 struct]
eigenvec: [1x1 struct]
eigenval: [1x1 struct]
average: [1x1 struct]
info: 'PCA 08-Jan-2024 16:06:46'
```

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

Bon courage !

SAISIR
UN ENVIRONNEMENT POUR L'EXPERIMENTATION
EN CHIMIOMETRIE
SOUS MATLAB[®]
MODE D'EMPLOI

Table des matières

Introduction

1. Principe général

2. Utilisation de l'environnement Saisir

3 Traitements statistiques de base

4 Dictionnaire des fonctions disponibles

5 Liste thématique des fonctions disponibles

6 Recommandations au développeur

Annexe A : module image

(module spécifique)

Annexe B : Module olfactogramme

(module spécifique)

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.

Afin de disposer d'un outil de manipulation de données, adapté à cette situation (tableaux comportant plusieurs milliers de lignes ou de colonnes), nous avons développé un ensemble de procédures, sous l'environnement MATLAB, susceptible de faciliter le travail des expérimentateurs, et de permettre aisément l'extension du système.

Cet environnement a été créé initialement pour manipuler des données spectrales et est désigné par le sigle « SAISIR » (*Statistiques Appliquées à l'Exploitation des Spectres Infrarouge*). Il comprend des fonctions de chargement et sauvegarde des données, de manipulation, des outils graphiques et des procédures statistiques : analyse multidimensionnelles, discrimination, régression, analyse de variance ...

L'utilisation des procédures de SAISIR suppose une certaine connaissance de l'environnement et de la programmation sous MATLAB.

1. Principe général

L'environnement SAISIR repose presque exclusivement sur la manipulation de tableaux de données, sous la forme de matrices de nombres réels. Le principe fondamental, respecté dans toutes les procédures, et que les colonnes et les lignes des matrices de données comportent des identificateurs, qui « suivent » l'ensemble des manipulations. Examinons, par exemple, le fichier de données suivant (Note 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

(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 conserver complètement le tableau, il est nécessaire de garder ces trois informations. Sous MATLAB, le format SAISIR pour conserver un tableau de données est *une structure* avec les champs « i » pour les identificateurs des individus (lignes), « v » pour les variables (« colonnes ») et « d » pour les données. Par exemple, si le fichier de données est conservé dans la structure « POMME », on aura :

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

On peut, bien sûr, extraire chaque champ en utilisant la commande MATLAB « . » d'extraction, par exemple :

```
>> POMME.i
GALA1
FUJI1
FUJI2
```

Il est donc toujours possible de « revenir » aux matrices simples de MATLAB, par exemple :

```
>> 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 » de la structure SAISIR sont des matrices de caractères et non des cellules (*cell array*).

Un cas particulièrement important concerne les données qui sont sous la forme de courbes numérisées (spectres, chromatogrammes, électrophorègrammes ...). 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 identificateurs d'individus.

« 1100 », « 1102 », « 1104 », « 1106 », « 1108 » sont des chaînes de caractères représentant les identificateurs des variables. Supposons que la structure SAISIR de ces données s'appelle spectre, on a:

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

et

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

La nature numérique des descripteurs de variables est exploitée dans les représentations de courbe (fonctions `curve`, `curves`, `tcurve`, `tcurves`).

2. Utilisation de l'environnement Saisir

On peut avoir une liste des commandes de SAISIR par l'ordre

```
>>help saisir
```

Pour chaque commande, l'ordre `help` « commande » donne un (court !) texte explicatif de l'utilisation de la fonction.

2.1 Prise en main (*Getting started*)

Pour pouvoir travailler dans l'environnement SAISIR, il faut disposer de données sous la forme de tableaux rectangulaires (matrices). Dans son état actuel, l'environnement ne gère pas les données manquantes dans les procédures numériques.

2.1.1 Chargement, création et sauvegarde de fichier SAISIR

Création à partir de matrices déjà sous MATLAB

A partir de matrices déjà chargées dans l'environnement MATLAB, on peut « manuellement » créer une structure SAISIR. Si on ne dispose que de la matrice de valeurs numériques, on peut utiliser l'ordre `matrix2saisir`. Le signe « 2 » est dans la logique (américaine) de MATLAB, et veut dire « to », c'est à dire « transformé en ».

Par exemple :

```
>>a=rand(5,2);% commande MATLAB : matrice de nombres aléatoires 5x2
```

```
>>b=matrix2saisir(a);
```

b =

```
    d: [5x2 double]  
    i: [5x1 char]  
    v: [2x1 char]
```

Dans ce cas, les identificateurs des lignes et des colonnes sont simplement des chaînes de caractères représentant le numéro de la ligne ou de la colonne.

Ainsi

```
>> b.i
```

```
1
2
3
4
5
```

On peut ajouter un préfixe sur les lignes et les colonnes, par exemple :

```
>> b=matrix2saisir(a,'ligne','col')
```

```
b =
```

```
  d: [5x2 double]
```

```
  i: [5x6 char]
```

```
  v: [2x4 char]
```

On a alors, par exemple :

```
>> b.v
```

```
col1
```

```
col2
```

Si l'utilisateur dispose d'identificateurs d'individus ou de variables, il peut toujours créer « complètement manuellement » la structure SAISIR.

Par exemple :

```
>>identificateur={'objet' 'chose' 'concept' 'element' 'entité'}
```

```
>>b.i=char(identificateur) ;
```

```
>> b.i
```

```
objet
```

```
chose
```

```
concept
```

```
element
```

```
entité
```

Il faut noter la conversion de *array of cells* concrétisé par l'utilisation des accolades { et } en matrice de caractères par l'ordre MATLAB char .

Chargement à partir de fichiers EXCEL

Il est possible de charger des fichiers Excel possédant une forme simple (imposée par SAISIR).

Le seul format admis par SAISIR est par exemple le suivant :

	OGLO	OTER	OCAV	
GALA1	2.8	1.2	0.3	
FUJ11	2.6	0.5	0.4	
FUJ12	7.5	0.3	0	

La première ligne contient, dans chaque cellule, l'identificateur de la colonne.

La première colonne du fichier Excel contient les identificateurs des lignes.

Les autres cellules contiennent des *valeurs numériques*, avec le *séparateur décimal* « . » et non « , ».

La cellule en position (1,1) n'est pas conservée par SAISIR.

Le fichier Excel doit être préalablement sauvegardé sous le format Excel (ASCII) « .CSV ». Dans le menu Excel, cliquez sur **enregistrer sous puis CSV (Séparateur point virgule)**.

Le chargement sous MATLAB s'effectue par l'ordre :

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

`filename` est une chaîne de caractères indiquant le fichier à charger (depuis le répertoire de travail). `res` est la structure SAISIR dans MATLAB.

Par exemple :

```
essai=excel2saisir('donnees') charge le fichier Excel donnees.csv et le place dans la structure SAISIR essai.
```

Le fichier Excel contient en première colonne les identificateurs des lignes, qui forment les éléments du champ `res.i`. Ainsi la première colonne de `res.d`, soit `res.d(:,1)` correspond à la deuxième colonne du fichier Excel ;

La fonction `excel2saisir` supprime entièrement les colonnes ou les lignes qui ne contiennent que des cellules vides. Sinon, les cellules Excel vides ou celles contenant des données non numériques sont remplacées par la valeur **NaN** (*not a number*).

On peut créer un fichier sans données manquantes (en perdant soit des lignes, soit des colonnes du fichier de départ) par l'ordre `eliminate_nan1`.

Exemples :

```
reduit = eliminate_nan1(tableau,0) ;% maximum de lignes, au dépend des colonnes
```

```
reduit = eliminate_nan1(tableau,1) ;% maximum de colonnes, au dépend des lignes
```

```
reduit = eliminate_nan1(tableau) ; % tentative d'optimisation : plus faible perte de données.
```

La commande `eliminate_nan1` avec un seul argument tente d'éliminer des lignes ou des colonnes en perdant le moins possible de données.

Sauvegarde de fichiers SAISIR

L'ordre `saisir2excel` permet de sauvegarder les données dans un format lisible par Excel.

Syntaxe :

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

Exemple de commande :

```
saisir2excel(tableau,'essai')
```

Sauvegarde la structure SAISIR `tableau` sous le nom `essai.CSV`

Attention ! le premier argument est une variable de MATLAB désignant une structure SAISIR, tandis que le deuxième est une chaîne de caractère (d'où la nécessité des signes apostrophes « ' »).

Le fichier est sauvegardé avec ses identificateurs de lignes et de colonnes, au format. CSV de MATLAB (séparateur : point-virgule).

Si le nombre de colonnes dépasse 255, Excel ne sait plus lire la totalité du fichier.

Lorsque le nombre de lignes est inférieur au nombre de colonnes, on peut éventuellement transposer la structure SAISIR avant la sauvegarde, par l'ordre `SAISIR_transpose`.

2.1.2 Manipulation élémentaire des données

Nota : dans la suite, nous désignerons les structures SAISIR sous le nom « matrices ».

Les manipulations élémentaires concernent la concaténation, la suppression de ligne et de colonne dans des matrices. Dans le nom des fonctions, les lignes sont désignées par **row** et les colonnes par **col**.

La concaténation est désignée par **append**, et la déléition par **delete**

On a ainsi les commandes :

appendrow, **deleterow**
appendcol et **deletecol**

Exemple de concaténation

Soit deux structures matrice *ayant le même nombre de colonnes*.

L'ordre **saisir3=appendrow(saisir1, saisir2)**

concatène les matrices **saisir1** et **saisir2** en augmentant le nombre de lignes.

Par exemple, soit **tableau1** avec 10 lignes et 25 colonnes, et **tableau2** avec 30 lignes et 25 colonnes. L'ordre :

complet=appendrow(tableau1, tableau2)

construit une matrice appelée **complet** représentant un seul tableau de 40 lignes et 25 colonnes. Les noms des variables et des individus sont évidemment correctement recopiés dans **complet.v** et **complet.i**.

Il est possible de concaténer un nombre arbitraire de matrices par l'ordre **appendrow1**. Sa syntaxe est (par exemple) :

saisir=appendrow1(saisir1, saisir2, saisir3, saisir4).

appendcol et **appendcol1** fonctionnent de la même manière, selon les colonnes. Dans ce cas, le nombre de lignes des matrices à concaténer doivent être identique.

Par exemple, si **A** représente une matrice de 25 lignes et 3 colonnes, et **B** une matrice de 25 lignes et 12 colonnes, l'ordre

C=appendcol(A,B)

construit une matrice **C** de 25 lignes et 12 colonnes.

Suppression de lignes et de colonnes

On dispose des ordres:

deleterow, **deletecol**, **selectrow** et **selectcol**.

Exemple d'utilisation :

Soit **A** une matrice de 15 lignes et 20 colonnes.

L'ordre

C=deletecol(A,1)

construit une matrice **C** à partir de **A** avec suppression de la première colonne de **A**.

Ainsi **C** possède 15 lignes et 19 colonnes.

C'est le même principe pour les lignes, avec **deleterow**.

On peut également désigner les colonnes que l'on veut garder. Par exemple, l'ordre

C=selectcol(A,1 :2 :8)

Sélectionne les colonnes 1, 3, 5, 7 de **A** et les place dans **C**. C'est le même principe avec les lignes (ordre **selectrow**).

Attention ! Il faut bien noter que dans le cas des données de type spectral ou temporel (courbes numérisées), les variables sont des mesures à des temps ou des longueurs d'onde.

Cependant, il n'y a en général pas de correspondance immédiate entre l'identificateur de la variable (chaîne de caractère représentant un nombre) et son indice (rang).

Par exemple, dans un fichier de spectres proche infrarouge dont le début est :

	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

La variable d'indice 1 représente la longueur d'onde « 1100 » .

La variable d'indice 4 représente la longueur d'onde « 1106 » .

Les ordres précédents s'appliquent sur les indices, et non sur les descripteurs !

Ainsi, pour sélectionner la colonne correspondant à la longueur d'onde « 1104 », il faut écrire, par exemple :

`lo1104=selectcol(spectre, 3)` ; et non `lo1104=selectcol(spectre, 1104)`

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

`index = seekstring (spectre.v, '1104')`

renvoie les numéros d'ordre des variables qui portent le nom « 1104 ».

On peut faire de même

`index = seekstring (spectre.i, 'golden')`

Renvoie les numéros d'ordre de tous les individus qui ont la chaîne « golden » dans leur nom.

Les minuscules et les majuscules sont considérées comme différentes.

Si on dispose de données de type "spectral", on peut utiliser la commande :

`index=find_index(spectre.v,1104);`

L'avantage de cette commande est qu'il n'y a pas la nécessité d'avoir une correspondance exacte des caractères. Par exemple, cette commande trouvera l'index approprié même si la variable « 1104 » est en fait codée par la chaîne « 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*), cultivés dans deux lieux de culture (*Paris* et *Montpellier*). On effectue 20 répétitions par essais.

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 lorsqu'on dépasse 9 répétitions !

De même, on aura TAMO19 : *Talent*, cultivé à *Montpellier*, répétition 19.

Il faut, dans l'exemple, surtout éviter de changer de dimensions des champs de caractères. Par exemple, CRMO12 et ARMPA1 ne sont pas compatibles, car le code CR comprend 2 caractères, tandis que le code ARM en comprend 3 !

On peut utiliser les codes des identificateurs pour extraire des sous-fichiers avec l'ordre `select_from_identifier`, qui a la syntaxe suivante :

```
[saisir1] = select_from_identifier(saisir,startpos,str)
```

Où `startpos` (position de départ) indique l'emplacement dans la chaîne de caractère à partir de laquelle la chaîne `str` (*string*) est recherchée. Par exemple l'ordre :

`Paris=select_from_identifier(ble, 3, 'PA')` cherche dans la matrice `ble` les identificateurs de lignes qui possède la chaîne PA en 3^{ème} et 4^{ème} position. Après cet ordre, la matrice `Paris` contient les seuls échantillons dont l'identificateur de ligne contient cette chaîne.

De la même manière, dans l'exemple :

`talent=select_from_identifier(ble, 1, 'TA')` recherche les individus de type TA et les place dans la matrice `talent`.

2.1.3 Deux exemples de traitement complet : l'analyse en composantes principales

L'analyse en composantes principales peut permettre 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 SAISIR.

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 tableau récupéré sous Excel. Les lignes forment les 572 individus. La matrice comprend 9 colonnes correspondant au groupe qualitatif de la région suivi des 8 dosages d'acide gras. On a ainsi une matrice dimensionnée 572 x 9.

Les identificateurs des individus comprennent, deux caractères indiquant la région de culture.

Par exemple :

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

```
>>Ca005
```

Indique que le 5^{ème} individu a été collecté dans la région Ca.

Après avoir mis les données au format Excel (fichier `olive.csv`), la matrice est chargée dans Matlab :

```
olive=excel2saisir('olive')
```

```
>>olive =
```

```
>> v: [9x20 char]
```

```
>> d: [572x9 double]
```

```
>> i: [572x20 char]
```

Pour effectuer une ACP, il faut disposer d'une matrice, *sans données manquantes, ne contenant que les lignes et les colonnes qui font partie de l'analyse (individus et variables actifs)*. Dans le cas particulier de ce fichier excel, la première colonne de valeurs numériques du fichier contient les groupes qualitatifs désignant les régions de culture. Pour faire une ACP, nous devons retirer cette colonne, qui ne sert pas dans cette analyse :

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

Les données possédant des ordres de grandeur différents, il est nécessaire de réduire (ou normer) les colonnes, c'est à dire diviser chacune d'elles par l'écart-type de la variable correspondante. Attention ! Contrairement à ce qui apparaît dans de nombreux logiciels, l'ACP de SAISIR n'est pas systématiquement effectuée sur des données réduites !! Cela est effectué par l'ordre `norm_col`.

```
olive2=norm_col(olive1);
```

On effectue maintenant l'ACP par l'ordre :

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

```
>>res =
```

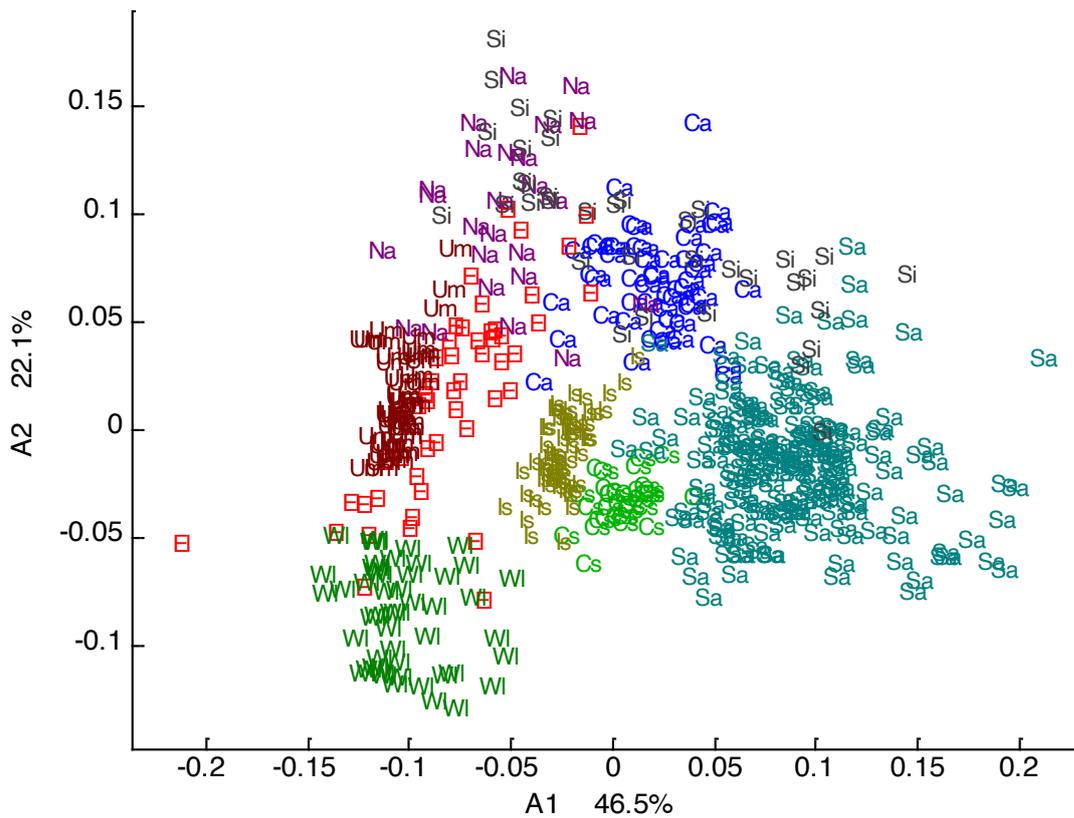
```
>>  score: [1x1 struct]
>>  eigenvec: [1x1 struct]
>>  eigenval: [1x1 struct]
>>  average: [1x1 struct]
      info: 'PCA 06-Dec-2002 08:44:24'
```

Dans l'exemple, `res` est une structure contenant tous les résultats d'une ACP. Chaque champ de cette structure (`score`, `eigenvec`, `eigenval`, `average`) est une structure SAISIR complète. On peut évidemment s'intéresser à chacun des résultats. Le plus important est `res.score` (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 .score` sont donc (logiquement) la recopie des identificateurs de `olive1.i`. Les identificateurs des colonnes de `res .score` sont créés par la commande `pca` elle-même, et sont de la forme :

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

Les premiers caractères représentent le numéro de la composante (« axe »), puis le nombre indique le pourcentage d'inertie de cette composante. Les *cartes factorielles* sont des représentations de paires de colonnes (choisies par l'utilisateur) de `res.score` sous la forme de graphiques X-Y. On peut représenter ces paires par l'ordre `map` et ses variantes.

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

On voit très clairement que les régions de culture ont une influence marquée sur la composition en acide gras des huiles d'olive.

L'ordre `colored_map2` procède exactement de la même manière, mais affiche les identificateurs complets des lignes. Il existe des arguments optionnels à ces commandes par exemple :

```
colored_map2(s,col1,col2,startpos,endpos,labcol1,labcol2,titre,charsize,marg)
```

`labcol1` et `labcol2` : légende des lignes et des colonnes (remplace les légendes par défauts, qui sont les identificateurs des variables)

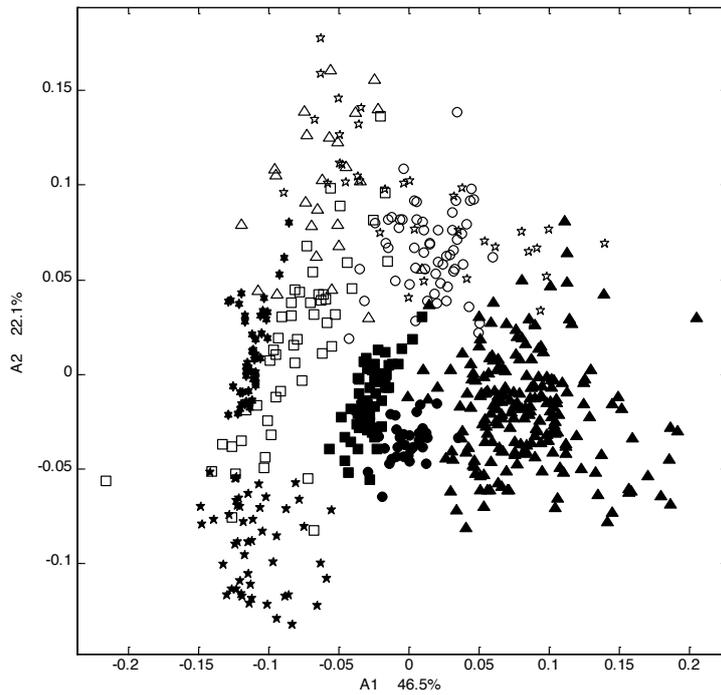
`titre` : titre du graphe

`charsize` : taille des caractères

`marg` : marge éventuelle pour améliorer la représentation des identificateurs longs.

Il est également possible, selon le même principe de représenter les groupes en leur affectant un symbole par l'ordre `symbol_map`. Par exemple :

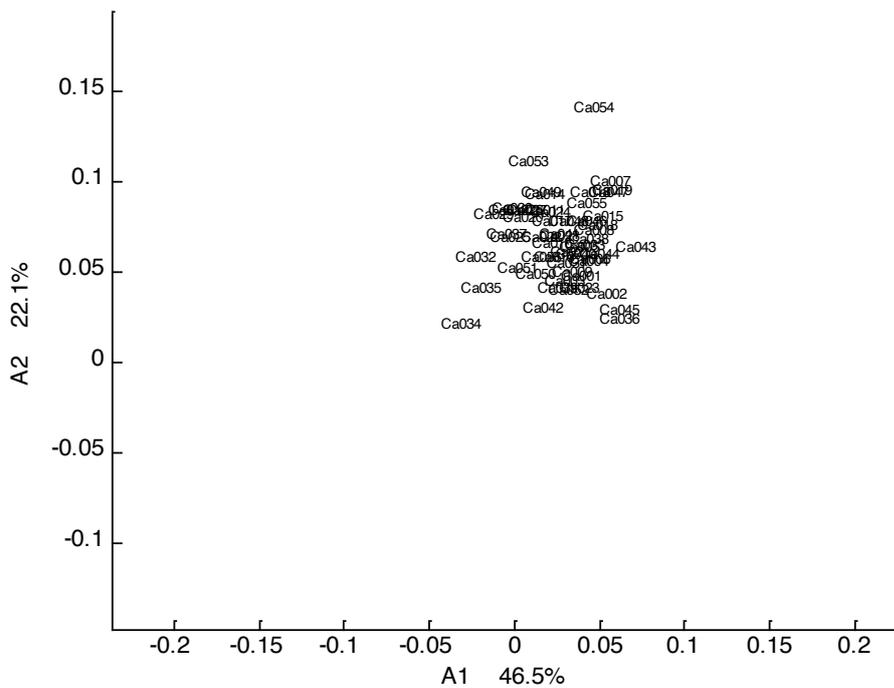
```
symbol_map(res.score,1,2,1,2)
```



Enfin, lorsque de nombreux points sont superposés, il peut être nécessaire de faire des cartes séparées de chaque groupe d'individus. Dans ce cas, on souhaite en général que l'échelle des abscisses et des ordonnées soient celles de la carte principale. L'ordre `submap` permet cette représentation. Par exemple :

```
submap(res.score,1,2,'Ca');
```

représente les seuls individus possédant la chaîne `Ca` dans leurs noms d'identificateurs.

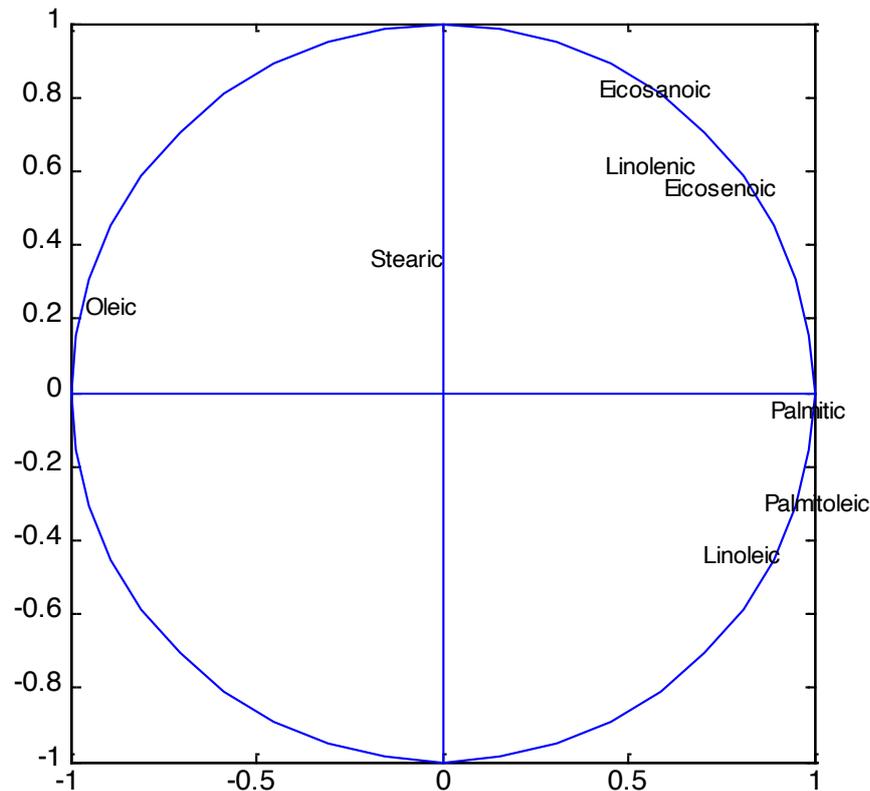


Il est facile d'obtenir le cercle de corrélation par la commande `correlation_plot(score,col1,col2, saisir1,saisir2, ...)`;

Le premier argument de cette fonction est une matrice dont les colonnes sont orthogonales entre-elles. `col1` et `col2` donne les numéros des composantes à représenter. Par exemple, la commande :

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

permet d'obtenir le cercle de corrélation des deux premières composantes.



Deuxième exemple : ACP sur des 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. On a ainsi 1050 mesures par spectre étudié.

La matrice des données spectrales est désignée par `ble`.

`ble =`

`d: [140x1050 double]`

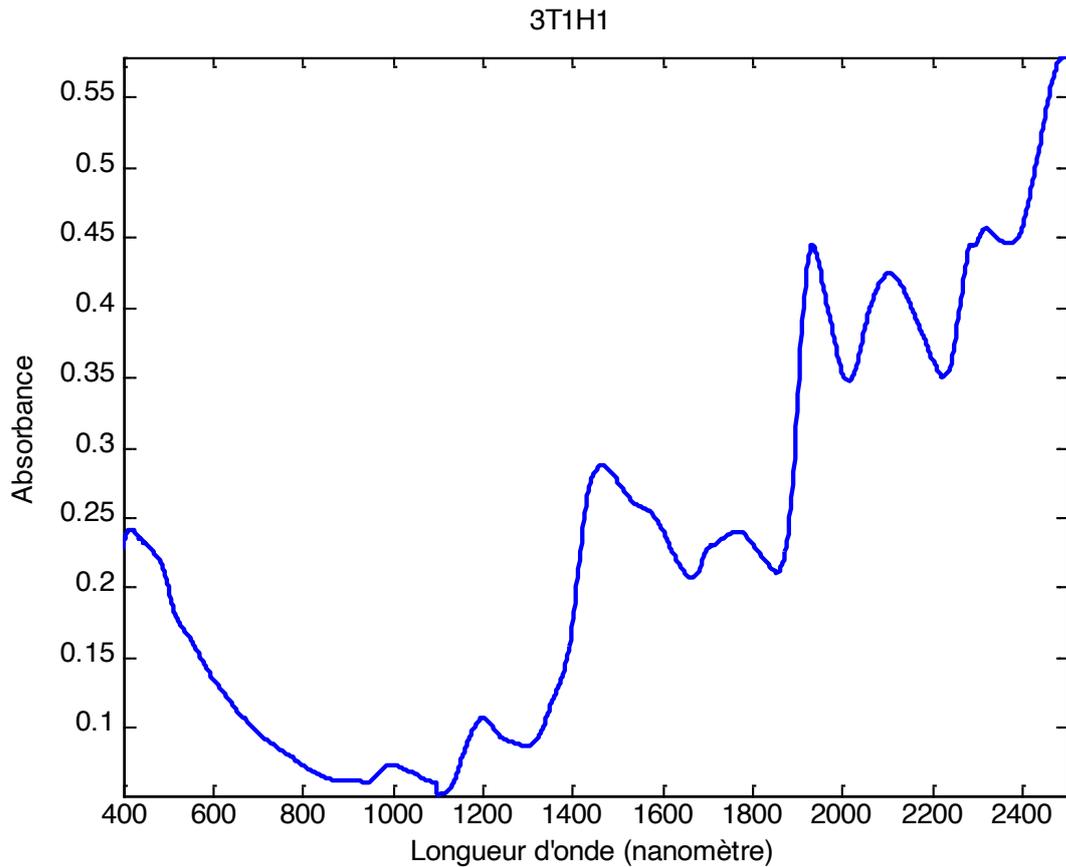
`i: [140x10 char]`

`v: [1050x10 char]`

On peut examiner les données spectrales par l'ordre `curve`, par exemple :

```
curve(ble,3,'Longueur d'onde (nanomètre)','Absorbance')
```

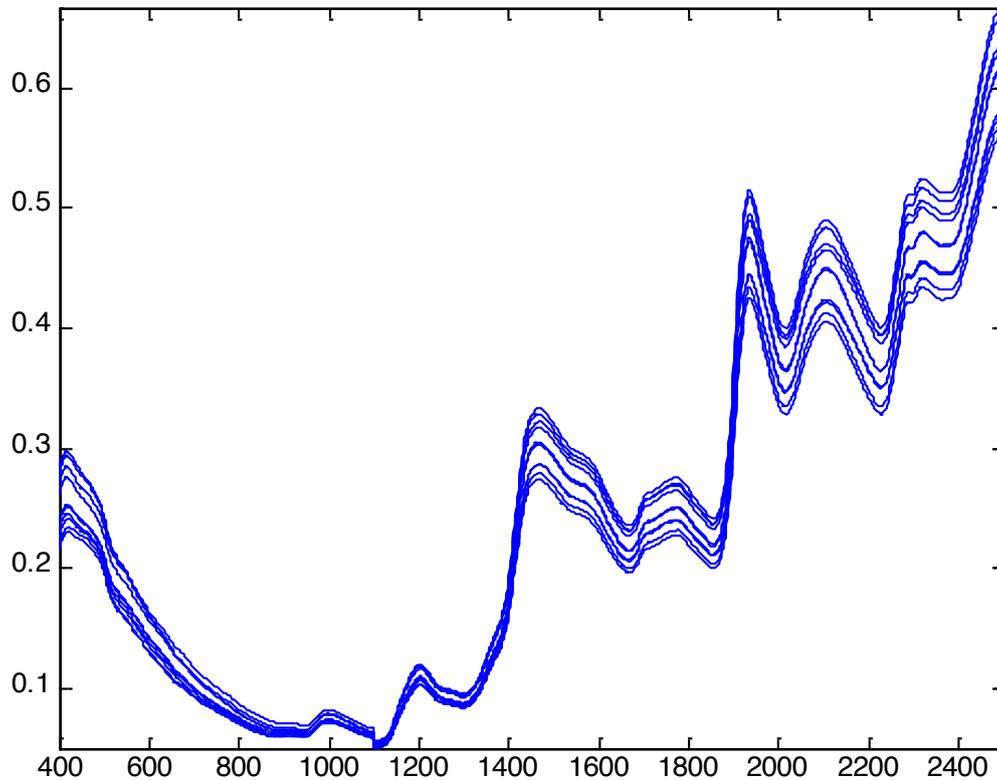
représente le troisième spectre de la collection :



Il faut noter que l'axe des X est correctement gradué selon les longueurs d'onde. L'ordre direct (MATLAB) `plot(ble.d(3,:))` effectue presque le même graphique, mais l'échelle des X est simplement graduée de 1 à 1050 (nombre de points de mesures).

La fonction `curve` 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 courbe sur le même graphique. Par exemple, la commande `curves(ble,1:10)` représente les 10 premiers spectres superposés.



On peut rapidement « feuilleter » l'ensemble des spectres par l'ordre `browse`.

`browse(ble)`

Right button to go down, Left button to go up, Ctrl C to exit

Les courbes défilent dans la fenêtre graphique. Chaque click droit de la souris fait afficher la courbe suivante. Un click gauche permet de « remonter » d'une courbe. On ne peut sortir qu'en pressant les touches [Ctrl C].

Avec l'ordre `curve`, il est toujours possible d'utiliser l'ordre MATLAB `ginput`, qui permet de connaître avec précision les coordonnées d'un point sur le graphe. Cela est utile pour repérer les sommets et les minima des courbes.

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.

Dans l'exemple, on a par exemple, la commande
`acp_ble=pca(ble)`

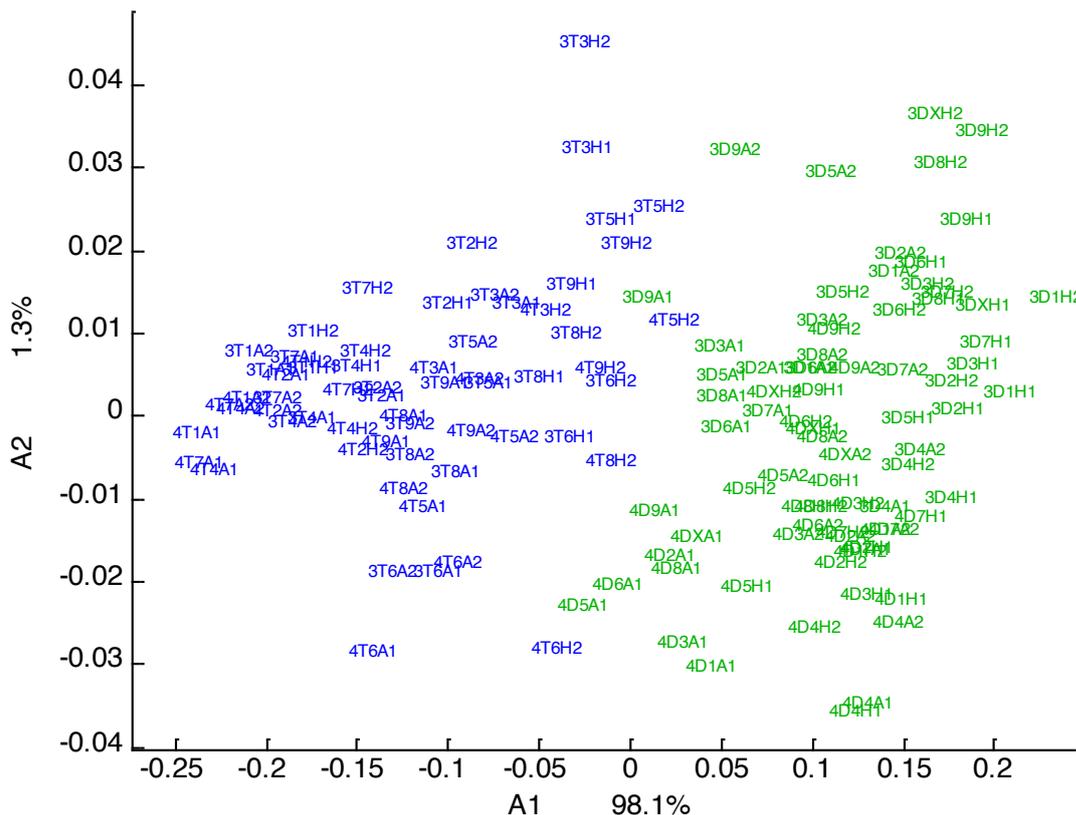
`acp_ble =`

```
score: [1x1 struct]
eigenvec: [1x1 struct]
eigenval: [1x1 struct]
average: [1x1 struct]
info: 'PCA 09-Dec-2002 16:14:57'
```

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

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

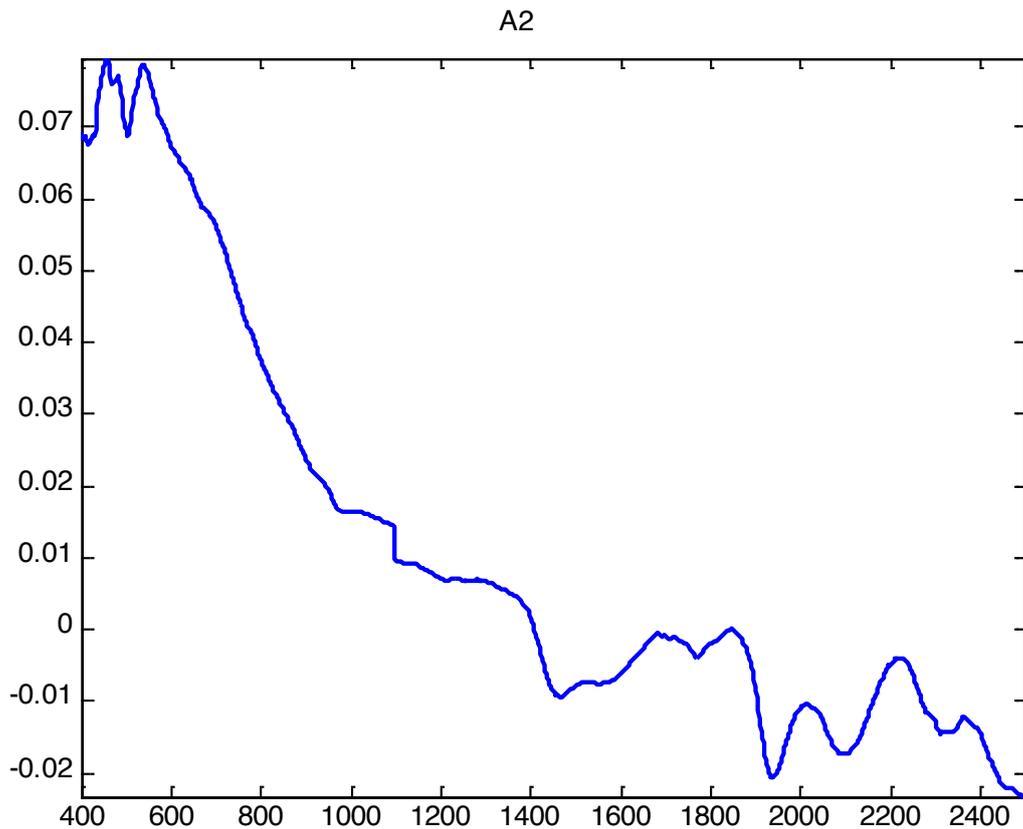
représente le premier plan factoriel, en utilisant les caractères en deuxième position (commençant et finissant en position 2) pour colorier la carte.



La nature *tendre/dure* des blés est très clairement visible.

Lorsqu'on effectue une ACP sur des courbes numérisées, il n'est pas très efficace de représenter le cercle de corrélation. En revanche, les vecteurs propres peuvent s'interpréter comme des « profils » qui indiquent les zones de la courbe qui jouent un rôle important dans la création des composantes. On montre facilement que l'ACP peut être vue comme une méthode permettant de décomposer les courbes comme des sommes des vecteurs propres, pondérés par les coordonnées factorielles correspondantes. Les vecteurs propres (*en colonnes*) sont rassemblés dans le champ `eigenvec` de la structure `pctype`.

On peut les examiner (comme des profils) par la commande `tcurve` (courbe sur matrice transposée). Ainsi `tcurve(acp_ble.eigenvec,2)` représente le deuxième vecteur propre comme un profil.



On voit ici, par exemple que les absorbances dans la région 400-800 nanomètres (lumière visible) jouent un rôle important dans la création de la deuxième composante. Ainsi, selon cette composante, les spectres des blés sont essentiellement séparés selon leur couleur (dans le Visible).

3 Traitements statistiques de base

De manière à montrer la logique des fonctions de SAISIR, les méthodes de régression et de discrimination sont présentées en détail (paragraphe 3.1 et 3.2). Les autres fonctions, sont présentées plus loin dans l'ordre alphabétique (paragraphe 4), et par thématique (paragraphe 5).

3.1 Régressions

Plusieurs méthodes de régression sont disponibles : *Partial least square* (PLS), Régression en composantes principales (PCR), régression *ridge*, régression en variables latentes, régression pas-à-pas.

D'une manière un peu hétérogène, on dispose de nombreux outils permettant la validation croisée. On peut avoir une liste (non exhaustive) des méthodes de régression disponibles par la commande `regression`.

Pour développer une méthode de régression, il faut disposer d'une matrice \mathbf{X} comprenant n lignes et p colonnes (variables prédictives), et d'un vecteur \mathbf{y} comprenant n lignes (variable à prédire) au format SAISIR. Les lignes de \mathbf{X} et de \mathbf{y} doivent évidemment se correspondre (voir l'ordre `reorder` en cas de non-correspondance initiale)

Les fonctions associées à PLS sont présentées de manière détaillée. Les autres méthodes ont des fonctions similaires.

3.1.1 PLS (Partial least square)

La régression PLS est probablement la méthode la plus connue. Elle est efficace même lorsque les données \mathbf{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. Cependant, la complexité est source d'instabilité. Il est en général 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).

Les différentes fonctions sont indiquées ici les unes après les autres, classées approximativement dans l'ordre de leur complexité. Dans la suite, `ndim` est un nombre indiquant le nombre de dimensions du modèle retenu.

BASIC PLS

exemple :

```
res = basic_pls(X, y, ndim)
```

```
res =
```

```
T: [1x1 struct]
```

```
P: [1x1 struct]
```

```
beta: [1x1 struct]
```

```
beta0: [1x1 struct]
```

```
meanx: [1x1 struct]
```

```
meany: [1x1 struct]
```

```
predy: [1x1 struct]
```

```
error: [1x1 struct]
```

```
corcoef: [1x1 struct]
```

```
BETA: [1x1 struct]
```

```
BETA0: [1x1 struct]
```

```
loadings: [1x1 struct]
```

```
Q: [1x1 struct]
```

```
PREDY: [1x1 struct]
```

```
RMSEC: [1x1 struct]
```

```
r2: [1x1 struct]
```

res.T = variables latentes du modèle PLS (*scores*)

res.P = *loadings* du modèle PLS (tels que $\mathbf{X} = \mathbf{TP}$)

res.loadings : *loadings* du modèle PLS (tels que $\mathbf{T} = \mathbf{X} * \mathbf{loadings}$). Ce sont les projecteurs PLS donne en effet deux sortes de loadings

res.corcoef corrélation simple entre variable prédites et observées.

res.PREDY contient toutes les prédictions pour 1 à ndim dimensions.

Attention basic_pls n'admet qu'une seule variable y qui est ici un vecteur ! En revanche, il donne les résultats associés à toutes les dimensions de 1 à ndim.

APPLYPLS

```
[predy]=applypls(X,plsmode,(knowny))
```

Prédit les valeurs de y, connaissant la matrice \mathbf{X} et le modèle PLS donné dans plsmode.

plsmode est l'argument de sortie de basic_pls.

Exemple d'utilisation :

```
res = basic_pls(Xcal, y, ndim) % étalonnage
```

```
predy = applypls(Xval, res) % validation
```

Si l'argument optionnel knowny (yconnu) est donné, la fonction donne également des statistiques sur l'erreur de ce jeu de validation.

knowny doit posséder autant de lignes que X.

Exemple:

```
test=applypls(X,modele_pls,y)
```

```
test =
```

```
PREDY: [1x1 struct]
```

valeur de y prédit

```
RMSEV: [1x1 struct]
```

erreur résiduelle de validation

```
r2: [1x1 struct]
```

coefficient de détermination sur le jeu de validation

CROSSVALPLS1A

res=crossvalpls1a(X, y, ndim ,selected)

res =

 calibration: [1x1 struct]

 validation: [1x1 struct]

et

res.validation

 PREDY: [1x1 struct]

 RMSEV: [1x1 struct]

 r2: [1x1 struct]

 T: [1x1 struct]

Etablit une validation sur le modèle PLS comprenant ndim dimensions. y ne représente qu'une seule variable (une seule colonne).

selected est un *vecteur* (MATLAB) et non une structure SAISIR, comprenant n éléments qui indiquent comment les données **X** et **y** sont séparées en jeux d'étalonnage (calibration) et de validation. Les éléments de selected doivent avoir la valeur 0 ou 1. les individus correspondant aux valeurs 0 de selected sont placés dans le jeu d'étalonnage. Les autres sont placés dans le jeu de validation. La fonction divise la collection en jeux d'étalonnage et de vérification. Le modèle PLS est calculé sur les étalons et appliqué sur le jeu de validation.

OBSY et PREDY sont respectivement les valeurs observées et prédites du jeu de validation. Comme les données sont découpées par cette fonction, il est utile de pouvoir disposer du vecteur OBSY.

Par exemple, la commande xy_plot(res.validation.OBSY,1,res.validation.PREDY,3) affiche, pour le jeu de validation, le couple variable observée/variable prédite pour le modèle PLS à 3 dimension.

RMSEV est la racine carrée de la somme des carrés des écarts entre valeurs observées et prédites du jeu de validation.

En anglais : RMSEV *Root mean square error of validation*.

Le vecteur selected peut être établi de manière aléatoire par l'ordre random_select.

selected = random_select(nel, nselect)

nel est le nombre d'élément du vecteur selected (qui prennent la valeur 0 ou 1).

nselect est le nombre d'éléments prenant aléatoirement la valeur 1.

Faire help random_select pour plus de détails.

3.1.2 Régression en composantes principales

La régression en composantes principales est une régression ordinaire, appliquée sur les composantes de l'ACP des données prédictives **X**. Comme dans la méthode PLS, il peut être intéressant de ne former le modèle prédictif qu'avec les composantes auxquelles sont associées les plus grandes valeurs propres.

PCR

```
pcrres=pcr(X,y, dimmax)
```

```
res =
```

```
  pca: [1x1 struct]
  beta: [1x1 struct]
  predy: [1x1 struct]
  r2: [1x1 struct]
  averagey: 12.62
  beta1: [1x1 struct]
  obsy: [1x1 struct]
  rmsec: [1x1 struct]
```

Effectue une régression en composantes principales pour estimer **y** (une seule variable), connaissant **X**.

La fonction PCR établit tous les modèles comprenant 1 à **dimmax** composantes introduites, et donne tous les résultats de chacun de ces modèles. Les composantes sont introduites dans l'ordre des valeurs propres. Par exemple, le modèle prédictif 1, est établi avec la première composante de l'ACP. Le modèle prédictif 5 est établi à partir des composantes 1 à 5.

La structure **PCRRES** contient tous les résultats pour chacun des modèles (soit **maxdim** modèles). Ainsi, par exemple, le champ **predy** est une structure SAISIR, avec en ligne les individus, et en colonnes, les valeurs prédites de **y** pour chacun des **maxdim** modèles.

L'erreur résiduelle (**rmsec**, *root mean square error of calibration*) est calculée avec $n-k-1$ degré de liberté, avec **n** : le nombre d'individus et **k** la dimension du modèle.

Les autres champs ont les mêmes significations que ceux employés dans la méthode PLS.

Les champs **beta** et **beta1** contiennent respectivement les coefficients de la régression appliqués sur les coordonnées factorielles (**beta**) ou sur les données d'origine. Le champ **PCA** est une structure contenant les résultats de l'ACP sur **X** (voir paragraphe 2.1.3).

APPLYPCR

```
predy=applypcr(pcrtype, Xval)
```

```
>>predy =
```

```
>> d: [140x10 double]
```

```
>> i: [140x10 char]
```

```
>> v: [10x6 char]
```

Prédit la valeur **y**, connaissant **Xval**, à partir des modèles prédictifs donnés par **PCRTYPE**.

PCRTYPE est le résultat de l'application de la fonction PCR.

DIMCROSSPCR1

```
pcrres1=dimcrosspcr1(X, y, maxdim, selected)
```

```
>>pcrres1 =
```

```
>> r2: [1x1 struct]
```

```
>> predy: [1x1 struct]
```

```
>> rmsecv: [1x1 struct]
```

```
>> obsy: [1x1 struct]
```

```
>> pcrtype: [1x1 struct]
```

Effectue une validation de la régression en composantes principales.

Voir la commande `dimcrosspls1a` pour les détails

`pcrtype` contient le modèle prédictif établi sur le jeu d'étalonnage.

3.1.3 régression pas à pas

STEPWISE REGRESSION

(Exemple tiré du manuel « G. Saporta, *Probabilités, analyse des données et statistique*, Editions Technip, Paris, 1990, page 396).

On cherche à prédire le prix d'une voiture (vecteur y) connaissant différentes variables caractéristiques : *CYL*, *PUIS*, *LON*, *LAR*, *POIDS*, *VITESSE* (matrice X).

Le seuil d'entrée ou de sortie de variables du modèle est fixé à 0.15.

```
>> stepwise_type=stepwise_regression(x,y,0.15)
```

```
>>Entering variable PUIS
```

```
>>error = 4076.0077
```

```
>>Entering variable POIDS
```

```
>>error = 3916.3302
```

```
>>Entering variable CYL
```

```
>>error = 3974.3516
```

```
>>Finished at previous step
```

```
stepwise_type =
```

```
 [1x1 struct] [1x1 struct] []
```

La régression introduit la variable *PUIS*, suivi de la variable *POIDS*. L'introduction de la variable suivante, *CYL* ne donne pas une probabilité suffisamment faible, et cette variable est rejetée.

`stepwise_type` contient les résultats de tous les modèles respectant le seuil de probabilité.

Ces résultats sont dans des structures de cellules (*Array of cells*). On accède aux éléments de ces cellules par les signes « `{}` » et non par « `()` ». Chaque indice donne les résultats associés à un pas donné de la régression.

On a par exemple

```
>>stepwise_type{2}
>> message: 'Entering variable POIDS          at step 2'
>> res: [1x1 struct]
>> intercept: 1.7756e+003
>> RMSE: 3.9163e+003
>> r2: 0.6866
>> adjusted_r2: 0.6448
>> F: 16.4331
>> probF: 1.6614e-004
>> ypred: [1x1 struct]
```

Le champ `res` est une matrice SAISIR, donnant les résultats associés à chacun des termes du modèle. On peut, par exemple, transférer cette matrice sous Excel par la commande : `saisir2excel(stepwise_type{2}.res, 'res2')`;

Le tableau Excel ainsi obtenu est le suivant :

	Lower reg. coef.	Higher conf	Std reg.coef.	t reg. coeff.	Preg. coef.	
PUIS	172.97	18.608	327.33	72.42	2.3884	0.030509
POIDS	16.451	-6.5141	39.416	10.774	1.5269	0.1476

Les variables sont indiquées en lignes.

Les colonnes donnent successivement : les coefficients de la régression, les limites basses et hautes au seuil $P < 0.05$ de ces coefficients, leur écart-type, le test de t, et la probabilité de l'hypothèse nulle : « *les coefficients sont égaux à 0* ».

[APPLY STEPWISE REGRESSION](#)

```
res=apply_stepwise_regression(stepwise_type,x,(y))
```

```
>>res =
  predy: [1x1 struct]
  rmsev: [1x1 struct]
  r2: [1x1 struct]
```

Même principe que dans les méthodes précédentes. Tous les modèles (correspondant à toutes les étapes de la régression pas à pas) sont appliqués. Si le paramètre optionnel `y` (valeur observée) est donné, la fonction donne également les coefficients de corrélation et les erreurs résiduelles attachés à chacun de ces modèles.

[DIMCROSS STEPWISE REGRESSION](#)

```
fonction[res]=dimcross_stepwise_regression(x,y,selected,Pthres)
```

Même logique que les méthodes précédente. **X** est la matrice des données prédictives, **y** le vecteur de la variable à prédire. `selected` donne la division en étalon et vérification.

`Pthres` (*Probability threshold*) est le seuil d'introduction ou d'élimination des données.

3.2 Discrimination

On peut avoir une liste non exhaustive des méthodes de régression disponibles par la commande `discrimination` .

Toutes les méthodes de discrimination demande une matrice de données explicatives **X** et un vecteur désignant les groupes, que nous appellerons `gr`, dans la suite. Si **X** est de dimension $n \times p$, `gr` est un vecteur de dimension $n \times 1$. Ce vecteur contient des nombres entiers, qui peuvent prendre des valeurs allant de 1 à `maxgroup`, où `maxgroup` est le nombre de groupes qualitatifs. Il est nécessaire que chaque groupe soit représenté au moins une fois dans la collection d'étalonnage. Dans SAISIR, `gr` doit être une structure. `gr.i` indique les identificateurs des individus (normalement identiques à `X.i`). `gr.d` est un vecteur colonne d'entiers indiquant les groupes, et `gr.v` contient un seul identificateur, par exemple « `group` ».

[CREATE_GROUP1, CREATE_GROUP](#)

Lorsque les descripteurs des individus sont des clés pouvant désigner les groupes qualitatifs, 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.

L'ordre `create_group1` peut être appliqué.

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

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

Les groupes sont créés en exploitant `oil.i` et en recherchant les chaînes de caractères qui commencent en position 1 du nom et finissent en position 2. Les échantillons ayant des chaînes extraites identiques font partie du même groupe.

la structure `gr.g` donne les effectifs de ces groupes, et les lettres qui correspondent.

<u><i>gr.g.i</i></u>	<u><i>gr.g.d</i></u>
Ca	56
Cs	33
EI	50
Is	65
Na	25
Sa	206
Si	36
Um	51
WI	50

Ainsi, 36 observations font partie du 7^{ème} groupe, la région Si.

Attention ! Si les chaînes de caractères servant à former les groupes sont déjà des nombres, la fonction ne garantit pas que l'on aura la correspondance {« 1 », groupe 1}, {« 2 », groupe 2} et ainsi de suite. En effet, les numéros sont associés aux groupes de manière arbitraire. Pour éviter cette difficulté, lorsque l'on veut imposer les numéros de groupes, on peut utiliser l'ordre `create_group`

```
gr=create_group(saisir,code_list,startpos,endpos)
```

Par exemple la commande

```
gr=create_group(qualite, ['1' ; '2' ; '3' ; ],2 , 2)
```

Recherche dans la matrice `qualite`, les descripteurs qui ont, en position 2, les chaînes de caractères « 1 », « 2 » ou « 3 », et établit le fichier `gr` correspondant. La fonction signale si des observations n'ont pas de chaînes de la liste `code_list`. Dans ce cas, il faut résoudre le problème avant de continuer.

Seules quelques méthodes sont présentées ici.

[MAHA1, MAHA3, MAHA4, MAHA5, MAHA6](#)

Discriminations ascendantes par calcul de la distance de Mahalanobis.

Ces fonctions sont très voisines, et repose sur le calcul de la distance de Mahalanobis sur la covariance totale.

Les méthodes se distinguent seulement par :

1) *Le mode de sélection des variables :*

-Soit par la maximisation du pourcentage d'observations correctement classées

-Soit par la maximisation de la trace de $T^{-1}B$

(voir JM Romeder, *Méthodes et programmes d'analyse discriminante*, Dunod, Paris, 1973)

2) *La possibilité d'avoir un jeu de validation.*

On dispose des méthodes suivantes (les numéros de ces méthodes n'ont pas de sens)

Sans validation :

```
fonction[discrtype]=maha3(saisir,group,maxvar)
```

Maximisation de la trace de $T^{-1}B$

```
fonction[discrtype]=maha4(saisir,group,maxvar)
```

Pourcentage d'échantillons correctement classés

Avec validation :

```
fonction[discrtype1]=maha1(calibration,calibration_group,maxvar,test,test_group)
```

Pourcentage d'échantillons correctement classés. Deux jeux de données.

```
fonction[discrtype]=maha5(saisir, group, maxvar, selected)
```

Pourcentage d'échantillons correctement classés. Un seul jeu de données, divisé selon les valeurs du vecteur `selected`.

```
fonction[discrtype]=maha6(saisir,group,maxvar,selected)
```

Maximisation de la trace de $T^{-1}B$.

Exemple d'utilisation (avec validation), sur l'exemple *huile d'olive*

```
>>sel=random_select(572,200) % 200 observations en validation
>>dis=maha6(oil1,gr,8,sel) % discrimination avec maximisation de la trace

>>dis =

>> ncorrect: [1x1 struct]
>> nscorrect: [1x1 struct]
>> classed: [1x1 struct]
>> sclassed: [1x1 struct]
>> confusion: [9x9 double]
>> varrank: [5 8 7 2 3 6 4 1]
>> sconfusion: [9x9 double]
```

Le préfixe 's' comme dans `sclassed` ou `nscorrect` désigne les individus de validation (*supplémentaires*).

Les champs `classed`, `sclassed` et `confusion`, `sconfusion` donnent les résultats associés au dernier pas de la méthode ascendante. Pour avoir des pas intermédiaires, il faut utiliser la fonction à nouveau, avec une valeur différente de `maxvar`. Par exemple, `dis=maha6(oil1,gr,5,sel)` « reconstitue » le pas n° 5.

`confusion` et `sconfusion` donne les matrices d'affectation des échantillons, avec en ligne : les groupes réels, et en colonnes, ceux trouvés par le calcul.

[PLSDA, CROSSPLSDA](#)

fonction[`type`]=`plsda`(`x`, `group`, `ndim`)

Analyse discriminante par la méthode PLS, avec `ndim` dimensions.

La méthode PLS consiste tout d'abord à établir un modèle linéaire PLS2. Dans ce modèle, les variables Y à prédire sont les indicatrices de groupe.

On peut ensuite procéder de deux manières différentes (désignées par 0 ou 1).

Méthode 0 :

On prédit les valeurs des indicatrices. Chaque individu à classer est affecté dans le groupe donnant la plus grande valeur de l'indicatrice.

Méthode 1 :

Les variables latente de PLS (scores T) deviennent les variables d'une analyse discriminante linéaire classique.

Exemple d'utilisation:

```
>>pl=plsda(oil1,gr,8)
```

```
>>pl =
```

```
>>    beta: [1x1 struct]
>>    beta0: [1x1 struct]
>>     t: [1x1 struct]
>>    predy: [1x1 struct]
>>    classed: [1x1 struct]
>>    ncorrect: 478
>>    confusion: [1x1 struct]
>>    classed1: [1x1 struct]
>>    ncorrect1: 533
>>    confusion1: [1x1 struct]
>>    tbeta: [1x1 struct]
>>    tbeta0: [1x1 struct]
>>    linear: [1x1 struct]
>>    linear0: [1x1 struct]
>>    info: [1x76 char]
```

Voici la définition de ces champs :

beta et beta0 : coefficients des modèles linéaires permettant de prédire les indicatrices.

t : variables latentes PLS (scores)

predy : indicatrices prédites (autant que de groupes qualitatifs)

classed : groupes prédits selon la méthode 0

ncorrect : nombre d'affectation correctes selon la méthode 0

confusion : matrice de confusion obtenue avec la méthode 0

classed1 : groupes prédits selon la méthode 1

ncorrect1 : nombre d'affectation correctes selon la méthode 1

confusion1 : matrice de confusion obtenue avec la méthode 0

tbeta et tbeta0 : coefficients des modèles linéaires permettant de prédire les variables latentes PLS.

linear et linear0 : forme linéaire permettant d'affecter rapidement les échantillons

Pour une observation inconnue x , on peut calculer le vecteur $\text{test} = x * \text{linear} + \text{linear0}$. Ce vecteur à autant d'éléments qu'il y a de groupes qualitatifs. La règle d'affectation est donnée par l'indice du minimum de test .

La fonction `crossplsda` permet de valider le modèle, par exemple.
`res=crossplsda(oil1,gr,5,sel)`

`res =`

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

Nota : La méthode 1 est en général supérieure à la méthode 0

[FDA1, FDA2 et famille FDA](#)

Ces méthodes d'analyse factorielle discriminante (*FDA : factorial discriminant analysis*) sont bien adaptées aux tableaux possédant de nombreuses variables très redondantes, comme les données de mesures physiques (chromatogrammes, spectres).

D'une manière similaire à la régression en composantes principales, elles procèdent en deux étapes : ACP sur les données, suivi d'une analyse discriminante linéaire. Elles offrent la possibilité de donner des cartes factorielles discriminantes. La méthode de base a été présentée dans l'article : *Bertrand et al., J of Chemometrics, Vol . 4, 413-427 (1990)*.

[FDA1](#)

Analyse factorielle discriminante ascendante sur les coordonnées factorielle d'une ACP.
`fonction[fdatatype]=fda1(pcatype, group, among, maxvar)`

Exemple :

```
>>p=pca(oil1) ;
>>fdatatype=fda1(p,gr,8,3)
```

```
>>fdatatype =
```

```
>>   introduced: [1x1 struct]
>>   ncorrect: [1x1 struct]
>>   beta: [1x1 struct]
>>   datafactor: [1x1 struct]
>>   centroidfactor: [1x1 struct]
>>   eigenval: [1x1 struct]
>>   confusion: [1x1 struct]
>>   average: [1x1 struct]
```

La méthode demande comme premier argument, le résultat d'une ACP (argument de sortie de la fonction `pca`).

Le deuxième argument est le vecteur de groupe, `gr`, comme précédemment.

Le troisième argument indique le sous ensemble des composantes que l'on souhaite considérer dans l'analyse. Par exemple, si `among` (*parmi*) vaut 5, la fonction ne choisira les variables prédictives que parmi les 5 premières composantes. Le dernier argument, `maxvar`, indique le nombre maximal de composantes introduites dans le modèle. On doit avoir :
 $\text{maxvar} \leq \text{among}$.

La fonction établit des modèles prédictifs comprenant 1 à `maxvar` composantes. Les composantes sont introduites, comme précédemment, de manière à maximiser $T^{-1}B$,

Le résultat est une structure contenant des matrices SAISIR, comme dans les méthodes précédentes. Les champs ont la même signification que précédemment ;

`fdatype.datafactor` donne les facteurs, qui peuvent être représentés sous forme de cartes factorielles, exactement comme pour l'ACP (fonction `pca`).

`fdatype.centroidfactor` donne les barycentres des groupes dans l'espace de projection.

`fdatype.beta` contient les coefficients du modèle linéaire qui s'appliquent sur le tableau X d'origine centré pour donner les facteurs. .

`fdatype.eigenval` donne les valeurs propres de l'analyse factorielle discriminante.

[FDA2](#)

Analyse factorielle discriminante ascendante sur les coordonnées factorielle d'une ACP, avec validation.

```
res=fda2(X,gr,among,maxvar,selected)
```

res =

```
introduced: [1x1 struct]
ncorrect: [1x1 struct]
nscorrect: [1x1 struct]
beta: [1x1 struct]
datafactor: [1x1 struct]
centroidfactor: [1x1 struct]
eigenval: [1x1 struct]
confusion: [6x6 double]
average: [1x1 struct]
sconfusion: [6x6 double]
```

Mêmes principes que précédemment, avec une validation. Attention ! Contrairement à `FDA1`, la fonction `FDA2` admet comme premier argument le tableau **X** des variables explicatives, et non le résultat d'une ACP.

Voir aussi les fonctions :

`APPLYFDA1` et `CROSSFDA1`

4 Dictionnaire des fonctions disponibles

Par ordre alphabétique. (Voir aussi la liste par thème au paragraphe 5)

On peut (en général !) obtenir une aide succincte par la commande `help` suivi du nom de la fonction.

L'importance et l'intérêt pratique des fonctions sont notés par le signe * (de * à ****).

Les descriptions sont essentiellement celles données par l'ordre `help` (en anglais)

ADDCODE*

```
function str1 = addcode(str,code,deb_end)
add the string "code" BEFORE or AFTER any string of vector str
function str1 = addcode(str,code)
useful for recoding identifier
if deb_end==0: addition before (default)
```

ALPHABETIC SORT

Sort the rows of the `saisir` file according to the alphabetic order of rows identifiers

```
function s1=alphabetic_sort(s,start_pos:end_pos)
```

useful in colouring maps

`startpos` `endpos` indique sur quelle partie des caractères des identificateurs porte le classement.

Par exemple: identificateurs

'1mais '

'3ble '

Si `startpos=2` et `endpos=5`, seront classés à partir des chaînes de caractères 'mais' et 'ble '

ANAVAR1****

One way analysis of variance on spectral data

```
function res = anavar1(X,g)
```

Performs a one-way analysis of variance for each of the column of X.

`g` is the SAISIR file indicating the groups.

The function returns

res.F : vector of Fisher values for each variable in X

res.p : vector of associated probabilities.

Note: **res.F** and **res.p** can be examined as curves (using `CURVE` function)

or by the command `SHOW_VECTOR` (discrete variables)

Cette fonction est très utile pour examiner très rapidement les relations entre des jeux de variables et des groupes qualitatifs.

ANOVAN1****

N-way analysis of variance (ANOVA) on data matrices.

```
function res = anovan1(X,model,gr1, gr2, ...)
```

Performs as many N-way analyses of variance as the number of columns in X

X: matrix of response values

model: see ANOVAN, gives the level of desired interactions

(1= no interactions studied; 2: first degree of interactions)

`gr1`; `gr2` ... (variable number of arguments): qualitative groups

forming a factor of the ANOVA

The function returns:

res.F : the F value associated with each effect and interaction

res.P : probability

res.df : (characters): degrees of freedom

res.singular: singularity of the model. If the singularity == 1, the model is redundant and a lowest level of interaction must be tested.

see also: ANOVAN, ANOVA, ANAVAR1

[appendbag1](#)

- Merge an arbitrary number of structure bags according to rows

usage: [saisir]= appendbag1(saisir2,

saisir is a structure i,v,d

the number of columns in saisir files must be equal

La structure bag (sac) est utilisée lorsque l'on doit manipuler des tableaux de chaînes de caractères.

C'est une structure comme saisir, avec les champs ".i" et ".v" servant à étiquetter les lignes et les colonnes.

En revanche le champ "d" ne contient pas des valeurs numériques mais des chaînes de caractères.

Exemple : fichier sous Excel, sauvegardé en format ".csv" sous le nom "exemple.csv"

	nom	prénom
personne1	DUPONT	Jean
personne2	DURAND	Paul
personne3	MARTIN	Jacques

La commande

[saisir, mybag]=excel2bag('exemple',['B';'C'])

La structure saisir est ici vide (voir la commande excel2bag).

mybag est une structure bag.

mybag =

v: [2x20 char]

d: [3x20x2 char]

i: [3x20 char]

Le champ .d est ici un tableau 3voies de caractères. Attention! C'est le deuxième indice qui désigne la chaîne.

Pa exemple:

bag.d(2,:,1)

DURAND

[APPENDCOL ****](#)

Merges two files according to columns

usage: [X]= appendcol(X1,X2)

saisir is a structure i,v,d

the number of rows in X1 and X2 must be equal

[APPENDCOL1 ****](#)

Merge an arbitrary number of files according to columns

usage: [X]= appendcol(X1, X2, ...)

the number of rows in X1 and X2 files must be identical

[APPENDROW****](#)

Merge two files according to rows

usage: [X]= appendrow(X1,X2)

The number of columns in X1 and X2 must be equal

APPENDROW1****

Merge an arbitrary number of files according to rows

usage: [X]= appendrow(X1,)

the number of columns in files must be equal

APPLY MULTIPLE REGRESSION

Applies multiple_regression on "unknown" data

function[res]=apply_multiple_regression(x,type,(y))

Using the models (in type) as returned by the function

"multiple_regression"

if y (optional) is given, gives also the Root mean square error on y

Ne fonctionne qu'en parallèle avec la fonction multiple_regression

APPLY RIDGE REGRESSION **

Apply ridge regression on "unknown data"

function[res]=apply_ridge_regression(ridgetype,x,(y))

if y is given, the verification stats (RMSEV are also assessed)

APPLY STEPWISE REGRESSION**

Apply stepwise_regression on "unknown" data

function[res]=apply_stepwise_regression(type,x,(y))

Using the models (in type) as returned by stepwise_regression

Build as many models as available in type

if y (optional) is given, gives also the Root mean square error on y

APPLYFDA1 **

Application of factorial discriminant analysis on PCA scores

function[res]=applyfda1(X,fda1type,(actual_group))

predict the qualitative group on X, the model is contained in fda1type

if actual_group is given, assess also the confusion matrix (actual *versus* predicted)

APPLYLR1**

Apply basic latent root model on saisir data x

function [predy]=applylr1(lr1type,x)

creates as many y predicted as allowed by the dimensions in lr1type

lr1type is the output argument of the function LR1

APPLY NUÉE *

apply Nuee dynamique (KCmeans)

function[res]=nuee(X,barycenter)

X : a matrix of dimension n x p

Barycenter : a matrix defining the barycenter k x p with k groups

Barycenter is possibly the field "center" of the output of function "nuee"

Cette fonction affecte les observations dans X dans des groupes à partir des barycentres contenus dans barycenter

Les individus sont affectés au barycentre le plus proche, au sens de la distance euclidienne usuelle.

APPLYPCA**

Assess the scores of supplementary observations

function [supscores]=applypca(pcatype, X)

assess the scores of supplementary observations

pcatype is the structure resulting of the application of pca on principal observations

The number of columns of X must be compatible with pcatype.

If normed_pca was applied, X is divided by the std of the principal observation prior to projection

APPLYPCR**

Assesses a basic PCR on data

function [predy]=applypcr(pcrtype,X)

apply a basic pcr (in pcrtype) on X

creates as many y predicted as allowed by the dimensions in pcrtype

APPLYPLS**

Applies a pls model on an unknown data set

function [predy]=applypls(x,plsmodel)

plsmodel is the structure obtained from saisirpls

returns predy

APPLYPLSDA *

Applies pls discriminant analysis after model assessment using plsda

function[res]=applyplsda(x,model,actual_group)

return the predicted group on "unknown" data X

'model' is the structure returned by function 'plsda'

APPLYSPCR*

Applies a stepwise PCR

function [predy]=applyspcr(spctype,X)

applies a pcr with selection (in spctype) on saisir data X

creates as many y predicted as allowed by the dimensions in pcrtype

APPLY QUADDIS

- Application of Quadratic discriminant analysis
function result = apply_quaddis(quaddis_type,x,(known_group));
Application of quadratic discriminant analysis (test)

-- Inputg args:

quaddis_type : output of function "quaddis" (structure)

x : predictive data matrix (n x p)

known_group : true groups of observations in x (n x 1) (optional)

-- Ouput args:

result with fields:

predgroup : predicted groups (n x 1)

density : pseudo-density (n x gmax)

proba : probability of belonging to a given group (n x gmax)
if "known_group" defined

nsincorrect100 : percentage of correct classification (number)

sconfus : confusion matrix (gmax x gmax)

BAG2GROUP

use the identifiers in bag to create groups

function [group_type]=bag2group(bag)

Use the identifiers for creating groups.

Build an array of cells. In each cell, the column of bag.d (matrix of char) are processed in order to form a vector of group

creates as many group as different strings in the column of bag.d

Cette fonction est un peu complexe, elle est utile pour manipuler des fichiers Excel dans lesquels des groupes qualitatifs sont désignés par des chaînes de caractères.

Voir la fonction Excel2group.

BASIC PLS**

Basic pls with keeping loadings and scores

function[res] = basic_pls(X,y,ndim)

A single y and 1 to ndim dimensions

res =

T: scores PLS

P: paramètre P

beta: beta (termes de la regression applicables sur X) du modèle à ndim dimensions

beta0: constante du modèle à ndim dimensions

meanx:moyenne de x

meany: moyenne de y

predy: y prédit du modèle à ndim dimensions

error: [1x1 struct] erreur quadratique du modèle à ndim dimensions

corcoef: [1x1 struct] coefficient de corrélation du modèle à ndim dimensions

BETA: beta (terme de la regression applicables sur X) de tous les modèles

BETA0: constantes de tous les modèles

loadings: loading pls

Q:coefficients de la regression sur les T

PREDY: y prédit sur tous les modèles
RMSEC: erreur résiduelles de tous les modèles
r2: coefficients de détermination de tous les modèles

BASIC PLS2**

PLS2 on several variables, several dimensions

function result=basic_pls2(X,Y,maxdim)

returns result.T : the latent variables

result.res: cells corresponding to each final predicted variable

Ce programme établit tous les modèles PLS2 avec **X** comme variables prédictives et **Y** une matrice de variables à prédire.

La matrice **result.T** contient les composantes (scores PLS) sous la forme d'une matrice.

result.res est un vecteur de cellule (accolade et non parenthèse). Chacune des cellules contient des structures donnant les resultants associé à chaque variable.

Exemple:

```
>>xcal =
  d: [32x801 double]
  i: [32x2 char]
  v: [801x4 char]
>>cv =
  i: [32x10 char]
  d: [32x3 double]
  v: [3x1 char]
>>manip=basic_pls2(xcal,cv,10);% prediction des variables dans cv, à l'aide de xcal,
10 dimensions maximum.
>>manip.T % scores de PLS
ans =
  d: [32x10 double]
  i: [32x2 char]
  v: [10x2 char]
>> manip.res; % vecteur de cellules contenant les resultants associés à chaque
variable
ans =
 [1x1 struct] [1x1 struct] [1x1 struct]
% Résultats associés à la deuxième variable:
manip.res{2}% remarquez les accolades { }
ans =
  nom: 'R' % la deuxième variable s'appelle 'R'
  BETA: [1x1 struct] % coefficients du modèle
  BETA0: [1x1 struct] % constantes du modèle
  PREDY: [1x1 struct] % variable Y prédite
  RMSEC: [1x1 struct] % Root mean square errors of calibration
  r2: [1x1 struct] % coefficients de détermination
```

Il faut noter que, tous les modèles (comportant dans l'exemple de 1 à 10 dimensions) sont calculés.

Par exemple:

```
>>manip.res{2}.BETA
  d: [801x10 double]
  i: [32x2 char]
```

v: [10x2 char]

Les coefficients du modèle à 5 dimensions pour la variable 2, peuvent par exemple être examinés par:

```
tcurve(manip.res{2}.beta,5)
```

BMAHA **

Assess a simple backward discriminant analysis

```
function discrtype]=bmaha(saisir,group)
```

assess a simple linear discriminant analysis introducing rows to 1 variables

at each step, the less discriminating variable (according to the percentage of correct classification) are discarded. Only backward

BROWSE***

Browses a series of curves

```
function browse(X)
```

Display the rows of the matrix X as curves

Right button to go down, Left button to go up, Ctrl C to exit

BUILD INDICATOR *

- build a disjoint table

```
function [indicator, groupings]=build_indicator(x);
```

each column of X must contain integer values

build the complete table of indicators

Useful for computing multiple correspondence analysis

Chaque colonne de X est considérée comme étant représentative de groupes qualitatifs codés par des nombres entiers.

Pour chaque colonne de X, le tableau des indicatrices est formé. Il y a autant d'indicatrices par colonnes que de nombres entiers différents dans cette colonne. Le nom des variables dans indicator est formé par le nom initial de la variable dans X, suivi d'un numéro indiquant le nombre originalement présent dans X.

Par exemple, une colonne de X contenant les seuls nombres 0, 3, 5 de nom quality sera à l'origine de 3 indicatrices quality0, quality3 quality5. La matrice de sortie indicator contient toutes les indicatrices de toutes les colonnes de X.

CA ***

CORRESPONDENCE ANALYSIS

```
function ca_type=ca(N);
```

Compute correspondence analysis from the contingency table in N

If only groupings are available, the contingency table must be computed before using this function (see for example function "contingency_table")

=====
Fields of the output

score :CA scores of rows followed by CA scores of columns

eigenval :eigenvalues, percentage of inertia, cumulated percentage

contribution :contribution to the component rows, then columns

squared_cos :squared cosinus row, then columns

khi2 :khi2 of the contingency table

df :degree of freedom

probability :probability of random values in contingency table
=====

The identifiers of rows of 'score' (which are the identifiers of rows and columns of N (N.i and N.v) are preceded with the letter 'r' or 'c'. It is therefore possible to use color for emphasizing row and columns in the simultaneous biplot of rows and columns

Source : G. Saporta. *Probabilités, analyse des données et statistiques.*
: Edition Technip, page 198 and followings.

REMARK : use function `ca_map` to plot the biplot observation/variable

CA MAP **

Colored map for correspondence analysis: using a portion of the identifiers as labels
`ca_map(X,col1,col2,startpos,endpos,(col1label),(col2label),(title),(charsize),(margin))`

Biplot of two columns as colored map useful for correspondance analysis (from function `ca`)

The coloration of the displayed descriptors depends on the arguments

`startpos` and `endpos`. If one of this argument is zero: single (black) color

Otherwise, from the names of individual, the string name(`startpos:endpos`) is extracted. Two observations

for which these strings are different, are also colored differently.

SPECIFIC TO CORRESPONDANCE ANALYSIS:

If the first letter of the identifier is either *c*(column) or *r* (row), this letter is removed in the name.

The letter *c* produces an italic display. This allows a representation in which the variables are in italic letter

MAP****

Graph of map of data using identifiers as names

function `map(X,col1,col2,(col1label),(col2label),(title),(charsize),(margin))`

Biplot of two columns as a map, with display of the identifiers

`margin`: expansion of axis in order to cope with the long identifiers

MAP3D ***

Draw a 3D map

function `map3D(X,col1,col2,col3,(labcol1),(labcol2),(labcol3),(title),(charsize))`

Display columns `col1`, `col2`, `col3` as X, Y and Z values of a 3D plot.

Options: `labcol`: label of axes

`charsize`: size of the characters (default :6)

BARYCENTER_MAP **

graph of map of barycentre

function barycenter_map(s,col1,col2,gr)

Display two columns as a map

Each observation is linked to its own barycentre by a straight line

- s saisir data file;

-col1 col2: index of the columns to be displayed

-gr : qualititative groups

COLORED_MAP1****

colored map : using a portion of the identifiers as labels

colored_map1(X,col1,col2,startpos,endpos,(col1label),(col2label),(title),(charsize),(margin))

Biplot of two columns as colored map

The coloration of the displayed descriptors depends on the arguments startpos and endpos.

From the names of individual, the string name(startpos:endpos) is extracted. Two observations for which these strings are different, are also colored differently.

COLORED_MAP2****

Colored map : using a portion of the identifiers as labels

colored_map2(X,col1,col2,startpos,endpos,(col1label),(col2label),(title),(charsize),(margin))

Biplot of two columns as colored map

The coloration of the displayed descriptors depends on the arguments startpos and endpos.

From the names of individual, the string name(startpos:endpos) is extracted. Two observations for which this strings are different, are also colored differently.

Contrary to carte_couleur1, the complete identifiers are displayed.

COLORED_MAP4 ****

Colored map according to given choice

colored_map4(X,col1,col2,color_choice,(symbol_choice);)

Biplot of two columns as colored map

The coloration of the displayed descriptors depends on the arguments

choice (either matrix of char, vector of number or saisir structure);

with number of elements equal to the number rows in s;

if the elements of "choice" are different, they are also colored differently.

if "symbol_choice" is also defined (either matrix of char, vector of number

or saisir structure) different symbols are used. The color of the point

is then given by "choice", and the shape of the symbol depends on

"symbol_choice"

Example of use:

Colored text with color determined by the second character in wheat.i

```
colored_map4(wheat,1,50,wheat.i(:,2));;
```

Colored symbol: shape determined by the second character in wheat.i

color determined by the third character in wheat.i

```
colored_map4(ble,1,50,ble.i(:,2),ble.i(:,3));
```

SYMBOLE_MAP ****

Map with symbols : using a portion of the identifiers for
symbole_map(X,col1,col2,startpos,endpos,(col1label),(col2label),(title),(charsize))

Biplot of two columns as a map with symbols

The symbols of the different observations depend on the arguments
startpos and endpos.

From the names of individual, the string name(startpos:endpos) is extracted. Two observations for which this strings are different, are represented with different symbols.

CENTER *

Subtracts the average row to each row

function X = center(X1)

CHANGE_NAME

- change the name of the observations of file X

function res=change_name(x,old_name,new_name);

old_name must contain at least all the identifiers in X.i;

new_name must have the same number of row than old_name;

CHANGE_SIGN *

Changes the sign of a component and of associated eigenvector

function [pctype] = change_sign(pctype,ncomp)

Lorsqu'on effectue plusieurs ACP sur des données similaires, il peut être avantageux d'avoir des composantes homologues représentées sur les cartes factorielles, avec le même signe.

Cette fonction offre la possibilité de changer la direction d'une composante à la fois.

CHECK_NAME*

Controls if some strings are strictly identical in a string array

function [detected,names]=check_name(string)

find the string witch are strictly identical in s.

Principalement utilisé en liaison avec l'ordre REORDER.

COL_MAP1

Colored map : using a portion of the identifiers as labels

col_map1(X,col1,col2,char_range,(col1label),(col2label),(title),(charsize),(margin))

Biplot of two columns as colored map

Fait la même chose que **colored_map1** La seule différence réside dans la manière de désigner les caractères de la coloration: char_range est ici un vecteur.

Par exemple, si char_range = [1 3 8] les observations seront coloriées en fonction des caractères en position [1 2 8].

COLORED CURVES

- displays curves coloured according to groups

function h=coloured_curves(s,range,group)

range is a vector indicating the rows to be displayed

group peut-être le résultat de la fonction create_group!

COMDIM **

Finding common dimensions in multitable data (saisir format)

function[res]=comdim(col,(ndim),(threshold))

Finding common dimensions in multitable according to method 'level3'

proposed by *E.M. Qannari, I. Wakeling, P. Courcoux and H. J. H. MacFie*

in Food quality and Preference 11 (2000) 151-154

threshold (optional): if the difference of fit<threshold then breaks the iterative loop

ndim : number of common dimensions

default: threshold=1E-10; ndim=number of tables

Returns:

Q : nrow x ndim the observations loadings

explained : 1 x ndim, percentage explanation given by each dimension

saliences : ntable x ndim weight of the original tables in each dimensions.

col is an ARRAY OF CELLS of saisir files (the numbers of rows in each table must be equal) .

CONTINGENCY KHI2**

compute khi2 stats on a contingency table

function res=contingency_kh2(table);

table can be computed from contingency_table

res =

theo: [1x1 struct]

Distribution théorique

khi2: [1x1 struct]

Valeur du KHI2

dll: [1x1 struct]

Degrés de liberté du KHI2

P: [1x1 struct]

Probabilité de l'hypothèse nulle

CONTINGENCY TABLE**

Compute a contingency table

function table=contingency_table(g1,g2)

g1 and g2 are the group files possibly computed from "create_group1".

g1.d and g2.d are vectors of integer values with the same number of elements.

In these vectors, a same number indicates the belonging to the same group.

CORMAP***

Assesses the correlation between two tables

function [cor] = cormap(X1,X2)

The tables must have the same number of rows

The results is a p1 x p2 matrix (with p1 and p2 the numbers of rows of X1 and X2 respectively)

An element cor.d(a,b) is the correlation coefficient between the column a of X1 and b of X2

CORRELATION CIRCLE **

Assesses and displays the correlation circle after PCA

function [res]=correlation_circle(pcatype,X,col1,col2)

pcatype is the result of the order pcatype=pca(X)

X is the original data matrix

col1 and col2 are the ranks of the components to be represented.

CORRELATION PLOT

- Draw a correlation between scores and tables

function handle=correlation_plot(scores,col1,col2, saisir1,saisir2, ...);

Input argument:

scores: - ORTHOGONAL scores obtained by multidimensional analysis

col1 and col2: - Indices of the scores to be plotted

saisir1, saisir2, ... - Arbitrary number of tables giving the variables to be plotted

The number of rows in the scores and other tables must be identical

Chaque tableau saisir1, saisir2 est associé à une couleur différente.

Le cercle en pointillé indique 50% de la variable expliquée

Si l'argument scores n'a pas des colonnes orthogonales, la représentation se fait, mais un message d'avertissement apparaît.

CORRECT BASELINE

simple linear baseline correction, using intensity

function [saisir] = correct_baseline (saisir1,col1,col2)

The baseline is modeled by a straight line going from data points col1 to col2

CURVE****

Represent a row of a matrix as a curve

usage handle=curve(X,(nrow), (xlabel),(ylabel),(title))

CURVES****

Represents several rows of a matrix as curves

usage handle=curves(saisir,range,(xlabel),(ylabel),(title))

range is a vector indicating the rows to be displayed

COVMAP **

Assesses the covariances between two tables

function [cov] = covmap(X1,X2)

the tables must have the same number of rows

The results is a p1 x p2 matrix (with p1 and p2 the numbers of rows of X1 and X2 respectively)

An element cov.d(a,b) is the covariance between the column a of X1 and b of X2

COVARIANCE_PCA**

-assesses principal component analysis when knowing the covariance (of variables)

function[pcatype]=covariance_pca(covariance_type,(ncomponent))

Performs PCA on the covariance matrix as calculated by % "cumulate_covariance"

returns scores, eigenvector, eigenvalues, average of the observations

The input argument "covariance_type" is the result of the function cumulate_covariance

such as : covariance_type1=cumulate_covariance([],covariance_type);%% finishing

if ncomponent is defined, only this number of component is assessed.

Default: all

This function is useful fo to carrying on PCA with huge data set (see

cumulate_covariance for an example of use)

CREATE_GROUP**

Create a vector of number indicating groups from identifiers

function [saisir] = create_group(saisir1,code_list,startpos,endpos)

use the identifier for creating groups.

the codes of the groups are in code_list , and the positions where to find the letter

are in startpos and endpos

see also create_group1

CREATE_GROUP1***

function [saisir]=create_group1(s,startpos,endpos)

use the identifier for creating groups.

creates as many group means as different strings from startpos to endpos

s: saisir file, startpos and endpos : position of discriminating characters

CROSS_RIDGE_REGRESSION**

Ridge regression with crossvalidation

function[res]=cross_ridge_regression(x,y,krange,selected)

divides a collection in calibration and verification set

applies ridge_regression on the validation set

CROSSFDA1**

crossvalidation of discrimination according to fda1 (directly on data)

function[res]=crossfda1(X, group, among, maxvar, ntest)

applies fda1 by dividing X into calibration and verification set

ntest observations in each group are randomly (with no repeat) placed in test group.

applied directly on X data

maxvar is the maximum number of components introduced, among is the highest rank of the possibly introduced component.

CROSSMAHA1**

Crossvalidation on discrimination according to maha1 (*directly on data*)

function[disctype2]=crossmaha(X,group,maxvar,ntest)

Applies maha1 by dividing X into calibration and verification set

ntest observations in *each group* are randomly (with no repeat) placed in test group.

CROSSPLSDA **

cross-validation on PLS discriminant analysis

function[res]=crossplsda(x,group,dim,selected)

selected is a VECTOR with 0= calibration sample, 1= verification sample

CROSSVALPLS1a

crossvalidation of pls with up to ndim dimensions.

function [res]=crossvalpls1a(x,y,ndim,selected)

ndim= PLS dimension;

Performs a single crossvalidation test. "selected" is a vector giving the samples in verification set. 1= in verification; 0 = in calibration set

returns predicted y and observed y in cross validation obtain from 1 to

ndim dimension

returns also rmsev the root mean square error of crossvalidation and corr the correlation coefficient

CROSSVAL MULTIPLE REGRESSION

crossvalidation of multiple regression.

function[res]=crossval_multiple_regression(x,y,selected)

Performs a single crossvalidation test. "selected" is a vector giving the samples in verification set. 1= in verification; 0 = in calibration set

returns predicted y and observed y

returns also rmsev the root mean square error of crossvalidation and r2 the squared correlation coefficient

CROSSVAL QUADDIS

crossvalidation of quadratic dis. analysis

function res=crossval_quaddis(x,group,selected)

=====

Input args:

x : matrix of predictive data (n x p)

group : vector of known groups (integers, n x 1)

selected : MATLAB vector (n x 1)

with elements 0: selected in the calibration set

1: selected in the validation set

Output args res with fields:

calibration: structure quaddis_type as defined in function "quaddis"

validation : structure as defined in "apply_quaddis"

CUMULATE COVARIANCE *

-calculate covariance on huge data set

```
function [covariance_type]=cumulate_covariance(x,covariance_type1);
```

if status=0 or undefined calculate $X'X$ in X_X , Sum of row in SX ,

and n the number of processed observation

if the first argument =[], assess the covariance in the SAISIR format

single argument : starting the work

A complete script must therefore be something like

```
cov=cumulate_covariance(collection1);%% starting
```

```
cov=cumulate_covariance(collection2,cov);%% cumulating values
```

```
cov=cumulate_covariance(collection3,cov);%% cumulating values
```

...

```
cov=cumulate_covariance(collection_n,cov);%% cumulating values
```

```
covariance=cumulate_covariance([],cov);%% finishing
```

Cette fonction est surtout utilisée pour faire des analyses en composantes principales sur des données qui ne tiennent pas en mémoire vive, en combinaison avec la fonction `covariance_pca`.

D2 FACTORIAL MAP ***

Assesses a factorial map from a table of *squared* distances

```
function [ftype] = d2_factorial_map(saisir)
```

Uses the Torgerson approach to transform squared distance into pseudo scalar products.

DELETECOL ****

Deletes columns of `saisir` structures

```
usage: [X]= deletecol(X1,index)
```

The deleted columns are indicated by the vector `index` (numbers of Booleans)

DELETEROW ****

Deletes rows

```
usage: [X]= deleterow(X1,index)
```

The rows to be deleted are indicated by the vector `index` (numbers of Booleans)

DENDRO **

Dendrogram using euclidian metric and Ward linkage

```
function z=dendro(saisir,(topnodes))
```

Voir les fonctions dendrogram et linkage de MATLAB, pour plus de détails.

DIMCROSS STEPWISE REGRESSION **

Test models obtained from stepwise regression

```
function[res]=dimcross_stepwise_regression(x,y,selected,Pthres,(confidence))
```

divides the collection into calibration and verification samples using `selected`

in `selected`: 0 calibration samples; 1 verification samples

`Pthres` probability threshold of entering or discarding variable

`confidence` (optional): confidence interval for the correlation coefficient

DIMCROSSPCR1**

Crossvalidation of PCR (samples in validation are selected)

function [pcrres]=dimcrosspcr1(x,y,dimmax,selected)
dimmax= max number of PCR dimension;
selected: vector of samples selected as calibration set (==0)
and verification set (==1)
returns predy,rmsecv,corr
rmsecv is the root mean square error of crossvalidation
y must contain a single variable!
Components introduced in the order of the eigenvalues

DISTANCE **

Assesses the Euclidian distances between the two tables

function `D = distance(X1,X2)`

the tables must have the same number of columns

ELIMINATE NAN ***

Suppresses "not a number" data in a saisir structure

function `[saisir1] = eliminate_nan(saisir,(row_or_col))`

From a given saisir structure possibly containing NaN values (not determined values)
create a file of known values

if `row_or_col==0` tries to have the maximum number of rows

if `row_or_col==1` tries to have the maximum number of columns

ELLIPSE MAP

`ellipse_map` - plots the ellipse confidence interval of groups

(useful in discriminant analysis and related methods)

function `ellipse_map(X,col1,col2,gr,centroid_variability,(confidence),(point_plot))`

arguments:

- `X`: data matrix

- `col1, col2`: represented columns

- `gr`: qualitative groups (referred as integer number)

- `centroid_variability` : either 0 (variability of individual data points), or 1 : variability of the centroid itself (default: 0)

`confidence`: P value of the confidence interval (default: 0.05)

`point_plot`: if `point_plot ~0` , plots also the individual points as symbols (inactivated if `centroid_variability` set to 1)

EXCEL2BAG *

-read an excel file and creates the corresponding text bag (array of character)

function `[saisir,bag] = excel2bag(filename,ref_text_col,(nchar),(deb),(xend))`

the excel file includes the identifiers of rows and columns

`filename` :excelfile in the '.csv' format

`nchar` :number of characters read in each cell of the excel files
(the other are ignored)

`ref_text_col` :an array of string giving the reference of columns designed as forming the columns of bag.d. THESE COLUMNS ARE DESIGNATED USING THE EXCEL STYLE ('AA','AB')

`deb` : number of the first row decoded

`xend` : final row decoded

`output` : saisir follows the usual logic.

`bag` is a structure (`bag.d, bag.i, bag.v`), with `bag.d` being here a matrix of char

`saisir` contains the numerical values from the excel file, with the exception of the columns referenced by `ref_text_col`

`bag` contains the character values from the excel file, referenced by `ref_text_col`

Example:

```
>>[value,bag]=excel2bag('olive',['A'; 'B'],20)
```

the columns 'A' and 'B' from excel are read as text (in output bag)

the other columns are read as number respecting the saisir structure.

*Cette fonction est un peu complexe, elle est utile pour manipuler des fichiers Excels dans lesquels des groupes qualitatifs sont désignés par des chaînes de caractères.
Voir la fonction bag2group.*

EXCEL2SAISIR ****

Reads an excel file

function [saisir] = excel2saisir(filename,(nchar),(start),(xend))

reads an excel file which has been saved under the format .csv (the delimiters are ';')

the excel file includes the identifiers of rows and columns

start : number of the first row decoded, xend: final row decoded

FDA1 ***

stepwise factorial discriminant analysis on PCA scores

function[fdatatype]=fda1(pctype, group among, maxvar)

assesses a stepwise factorial discriminant analysis according to Bertrand et al., J of Chemometrics, Vol . 4, 413-427 (1990).

the basic idea is to assess a factorial discriminant analysis on the scores of a previous pca. The criterion of score selection is the maximisation of the trace.

group is the saisir file of group, among is the number of the first components

on which the selection is applied

maxvar is the maximum number of components possibly introduced in the model

FDA2 ***

stepwise factorial discriminant analysis on PCA scores with verification

function[fdatatype]=fda2(x,g,among, maxvar,selected)

criterion: maximisation of trace of $T^{-1}B$

selected : VECTOR indicating samples in calibration set(0), or in validation set (1);

FIND_INDEX

- find the index corresponding to the closest value

function index=find_index(str,value);

useful for finding the wavelength index in strings

Ne fonctionne que dans le cas ou str est une chaîne de caractères interprétable comme des nombres.

FIND_MAX

-gives the indices of the max value of a MATLAB Matrix

function [row,col,value]=find_max(matrix)

Attention! Il s'agit bien d'une matrice Matlab.

Exemple:

[row,col,value]=find_max(ble.d)

FIND_MIN

-gives the indices of the min value of a MATLAB Matrix

function [row,col,value]=find_max(matrix)

Attention! Il s'agit bien d'une matrice Matlab!

FIND_PEAKS

-find peaks greater than a threshold value

function vect=find_peaks1(saisir,row,threshold,gap,startvalue)

inside a window of size (data points) defined by `gap`. `gap` is a odd number
return the VECTOR of positions

GROUP CENTERING

Center data according to groups

```
function res=group_centering(X,group);
```

Centre les observations selon les groupes donnés par le fichier de groupe `group`.

Surtout utile pour centrer les notes par juge en analyse sensorielle.

ISI2SAISIR **

ISI2SAISIR1**

ISI2SAISIR2**

ISI2SAISIR3**

ISI2SAISIR4**

Load NIR data obtained with ISI software from disk

```
function [saisir] = isi2saisir?(filename)
```

The NIR data obtained from ISI software (on a NISsystems instruments) must be first converted into ASCII files by using the management program (option print data files) with printer off. In this way, it is possible to obtain ASCII files with the extension ".DAT". The program reads the files and give them the SAISIR format.

A few extra work is needed to adapt the program for other situation

En cas de difficultés, ces fonctions sont à essayer successivement.

LABELLED HIST ***

Draws an histogram in which each name is considered as a colored label

```
function labelled_hist(s,col,startpos,endpos,(nclass),(charsize))
```

with

`s` : a saisir structure

`col` : the column from which the histogram is drawn

`startpos` and `endpos`: the position in the row identifier strings considered as keys for

coloration

`nclass` : number of desired classes

`charsize` : the size of character on the graph

LEAVE ONE OUT PLS1

PLS1 with leave_one out validation
function res=leave_one_out_pls1(x,y,ndim);
A n'utiliser que sur des fichiers avec peu d'observations.

LEAVE ONE OUT PLS2

- PLS2 with leave_one out validation
function res=leave_one_out_pls2(x,y,ndim);
y may represent several variables
A n'utiliser que sur des fichiers avec peu d'observations.

LIST

Lists rows (only with a small number of columns)
function list(X)
Pour des fichiers de grande taille, il faut utiliser la fonction saisir2excel, et examiner le fichier sous Excel.

LR1

Assesses a Latent root 1 model
function [lr1type]=lr1(x,y,maxdim,ratioxy)
assess a basic lr1 model on files following the saisir format
returns lr1type
returns the individual predictors and the y predicted with 1 to maxdim latent variables
best choice obtained from a weighted sum of the predicted y according to *Vigneau et al.*
ratioxy(optional): positive number greater than 0 less than 1
giving the relative importance of x and y. 1: x important; 0 x not important
default : 0.5 (x and y have the same importance)

MAHA**

Simple discriminant analysis forward introducing variables no validation samples
function[discrtype]=maha(saisir,group,maxvar)
Assesses a simple quadratic discriminant analysis introducing up to maxvar variable
at each step, the more discriminating variable (according to the percentage of correct classification) is introduced. Only forward

MAHA1**

forward discriminant analysis DIRECTLY ON DATA with validation samples
function[discrtype1]=maha1(calibration,calibration_group,maxvar,test,test_group)
Assesses a simple linear discriminant analysis introducing up to maxvar variable
at each step, the more discriminating variable (according to the percentage of correct classification of the calibration set) is introduced. Only forward

MAHA3**

Simple discriminant analysis forward introducing variables no validation samples
function[discrtype]=maha3(saisir,group,maxvar)
assess a linear discriminant analysis introducing up to maxvar variable
at each step, the more discriminating variable
(according to the maximization of the trace of $T^{-1}B$ is introduced)

MAHA4**

Simple discriminant analysis forward introducing variables no validation samples
function[discrtype]=maha(saisir,group,maxvar)
Assesses a linear discriminant analysis introducing up to maxvar variable
at each step, the more discriminating variable
(according to the maximization of the number of correctly classified samples)

MAHA5**

Simple discriminant analysis forward introducing variables with validation samples
function[discrtype]=maha5(saisir,group,maxvar,selected)
Assesses a linear discriminant analysis introducing up to maxvar variable
at each step, the more discriminating variables are selected
according to the maximisation of the number of correctly classified samples

MAHA6**

Simple discriminant analysis forward introducing variables with validation samples
function[discrtype]=maha6(saisir,group,maxvar,selected)
Assesses a linear discriminant analysis introducing up to maxvar variable
at each step, the more discriminating variable
according to the maximization of the trace of $T^{-1}B$ is introduced
the collection is divided in cal. sample and test samples according to selected:
selected=0 , sample placed in calibration, =1 verification

MATRIX2SAISIR***

Transforms a MATLAB matrix in a saisir structure
saisir = matrix2saisir(data,coderow,codecol)
Saisir means "*statistique appliquée à l'interprétation des spectres infrarouge*"
or "*statistics applied to the interpretation of IR spectra*"
coderow or codecol indicates the characters before the rank number of the identifier
(optional)

MFA**

Multiple factor analysis
function res=mfa(collection)
collection : ARRAY OF CELL of SAISIR structure
example of building a collection : collection col{1}=table1;col{2}=table2; col{3}=table3
in which table1, table2, table3 are SAISIR structure
NOTE HERE THE USE OF { } (ACCOLADE) AND NOT () (BRACKET);
Each table must include the same observations, but not necessarily the same variables.
WARNING! In this version, the variables are not normalised !
Let n be the number of observations, and t the number of tables
Let WHOLE be the matrix of appended tables normalized according to MFA
(dimensions n x m)
Let k be the rank of WHOLE

Fields of the output :

score (n x k)	: scores of the individuals (compromise)
eigenvec (m x k)	: eigenvectors of the PCA on WHOLE (no direct use)
eigenval (1 x k)	: eigenvalues of the PCA on WHOLE

average (1 x m) : averages of the variables of WHOLE
 var_score (m x k) : scores of the variables
 proj {1xt cell} : projectors for computing the projection of new observations of each table
 first_eigenval (1xt) : first eigenvalues of the individual PCAS on each table
 observation_score (q x k) : individual score of each row of each table (q = total number of rows in all the tables)
 id_group (q x 1): identification of the belonging of the observation_score into a given table
 table_score (t x k) : scores of the tables
 Information on FMA can be found in SPAD TM Version 5.0 (procédure AFMUL);

MIR_STYLE *

Changes the sign of the variables of MIR spectra

function [names] = mir_style(names1)

In order to have the usual sense on the graph of mid infrared spectra

MULTIPLE REGRESSION

- Simple Multiple linear regression (all the variables)

function res=multiple_regression(x,y);

A single y variable !!!!

A utiliser seulement dans des situations dans lesquelles les colonnes de X ne sont pas quasi-colinéaires ou colinéaires.

MULTIWAY PCA **

Multi way principal component analysis

Function res=multiway_pca(collection);

collection : ARRAY OF CELL of SAISIR structure

example of building a collection : collection col{1}=table1;col{2}=table2; col{3}=table3

in which table1, table2, table are SAISIR structure

NOTE HERE THE USE OF {} (ACCOLADE) AND NOT () (BRACKET);

Each table must include the same observations, but not necessarily the same variables.

WARNING! In this version, the variables are not normalised !

Let n be the number of observations, and t the number of tables

Let WHOLE be the matrix of appended tables

Let k be the rank of WHOLE

Fields of the output :

=====

score (n x k) : scores of the individuals (compromise)
 eigenvec (m x k) : eigenvectors of the PCA on WHOLE (no direct use)
 eigenval (1 x k) : eigenvalues of the PCA on WHOLE
 average (1 x m) : averages of the variables of WHOLE
 var_score (m x k) : scores of the variables
 proj {1xt cell} : projectors for computing the projection of new observations of each table
 trajectory (q x k) : individual score of each row of each table (q = total number of rows in all the tables)
 id_group (q x 1) : identification of the belonging of the observation_score into a given table

`table_score (t x k)` : scores of the tables

Cette fonction (experimentale) permet de traiter des tableaux multiples avec une approche très simple, plus logique que celle de la fonction MFA.

MSC*

Applies a multiplicative scatter correction on spectra

function `[x] = msc(x1,(reference))`

multiplicative scatter correction, reference is the reference spectrum

If only one argument, the reference is the average spectrum

NORM COL**

Divides each column by the corresponding standard deviation

function `[saisir] = norm_col(saisir1)`

NORMED PCA****

PCA with normalisation of data

function `[pcatype]=normed_pca(saisir)`

NUEE **

Nuee dynamique (Kcmeans)

function `[res]=nuee(X,ngroup)`

Clusters the data into ngroup according to the KCmeans method ("nuée dynamique");

NUM2STR1 *

Justified num2str

function `str=num2str1(number,ndigit);`

Exemple

`num2str1(12,5)`

00012

La sortie est une chaîne de caractère. Utile pour créer des champs dans les noms d'individus.

PCA ****

Assesses principal component analysis on raw data

function `[pcatype]=pca(saisir,info)`

Assesses principal component analysis (on not normalised data)

returns coord, eigenvector,eigenvalues, average.

`pcatype` is a structure containing

scores (coordinates)

`eigenvec` :eigenvectors (loadings)

`eigenval` :eigenvalues

`average` :average observation

PCA CANO **

Generalized canonical analysis after PCAs on each table

```
function res=pca_cano(col,ndim,graph)
```

```
function res=pca_cano(col,ndim,graph);
```

input argument :

ndim : dimension of each individual PCA (must be less than the smallest number of variables)

graph : if different from 0 : display examples of graph

fields of the output:

let n be the number of observations in each table, k the number of tables

compromise : PCA giving the compromise

observation_score : scores of each observation of each table (nxk rows)

id_group : groups identifying each observation in observation_score (for graph)

projector : struct array giving the vectors allowing the projection of each data set

score_correlation:correlations between compromise and each scores of each table

table_average : struct array giving the average of each original data table

Adapted from : G. Saporta. Probabilités, analyse des données et statistiques.

: Edition Technip, page 192 and followings.

La fonction effectue l'analyse canonique sur la collection de tableau col après des ACP tableau par tableau. Les coordonnées factorielles des différents tableaux sont ensuite concaténées en ligne, en ne tenant compte que de *ndim* dimensions. Soit **C** ce tableau.

C est alors réduit (divisé par les racines carrées des valeurs propres correspondantes), puis une autre ACP est effectuée sur **C** réduit.

Le fichier `res.compromise.score` contient les coordonnées factorielles de cette ACP; qui sont les points compromis de chaque observation vue par tous les tableaux.

On peut projeter individuellement chaque observation de chaque tableau dans un espace factoriel commun. `projector` donne, tableau par tableau, la matrice des projecteurs (transformations linéaires) utilisable sur chaque tableau d'origine centré. Pour pouvoir éventuellement projeter les individus supplémentaires d'un tableau donné, les moyennes des colonnes de chacun des tableaux sont conservées dans `res.table_average`.

`Observation_score` donne les scores des observations tableau par tableau (*n x k* lignes).

Pour pouvoir effectuer des représentations barycentriques, `id_group` contient les numéros des rangs des individus dans chacun des tableaux (*n x k* lignes, 1 colonne).

Exemple d'utilisation:

```
%% fichier de démonstration de PCA_cano
%% analyse canonique sur coordonnées d'ACP
load ble;
%% on simule un système multitableau
clear col;
for i=1:4
    col(i)=selectcol(ble,((i-1)*250+1):i*250);%
end

res=pca_cano(col,10,1);%% version avec graphique: dernier paramètre =1;
figure;
```

```
tcurve(res.projector(1),1);% examen d'un projecteur %% ou
proj=[]; for i=1:
4
    proj=appendrow1(proj,res.projector(i));% création d'une courbe (continue ?) de tous les
projecteurs
end tcurve(proj,1);
```

PCA CROSS RIDGE REGRESSION **

PCA ridge regression with crossvalidation
function[res]=cross_ridge_regression(x,y,krange,selected) divide
a collection in calibration and verification set
apply ridge_regression on the validation set
some trick about range (which is the cutting component) *see*
PCA_ridge_regression

PCA RIDGE REGRESSION **

Assess a basic ridge regression after PCA
function[ridgetype]=pca_ridge_regression(pcatype,y,range) ONLY
ONE VARIABLE TO BE PREDICTED (scan the dimensions) return as
many beta as the number of elements in range
warning! range is NOT a range of kvalue it is a
rank of component.
k is in fact the eigenvalue of the corresponding PCA component
The rationale of this, is that k in ridge_regression is very difficult to find it is a
good idea to test a value in the range of the observed eigenvalues

PCARECONSTRUCT *

Reconstructs original X data from PCA and a file of score
function[res]=pcareconstruct(pcatype,score,maxdim)
from the previously assessed score rebuild the original data matrix.
the precision of the reconstruction is depending on the number of introduced components in
maxdim)

PCA STAT

- Gives some complementary statistiques on PCA observations
function res=pca_stat(pca_type, col1, col2);

Input argument :

pca_type: result of function pca

comp1, comp2: number of the PC components of the PCA to be analyzed Ouput

:

Saisir Matrix with 7 columns : QTL, col1 CO2col1, CTRcol1, col2, CO2col2, CTRCol2 QLT :
squared cosinus with the plan (quality of the representation of the observations CO2col1 and CO
2col2 : squared cosinus of the angle between the observation and the axis We have QLT=CO2col
1 + CO2col2

CTRcol1 and CTRcol2 : Contribution of the observation to the component.

From G.Saporta, Probabilités analyse des données et statistiques, Ed Technip, page 182

PCR***

Assesses a basic model of PCR (components introduced in the order of eigenvalues

function [pcrtype]=pcr(x,y,maxdim)

returns pcrtype: pca, beta, predy, correlation coeff with 1 to maxdim elements, averagey
ONLY ONE VARIABLE TO BE PREDICTED (scan the dimensions)

PCR1 **

Assess a basic model of PCR (components introduced in the order of eigenvalues

function [pcrtype]=pcr1(x,y,dim)

returns pcrtype: pca, beta, predy, correlation coeff with dim elements, averagey
SEVERAL VARIABLES CAN BE PREDICTED (no scan of the dimensions)

PLOTMATRIX1

-biplots of columns of matrices with colors

function plotmatrix1(s,startpos,endpos,charsize)

Même logique que la commande Matlab plotmatrix, mais avec des identificateurs en couleur.

PLSDA ***

Assesses pls discriminant analysis following the saisir format

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

returns (in structure type):

beta :coeff for predicting the indicator matrix

beta0 :intercept for predicting the indicator matrix

t :PLS latent variable

predy :predicted indicator matrix

classed :predicted groups according to method #0

ncorrect :number of rightly classified samples according to method #0

(attribution to index of max of predicted Y)

confusion :confusion matrix according to method #0

ncorrect1 :number of rightly classified samples according to method #1

(mahalanobis distance on latent variable t)

confusion1 :confusion matrix according to method #1

tbeta :coeff for predicting the latent variables t

tbeta0 :intercept for predicting the indicator matrix

linear :linear form for direct prediction of group

linear0 :%the min of x'*linear + linear0 gives the predicted group

QUADDIS **

- Quadratic discriminant analysis

function quadis_type=quaddis(x,group);

Quadratic discriminant analysis

(Training)

A multinormal distribution is assumed in each qualitative group

Input args:

x: predictive data set (matrix n x p)

g: qualitative groups (matrix n x 1) with integer ranging from 1
to maximum number of groups (gmax)

Output args:

quaddis_type with fields:

ncorrect100: percentage of correct classification (number)
confus : confusion matrix (gmax x gmax)
mean : means according to each group (gmax x p)
predgroup : predicted groups (integer) (n x gmax)
density : pseudo-densities of each observation (n x gmax)
proba : probability of belonging to a given group (n x gmax)

model : predictive model with fields:

inv: Matlab matrices of Mahalanobis metrics (cube p x p x gmax)
Mut: Matlab matrices of means according to each group (gmax x p)
det: Matlab vector of determinants of covariance matrices of each group (1 x gmax)

See also apply_quaddis, crossval_quaddis

QUICKPLS

Quick PLS regression from 1 to ndim dimensions

function [plstype]=saisirpls(x,y,ndim)

assesses a pls1 model

Only a single variable can be predicted, but with all the dimensions computed

Rapide, mais n'utilise pas l'algorithme NIPALS de référence

RANDOM_SELECT***

Random selection of samples

function[selected]=random_select(nel, nselect, (nrepeat))

returns a vector containing 1 in random position

among nel elements, nselect elements are set to 1

nrepeat (optional) randomly selects nselect values, but organised by block of nrepeat groups. For example, if nrepeat =3 a possible result is [0 0 0 1 1 1 0 0 0 1 1 1 1 1 1 ...]

This is useful for dividing a collection in calibration and verification set when the repetitions are consecutive.

Normally nselect is an integer multiple of nrepeat.

RANDOM_SPLITROW *

Random selection of samples

function[X1,X2]=random_split(X, nselect)

randomly divide a collection in two collections

nselect samples in X1 and n-nselect samples in X2

where n is the number of rows in X

RANDOMIZE *

Build a file of randomly attributed rows

function X1=randomize(X)

the row vector in X are randomly attributed to each obs.

X may be the result of function 'create_group1'

Peut être utile lorsque l'on veut comparer une prédiction à celle obtenue par hasard.

READIDENT **

Loads a file of strings

usage: [res,nindent]=readident(filename, (namesize))

loads an array of string in a matrix format

namesize gives the maximum number of characters in each string

REGRESSION SCORE

- build a factorial space for regression

function res=regression_score(x,beta,(y))

Input arguments:

x : data matrix (n x p)

beta : vector of regression coefficients (p x 1)

y :(optional) known y value

Output argument:

res with fields

score : regression scores (also y split)

reconstructed_norm2: squared norms of the scores

cumulated_norm2: cumulated squared norms of the scores

projector: matrix such as score=x*projector

eigenvec_sum: sum of the eigenvectors of PCA (linked to the theory)

xmean: mean of x

r2 : if (y defined) r2 of the cumulated model

Given a data matrix x and the regression coefficients beta,

the function build up a matrix of orthogonal scores

such as predicted y is equal to the sum of this scores

The scores can be used to examine the observations "oriented" in the prediction of y.

As the scores are ranked as a function of their ability to predict y,

It is possible to examine the observations beginning by the first scores.

Cette fonction peut être utile pour créer un espace factoriel "orienté" pour prédire y.

Cette fonction demande en entrée des coefficients de régression beta calculés précédemment (par PLS, ridge régression ou autre).

REPEAT STRING

build a matrix of char by repeating a string

function str1=repeat_string(str,ntimes);

Input arg:

- str : a character string

- ntimes: number of repetition

Output arg:

- str1 the matrix of char with the repeated string.

example:

>> repeat_string('Vanessa',3)

ans =
Vanessa
Vanessa
Vanessa

Utile pour manipuler des noms.

REDUCE NVARIABLE **

Reduces variable number with averaging

[X] = reduce_nvariable(X1,nvar)

From the original data, averages *nvar* neighbouring columns.

The number of resulting variables is roughly equal to *nvariable/nvar*

A few variables (less than *nvar*) might be lost at the end of the data

nvar is preferably an odd value

REORDER ****

Reorder the data of files *s1* and *s2* according to their identifiers

function [B1 B2]=reorder(A1,A2)

This function makes it possible to realign the rows of *A1* and *A2*, in order to have the identifiers corresponding.

This is necessary for any predictive method (particularly regressions).

The function discards the observations which are not present in *A1* and *A2*.

The matrix *B1* corresponds to *A1* and matrix *B2* to *A2*

Fails if *A1* or *A2* contains identical identifiers of rows.

RIDGE REGRESSION ***

Assesses a basic ridge regression

function [ridgetype]=ridge_regression(x,y,krange)

ONLY ONE VARIABLE TO BE PREDICTED (scan the dimensions)

return as many *beta* as the number of elements in *krange*

RIDGE REGRESSION1

Basic ridge regression at a given norm

function [ridgetype]=ridge_regression1(x,y,normrange)

ONLY ONE VARIABLE TO BE PREDICTED (scan the dimensions)

return as many *beta* as the number of elements in *normrange*

ROW CENTER

- subtracts the average column to each column

function [X] = center(X1)

SAISIR_CHECK ****

Check if the data respect the saisir structure

function check=saisir_check(X)

check = 1 if x is in the SAISIR format (no warning)

check = 2 if x is in the SAISIR format (with warning)

check = 0 if x is not in the SAISIR format (fatal error)

Cette fonction permet de tester si un fichier est bien au format de SAISIR.

Indique une '*fatal error*' si quelque chose est incorrecte.

Un '*warning*' si les données peuvent poser des problèmes dans les autres fonctions.

SAISIR2EXCEL ****

Saves a saisir file in a format compatible with Excel

function saisir2excel(X,filename)

Transforms a saisir file into a .CSV file and saves it on disk

This file can then be read as an excel file (separator : ";")

SAISIR_LINKAGE *

Assesses a simple linkage vector from a matrix of distance

function z=saisir_linkage(dis)

Extracts the "unfolded" triangular matrix in order to enter the MATLAB program linkage with the option ward

Returns the Z vector as required by dendrogram function

SAISIR_MEAN **

Assesses the mean of the columns, following the saisir format

function[average]=saisir_mean(X)

SAISIR_DERIVATIVE

n-th order derivative using the Savitzky-Golay coefficients

[saisir]=saisir_derivative(saisir1,polynome_order>window_size,derivative_order)

: example res=saisir_derivative(x,3,21,2);

Compute the second derivative using a polynom of power 3 as model and a window size of 21

SAISIR_SORT *

- sorts the rows of s according to the values in a column

function [s1 s2]=saisir_sort(s,col)

s1 : data sorted according to the column col

s2 : a column rank is added (fro using graph functions)

SAISIR_STD **

Assesses the standard deviations of the columns, following the saisir format

function[st]=saisir_std(saisir)

SAISIR2ASCII *

- Saves a saisir file into a simple ASCII format
function saisir2ascii(X,filename,separator)
Transform a saisir file into a simple .txt file and save it on disk
separator is a single character like ' ' or ';' or its ASCII code;

SAISIR_TRANSPOSE **

Transposes a data matrix
function [saisir] = saisir_transpose(saisir1)

SAISIRPLS ***

PLS regression with dim dimensions
function [plstype]=saisirpls(x,y,dim)

returns plstype.beta, plstype.beta0, plstype.predy

SDIMCROSSPCR1 **

Crossvalidation of stepwisePCR
function [pcrres]=sdimcrosspcr1(x,y,dimmax,maxrank,selected,(corr_cov))
(samples in validation are selected)
dimmax : max number of PCR dimension;
maxrank : the selection of components is limited to the first maxrank components
selected : vector of samples selected as calibration set (==0)
and verification set (==1)
returns predy, rmsecv, corr
rmsecv is the root mean square error of crossvalidation
y must contain a single variable!
corr_cov: =1: introduction of components in the order of r2 between y and component
corr_cov: =0: introduction in the order of covariance between y and component

SEEKSTRING ***

Returns a vector giving the indices of string in matrix of char x in which 'str' is present
function index = seekstring(x,str1)
Cette fonction sert essentiellement à déterminer le numéro d'ordre d'une variable ou d'une observation dont on connaît l'identificateur.

SELECT_FROM_IDENTIFIER ***

Uses identifiers of rows for selecting samples
function [X1] = select_from_identifier(X,startpos,str)
Creates the data collection X1 which is the subset of X
the identifiers of which contain the string str, in starting position startpos

SELECT_FROM_VARIABLE **

use identifier of columns for selecting variables
function [saisir1] = select_from_variable(saisir,startpos,str)
Creates the data collection "saisir1" which is the subset of "saisir"
the names of variables of which contain the string str, in starting position startpos

SELECTCOL ****

Creates a new data matrix with the selected columns

function `X1= selectcol(X,index)`

The resulting file correspond to the selected columns

`index` is a vector of indices (integer) or of booleans

SELECTROW ****

Creates a new data matrix with the selected rows

function `[X1]= selectrow(X,index)`

The resulting file correspond to the selected rows

`index` is a vector of indices (integer) or of booleans

SENSO ANOVAN

2-way analysis of variance (ANOVA) on judge x product data matrices.

function `res = senso_anovan(saisir,judge,product)`

Performs as many N-way analyses of variance as the number of columns in X

JUDGE ARE SEEN AS RANDOM EFFECT.

L'ordre *paramètre 2 = juge, paramètre 3 = produits* doit être respecté!!

SENSORY PROFILE

Graphical representation of sensory profile

function `[h]=sensory_profile(X,range,max_score,(title))`

Graphical display of sensory profiles in a "circular" representation.

-X : matrix of data to be displayed

-range : vector of the indices of the rows to be displayed

-max_score : maximal score used in the scale

-title : (optional) title of the graph.

Warning: will not work properly with more than 15 variables

Preferably reduce the identifiers of variables to 3-4 characters

SHOW VECTOR ***

Represents a row of a matrix as a succession identifiers

function `handle=show_vector(X,nrow ,(xlab),(ylab),(title))`

The identifiers of the columns are plotted with X being the index of the variable and Y the actual value of the variable for the selected row `nrow`

`xlab, ylab, title`: optional xlabel, ylabel, title.

SIMPLE REGRESSION

mono_linear regressions

function `[beta beta0]=simple_regression(X,y);`

y is predicted by each column i of X according to $y_{pred}=X.d(:,i)*beta.d(i)+beta0.d(i);$

Thus as many models as the number of columns in X

SMART COORD

- Smart coordinates

function res=smart_coord(x);

Multiply the columns of x by 1 or -1 in order to have the maximum absolute value positive

Useful for an easy comparison of several factorial maps (PCA or others)

Two maps rotationnaly equivalent will be identical after this modification

Utile pour orienter des cartes factorielles identiques dans le même sens.

Attention ! Cette fonction ne fait pas “suivre” logiquement les vecteurs projecteurs tels que les vecteurs propres d’une ACP

SNV **

Applies a standard normal variate correction on spectra

function [X1] = snv(X)

SNV is commonly used in spectroscopy. It basically consists in centering and standardizing the ROWS (not the columns) of the data matrix.

This procedure may reduce the scatter deformation of spectra

SPCR **

Stepwise Principal component regression

function [spcrtype]=spcr(x,y,maxdim, (maxrank)(corr_cov))

Calculates a PCR model

returns spcrtype: pca, beta, selected_component, predy, correlation coeff with 1 to maxdim elements, averagey

The components are introduced in the order of their regression coefficient or their covariance with y

maxdim: maximal number of components in the model

maxrank: (optional) maximal rank of the components possibly introduced in the model (the components of higher rank are not considered). Default value: all components possibly introduced.

corr_cov : 1 introduction according to correlation coeff (corr_cov=1,default);

or : 0 introduction according to covariance

SPLIT AVERAGE **

Averages observations according to the identifiers

function average_type=split_average(X,startpos,endpos)

Uses the identifier for averaging

Creates as many average as different strings from startpos to endpos

Returns average_type.average and average_type.effectif: number of obs. in each group.

SPLITROW *

split a data matrix into 2 resulting matrices

function: [X1, X2]= splitrow(X,index)

Divides X into two matrices X1 and X2

The first one (X1) correspond to kept rows (according to index)

The second one is the complement

index is either a vector of indices of the rows (integers) or of Booleans.

SPLIT STD

Computes the standard deviation according to the identifiers

function `std_type=split_std(X,startpos,endpos,(option))`

Use the identifier for computing standard-deviation

Creates as many vector of standard deviation as different strings from startpos to endpos

Returns `std_type.std` and `std_type.g`: effectif in each group .

option: 0 : division by N-1; 1: division by N (default : 0)

STANDARDISE *

Divides each column of a matrix with the corresponding standard deviation

function `[X1]=standardize(X2)`

STATIS **

STATIS - Multiway method STATIS

function `res=statis(col);`

The STATIS method is described in "C.Lavit, Analyse conjointe de tableaux qualitatif, Masson pub, 1988."

Basically the method attempts to establish a factorial compromise between table having the same number of observations.

`col` is an **array of cells** containing all the 2-D data tables (SAISIR format).

Each table must include the same observations, but not necessarily the same variables.

Let n be the number of observations, and t the number of tables

the field of the output argument `res` are:

-RV: [1x1 struct] matrix $t \times t$ of the RV value indicating the agreement between the table (max value = 1)

-eigenval1: [1x1 struct] first eigenvalue of the RV matrix

-eigenvec1: [1x1 struct] first eigenvector of the RV matrix ($t \times 1$) . Indicates the weight associated with each table

-Wk: {1x t cell} cell of $n \times n$ array giving the scalar products between observations

-W_compromise $n \times n$ array giving the compromise of the array WK

-eigenval2: [1x1 struct] r eigenvalues of W_compromise, with r the rank of the W compromise.

-observation_score: [1x1 struct] ($n \times r$) Scores of the compromise of the observations. Can be represented as factorial map

-trajectory: [1x1 struct] ($n \times t \times r$) Projection of each row vector of each table in the space of observation_score

-group: [1x1 struct] ($n \times t \times 1$) table giving the belonging of a given row_vector to a table

group is useful with the command 'carte_barycentre'

For example, a command such as "carte_barycentre(res.trajectory,2,3, res.group)" will produce the representation of the row vector of each table for the score 2 and 3. The representation shows the compromise point and its link to each vector of the tables.

- table_score: [1x1 struct] (t x r) scores of the tables obtained from diagonalisation of RV.
-table_eigenval: [1x1 struct] (r x 1) eigenvalues of RV . The first one is the same as eigenval1

STEPWISE REGRESSION ***

Stepwise regression between x and y

function[result]=stepwise_regression(x,y,Pthres,(confidence))

Pthres is the probability threshold for entering or discarding a variable

confidence (default=0.05) is the probability of the confidence interval for the regression coefficients

The function returns an array of cells corresponding to each step of the regression

message : gives the name of the entered or discarded variable'

res : a structure described below

intercept : constant value (beta0) of the current model

RMSE : root mean square error of the model

r2 : determination coefficient

adjusted_r2 : adjusted determination coefficient (taking into account of the dimensions

F : Fisher F value of the current model

probF : probability value associated with F

ypred : predicted y values

res : the rows indicate the variables introduced
the columns give information on the corresponding regression coefficients:

regression coefficients

Lower confidence limit

Higher confidence limit

Std of regression coeff.

t value of reg. coeff.

Prob. of reg. coef.

Rank of variables

STRING2SAISIR *

creation of a saisir file from a string table (first column=name)

function [saisir] = string2saisir(data)

Cette fonction n'a pas d'utilisation directe.

STRING2TEXT *

save a vector of string in a .txt format

function string2text(str,filename)

SUBMAP *

Represents only the observations whose identifiers contains the string xstring
function `submap(X,col1,col2,xstring,labcol1,labcol2,titre,charsize,marg)`
The scale of the COMPLETE map is used.

Utile lorsque l'on veut faire des représentations factorielles d'une partie des observations.

SUBTRACT VARIABLE *

Subtracts a given variable to all the others

function `[saisir] = subtract_variable(saisir1,ncol)`

Subtract the variable of index ncol to each other variables of the observations

SURFACE1 *

Represent a surface in three dimensions

function `[zmin, zmax]=surface1(saisir)`

SURFACE STD *

Divides each row by the sum of the corresponding columns

function `[X1] = surface_std(X)`

TCURVE **

Representation of a column of a given matrix

function `tcurve(X, ncol, (xlabel),(ylabel),(title))`

TCURVES

Represents several columns of a matrix as curves

usage `handle=Tcurves(X, select_col, (xlabel),(ylabel),(title))`

`select_col` gives the indices of the selected columns

TRAJECTORY CURVE

Plots coloured XYcurves

function `handle=trajectory_curve(x,col1,col2,startpos,endpos);`

The function represents the columns `col1` and `col2` as curves (ark)

The observations which have the same strings in their identifiers are joined.

The points are joined consecutively according to their order (rank) in `x`.

Input arguments:

`x` : saisir matrix

`col1, col2` : columns of `x` to be represented (integer number).

`startpos, endpos` : positions in identifiers indicating which identifiers are to be joined.

exemple of use : time series

identifiers of rows (`x.i`) like A01; A02; A03... A100; B01 ... B100; C01 ...

with 1 ... 100 indicating times, and A B .. observations varying with time

Command:

`trajectory_curve(x,1,2,1,1);`

Join the point labelled 'A' together; the ones labelled 'B' ... and so on

XY_PLOT

Biplot of one column of X versus one column of Y

```
function handle=xy_plot(X, xcol, X, ycol,start_pos,end_pos);
```

X, Y : saisir files

xcol, ycol : rank number of the columns to be plotted

If start_pos and end_pos defined: plot colored according to the characters of the identifiers of rows in position start_pos:end_pos.

5 Liste thématique des fonctions disponibles

Seuls les noms des fonctions et un court descriptif sont donnés ci-dessous. Une information plus complète peut-être obtenue en consultant le dictionnaire (paragraphe 4) ou par la commande `help` de MATLAB, suivi du nom de la fonction.

On retrouvera cette liste en tapant « `help saisir` » sous MATLAB.

5.1 Chargement et sauvegarde des fichiers

[EXCEL2SAISIR](#) ****

Reads an excel file

[EXCEL2BAG](#) *

-read an excel file and creates the corresponding text bag (array of character)

[ISI2SAISIR](#) **

[ISI2SAISIR1](#)**

[ISI2SAISIR2](#)**

[ISI2SAISIR3](#)**

Load NIR data obtained with ISI software from disk

[MATRIX2SAISIR](#)***

Transform a MATLAB matrix in a `saisir` structure

[READIDENT](#) **

Loads a file of strings

[SAISIR_CHECK](#) ****

Check if the data respect the `saisir` structure

[SAISIR2EXCEL](#) ****

Saves a `saisir` file in a format compatible with Excel

[SAISIR2ASCII](#) *

Saves a `saisir` file into a simple ASCII format

[STRING2SAISIR](#) *

creation of a `saisir` file from a string table (first column=name)

[STRING2TEXT](#) *

save a vector of string in a `.txt` format

5.2 Manipulations élémentaires des données

[APPENDCOL](#) ****

Merge two files according to columns

[APPENDCOL1](#) ****

Merge an arbitrary number of files according to columns

[APPENDROW****](#)

Merge two files according to rows

[APPENDROW1****](#)

Merge an arbitrary number of files according to rows

[CHANGE_NAME](#)

Change the name of the observations of file X

[CREATE_GROUP**](#)

Create a vector of number indicating groups from identifiers

[CREATE_GROUP1***](#)

Create a vector of number indicating groups from identifiers

[DELETECOL ****](#)

Deletes columns

[DELETEROW ****](#)

Deletes rows

[DISTRIBUTE_DATA](#)

distribute the values of model according to row_identifier

[ELIMINATE_NAN1 ***](#)

Suppresses "not a number" data in a saisir structure

[MATRIX2SAISIR***](#)

Transform a MATLAB matrix in a saisir structure

[RANDOM_SELECT***](#)

Random selection of samples

[RANDOM_SPLITROW *](#)

Random selection of samples

RANDOMIZE *

Build a file of randomly attributed rows

REORDER ****

Reorder the data of files s1 and s2 according to their identifiers

SAISIR TRANSPOSE **

Transposes a data matrix

SEEKSTRING ***

Returns a vector giving the indices of string in matrix of char x in which 'str' is present

SELECT FROM IDENTIFIER ***

Uses identifiers of rows for selecting samples

SELECT FROM VARIABLE **

use identifier of columns for selecting variables

SELECTCOL ****

Creates a new data matrix with the selected columns

SELECTROW ****

Creates a new data matrix with the selected rows

SPLIT AVERAGE **

Averages observations according to the identifiers

SPLITROW *

split a data matrix into 2 resulting matrices

5.3 Transformation élémentaires des données

CENTER *

subtracts the average row to each row

CORRECT BASELINE*

simple linear baseline correction, using intensity

DERIVATIVE2

Simple computation of second derivative

SAISIR DERIVATIVE

n-th order derivative using the Savitzky-Golay coefficients

DISTRIBUTE DATA

distribute the values of model according to row_identifier

GROUP CENTERING

Center data according to groups

MSC*

Applies a multiplicative scatter correction on spectra

NORM_COL**

Divides each column by the corresponding standard deviation

REDUCE_NVARIABLE**

Reduces variable number with averaging

ROW_CENTER

Subtracts the average column to each column

SAISIR_SORT*

- sorts the rows of **S** according to the values in a column

SAISIR_TRANSPOSE**

Transposes a data matrix

SMART_COORD

- Smart coordinates

SNV**

Applies a standard normal variate correction on spectra

SPLIT_AVERAGE**

Averages observations according to the identifiers

SPLIT_STD

Computes the standard deviation according to the identifiers

STANDARDISE*

Divides each column of a matrix with the corresponding standard deviation

SUBTRACT_VARIABLE*

Subtracts a given variable to all the others

SURFACE_STD*

Divides each row by the sum of the corresponding columns

5.4 Représentations graphiques

BROWSE***

Browses a series of curves

CA MAP

Représentation factorielle de l'analyse des correspondances (lignes et colonnes de la matrice)

CARTE****

Graph of map of data using identifiers as names

CARTE3D ***

Draw a 3D map

CARTE3D COULEUR1 ***

Draw a colored 3D map

CARTE BARYCENTRE**

Graph of map showing the barycentre of qualitative groups

CARTE COULEUR1****

colored map : using a portion of the identifiers as labels

CARTE COULEUR2****

Colored map : using a portion of the identifiers as labels

CARTE COULEUR4 ****

Colored map according to given choice

CARTE SYMBOLE ****

Map with symbols : using a portion of the identifiers for

COL MAP1

Colored map : using a portion of the identifiers as labels

COLOURED CURVES

- displays curves coloured according to groups

CORRELATION CIRCLE **

Assesses and displays the correlation circle after PCA

CORRELATION PLOT**

- Draw a correlation between scores and tables

CURVE****

Represent a row of a matrix as a curve

CURVES****

Represents several rows of a matrix as curves

ELLIPSE MAP

Plot the ellipse confidence interval of groups

FIND PEAKS**

Represents a curve showing its peaks

IMAGE1 **

Represent a surface in false color

LABELLED HIST ***

Draws an histogram in which each name is considered as a colored label

LIST

Lists rows (only with a small number of columns)

MIR STYLE *

Changes the sign of the variables of MIR spectra

PLOTMATRIX1

-biplots of columns of matrices with colors

SHOW VECTOR ***

Represents a row of a matrix as a succession identifiers

SUBMAP *

Represents only the observations whose indentifiers contains the string xstring

SURFACE1 *

Represent a surface in three dimensions

TCURVE **

Representation of a column of a given matrix

TCURVES

Represents several columns of a matrix as curves

TRAJECTORY CURVE

Plots coloured XYcurves

XY PLOT**

Biplot of one column of X versus one column of Y

5.5 Méthodes statistiques

5.5.1 Analyse de variance et statistiques élémentaires

ANAVAR1****

One way analysis of variance on spectral data

ANOVAN1****

N-way analysis of variance (ANOVA) on data matrices.

CONTINGENCY KHI2**

compute khi2 stats on a contingency table

CONTINGENCY TABLE**

Compute a contingency table

CORMAP***

Assesses the correlation between two tables

COVMAP **

Assesses the covariances between two tables

DISTANCE **

Assesses the Euclidian distances between the two tables

DISTANCE STRESS *

Estimates the stress (default of the reconstruction of distances)

SAISIR LINKAGE *

Assesses a simple linkage vector from a matrix of distance

SAISIR MEAN **

Assesses the mean of the columns, following the saisir format

SAISIR STD **

Assesses the standard deviations of the columns, following the saisir format

SENSO ANOVAN

2-way analysis of variance (ANOVA) on judge x product data matrices.

5.5.2 ACP

[APPLYPCA**](#)

Assess the scores of supplementary observations

[CHANGE_SIGN*](#)

Changes the sign of a component and of associated eigenvector

[CORRELATION_CIRCLE**](#)

Assesses and displays the correlation circle after PCA

[COVARIANCE_PCA**](#)

-assesses principal component analysis when knowing the covariance (of variables)

[CUMULATE_COVARIANCE*](#)

calculate covariance on huge data set

[MULTIWAY_PCA**](#)

Multi way principal component analysis

[NORMED_PCA****](#)

PCA with normalisation of data

[PCA****](#)

Assesses principal component analysis on raw data

[PCARECONSTRUCT*](#)

Reconstructs original X data from PCA and a file of score

[PCA_STAT](#)

- Gives some complementary statistiques on PCA observations

[SMART_COORD](#)

- Smart coordinates

5.5.3 Régressions

[APPLY MULTIPLE REGRESSION](#)

Applies multiple_regression on "unknown" data

[APPLY RIDGE REGRESSION](#) **

Apply ridge regression on "unknown data"

[APPLY STEPWISE REGRESSION](#)**

Apply stepwise_regression on "unknown" data

[APPLYLR1](#)**

Apply basic latent root model on saisir data x

[APPLYPCR](#)**

Assesses a basic PCR on data

[APPLYPLS](#)**

Applies a pls model on an unknown data set

[APPLYSPCR](#)*

Applies a stepwise PCR

[BASIC PLS](#)**

Basic pls with keeping loadings and scores

[BASIC PLS2](#)

Basic pls2 with keeping loadings and scores

[CROSS RIDGE REGRESSION](#)**

Ridge regression with crossvalidation

[CROSSVAL MULTIPLE REGRESSION](#)

crossvalidation of multiple regression.

[CROSSVALPLS1a](#) **

crossvalidation of pls with ndim dimensions.

[DIMCROSS STEPWISE REGRESSION](#) **

Test models obtained from stepwise regression

[DIMCROSSLR1a](#) **

Cross validation of latent root model LR1

[DIMCROSSPCR1](#)**

Crossvalidation of PCR (samples in validation are selected)

[DIMCROSSPLS1a**](#)

crossvalidation of PLS (changing the dimensions)

[LEAVE ONE OUT PLS2](#)

- PLS2 with leave_one out validation

[LR1](#)

Assesses a Latent root 1 model

[PCA CROSS RIDGE REGRESSION **](#)

PCA ridge regression with crossvalidation

[PCA RIDGE REGRESSION **](#)

Assess a basic ridge regression after PCA

[PCR***](#)

Assesses a basic model of PCR (components introduced in the order of eigenvalues)

[PCR1 **](#)

Assess a basic model of PCR (components introduced in the order of eigenvalues)

[QUICKPLS](#)

Quick PLS regression from 1 to ndim dimensions

[REGRESSION SCORE *](#)

build a factorial space for regression

[RIDGE REGRESSION ***](#)

Assesses a basic ridge regression

[SAISIRPLS ***](#)

PLS regression with dim dimensions

[SDIMCROSSPCR1 **](#)

Crossvalidation of stepwisePCR

[SIMPLE REGRESSION](#)

Mono_linear regressions

[SPCR **](#)

Stepwise Principal component regression

[STEPWISE REGRESSION ***](#)

Stepwise regression between x and y

5.5.4 Discriminations

[APPLY_QUADDIS](#)

Application of Quadratic discriminant analysis

[APPLYFDA1](#) **

Application of factorial discriminant analysis on PCA scores

[APPLYPLSDA](#) *

Apply pls discriminant analysis after model assessment using plsda

[BAG2GROUP](#)

use the identifiers in bags to create groups

[BMAHA](#) **

Assess a simple discriminant analysis

[CREATE_GROUP](#)**

Create a vector of number indicating groups from identifiers

[CREATE_GROUP1](#)***

Create a vector of number indicating groups from identifiers

[CROSSFDA1](#)**

crossvalidation on discrimination according to fda1 (directly on data)

[CROSSMAHA](#)**

Crossvalidation on discrimination according to maha1 (*directly on data*)

[CROSSMAHA1](#)**

Cross validation (random) of discriminant analysis *applied after PCA*

[CROSSPLSDA](#) **

Cross-validation on PLS discriminant analysis

[CROSSVAL_QUADDIS](#)

crossvalidation of quadratic dis. analysis

[FDA1](#) ***

Stepwise factorial discriminant analysis on PCA scores

[FDA2](#) ***

Stepwise factorial discriminant analysis on PCA scores with verification

[MAHA](#)**

Simple discriminant analysis forward introducing variables no validation samples

[MAHA1](#)**

Forward discriminant analysis DIRECTLY ON DATA with validation samples

[MAHA3**](#)

Simple discriminant analysis forward introducing variables no validation samples

[MAHA4**](#)

Simple discriminant analysis forward introducing variables no validation samples

[MAHA5**](#)

Simple discriminant analysis forward introducing variables with validation samples

[MAHA6**](#)

Simple discriminant analysis forward introducing variables with validation samples

[PLSDA ***](#)

Assesses pls discriminant analysis following the saisir format

[QUADDIS](#)

Quadratic discriminant analysis

5.5.5 Méthodes factorielles diverses

[accpsNew](#)

Finding common dimensions in multitable data (saisir format)

[APPLY NUÉE *](#)

apply Nuee dynamique (KCmeans)

[CA ***](#)

Analyse des correspondances

[COMDIM **](#)

Finding common dimensions in multitable data (saisir format)

[D2 FACTORIAL MAP ***](#)

Assesses a factorial map from a table of *squared* distances

[MFA**](#)

Multiple factor analysis

[MULTIWAY PCA **](#)

Multi way principal component analysis

[NUÉE **](#)

Nuee dynamique (Kcmeans)

[PCA CANO **](#)

Generalized canonical analysis after PCAs on each table

STATIS **

Factorial processing of multitable data using the STATIS method

5.6 Divers

CHANGE_NAME

- change the name of the observations of file X

CHECK_NAME*

Controls if some strings are strictly identical in a string array

SAISIR_CHECK

Check if the data respect the saisir structure

DENDRO **

Dendrogram using euclidian metric and Ward linkage

DENDRO1 **

Dendrogram using euclidian metric and Ward linkage

DURBIN_WATSON *

Assess the Durbin Watson value on the column of a table1
function [saisir] = durbin_watson(X)

FIND_INDEX

- find the index corresponding to the closest value in variable names

FIND_PEAKS**

Represents a curve showing its peaks

FORMAT1 **

-format input arguments for display

LIST**

Lists rows (only with a small number of columns)

MIR_STYLE *

Changes the sign of the variables of MIR spectra

MSC*

Applies a multiplicative scatter correction on spectra

NUM2STR1 *

Justified num2str

NUEE **

Nuee dynamique (Kcmeans)

REPEAT STRING

build a matrix of char by repeating a string

SAISIR LINKAGE *

Assesses a simple linkage vector from a matrix of distance

SEEKSTRING ***

Returns a vector giving the indices of string in matrix of char x in which 'str' is present

6 Recommandations au développeur

(Ces règles ne sont que des suggestions)

Il peut être utile de modifier ou de développer des fonctions spécifiques utilisables dans l'environnement SAISIR.

Voici quelques règles qui devraient faciliter la portabilité des fonctions, et permettre d'enrichir progressivement l'environnement disponible.

L'environnement SAISIR ne comprend pas de script mais seulement des fonctions, de la forme `function [result, result1]=name(param1,param2, ...)`

Nom des fonctions

Les fonctions doivent être écrites en anglais-américain, aussi bien en ce qui concerne le nom de la fonction que le code Matlab et les remarques.

Le nom d'une nouvelle fonction doit être forgé, si possible, selon la logique de MATLAB, avec un emploi systématique des mêmes lexèmes lorsqu'ils existent (par exemple : `str` pour *string*, `num` pour *number*, `nan` pour *not a number* etc.).

Ordre des paramètres

Les paramètres d'entrée sont donnés dans l'ordre de leur importance, avec en premier les matrices sur lesquelles porte la fonction, suivis des paramètres de la fonction. Les matrices d'entrée, doivent (sauf cas exceptionnel) être elles-mêmes formées selon la logique de SAISIR, avec les trois champs `.d`, `.i`, `.v` .

Les paramètres optionnels sont donnés en dernier.

Remarques d'entête

Les premières lignes de la fonction doivent être des remarques qui peuvent être consultées par la fonction `help`.

La première ligne de remarque (qui est lue par la fonction `help <directory>`) doit impérativement redonner le nom de la fonction, suivi d'espaces, suivi d'une brève définition tenant sur une ligne. Par exemple, la première ligne de remarque de la fonction `pca` est :

```
%pca          - Principal component analysis on raw data
```

Cette ligne est impérativement suivie de la recopie de l'entête de la fonction. Par exemple :

```
%function [pcatype]=pca(x, info)
```

Si la fonction comprend des paramètres optionnels, ils sont indiqués entre parenthèses. Par exemple :

```
%function [x1] = norm_col(x,(mode))
```

```
%mode (optional): 0 or 1 division by n or by n-1 respectively
```

```
%default : 0
```

Si il y a des paramètres optionnels, leur valeur par défaut doit être donnée dans les remarques d'entête.

Le reste des remarques d'entête donne la signification des arguments d'entrée, puis celles des arguments de sortie.

A partir de la version 7 de Matlab, le nom de la fonction doit être identique au nom du fichier « .m » de la fonction en respectant les lettres majuscules et minuscules. Ainsi, une fonction appelée **demo**, sera bien reconnue si le fichier s'appelle **DeMo.m** par exemple, mais conduira à l'affichage d'un warning Matlab « *inexact Matching* ». Pour cette raison, nous conseillons d'écrire tous les noms de fonctions en minuscules, avec correspondance exacte du nom du fichier.

De plus la première ligne, qui est donnée par la commande Matlab `help <directory>` doit être écrite de la manière suivante : par exemple

```
%demo[espace][espace]-[espace]Fonction de démonstration
```

L'enchaînement nom de fonction, espace(s), tiret , espace , texte permet que la fonction soit reconnue par Matlab dans le `help`, qui est alors interactif et prend la forme d'un « lien hypertexte ».

Arguments de sortie

Lorsque les arguments de sortie sont des vecteurs ou des matrices, ils doivent impérativement respecter la logique de SAISIR avec les trois champs. Par exemple, une fonction `matsum` qui fait la somme de deux matrices doit être écrite de la manière suivante :

```
function [x3] = matsum(x1,x2)
%MATSUM          - sum of two matrices
%function [x3] = matsum(x1,x2)
% Returns the matrix sum x1 + x2
```

```
[n1,p1]=size(x1.d);
[n2,p2]=size(x2.d);
```

```
if((n1~=n2)|(p1~=p2))
    error('Irrelevant dimensions')
end;
```

```
x3.d=x1.d+x2.d;
x3.i=x1.i;
x3.v=x1.v;
```

La fonction vérifie que les dimensions sont compatibles, mais, pour des raisons de rapidité, ne vérifie pas la cohérence complète des identificateurs des individus ou des variables.

Lorsque une fonction retourne de nombreuses informations, il est souhaitable de regrouper les arguments de sortie dans un petit nombre de structures (en général, une seule !) dont chaque champ contient une information au format de SAISIR.

Par exemple, la fonction `PCA` (analyse en composantes principales) n'a qu'un argument de sortie qui est une structure. On aurait par exemple :

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

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

Chaque champ est une structure SAISIR. Lorsque les matrices de sortie sont créées par la fonction (comme les coordonnées factorielles de l'ACP, dans le champ `score`) le développeur doit se soucier de construire lui-même les identificateurs de variables et d'individus. On a ainsi :

```
>>p.score.v
```

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

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

Dans l'exemple, `p.score.v` est une matrice de caractère, dont la deuxième ligne est, par exemple :

```
« A2 16.9% » (11 caractères).
```

Organisation des répertoires

Il est préférable de ne pas modifier une fonction de l'environnement SAISIR en lui gardant son nom d'origine. Les nouvelles fonctions créées par le développeur doivent être sauvegardées tout d'abord (pour test) dans un répertoire, qui fait partie du chemin d'accès (*path*) de MATLAB. Lorsque ces fonctions sont validées, elles peuvent être placées dans le répertoire SAISIR, si elles ont un intérêt général.

Transmission des fonctions développées et signalement des bogues

Les fonctions d'intérêt général, respectant le format SAISIR et développées par les utilisateurs peuvent être envoyées par e-mail à bertrand@nantes.inra.fr.

Il est alors utile que la fonction comprenne une remarque indiquant les coordonnées de l'auteur. L'utilisateur faisant cette démarche (bénéfique) admet implicitement que la fonction peut alors être diffusée librement, sans redevance ou droit de propriété. Dans les articles et les communications, les utilisateurs sont tenus de citer les auteurs des fonctions originales.

Quelques lexèmes des fonctions SAISIR

(A utiliser dans les noms et les arguments des fonctions)

2 : 'to' indique une transformation (par exemple num2str)

apply : appliquer (un modèle construit ailleurs)

append : concaténer

bag : structure de type saisir dans laquelle la partie .d est formée de matrice de chaîne de caractère

col : numéro de colonne ou collection de tableaux

delete : retirer

dim : dimension

group : groupe qualitatif (en général : vecteur SAISIR de nombre entiers)

load : charger

map : carte

range : vecteur (Matlab) de valeurs qui désignent un choix

read : lire

row : ligne d'un tableau

startpos endpos : désigne le début et la fin d'une zone à sélectionner

saisir : nom d'un fichier saisir

sel : vecteur de sélection

x : fichier saisir

Annexe A : module image

Cette annexe donne la liste des fonctions spécifiquement dédiées aux images.

Attention ! Ces fonctions utilisent le module commercial MATLAB **IMAGES**.

Pour des raisons de rapidité d'exécution, contrairement aux autres fonctions de l'environnement SAISIR, certaines des fonctions ne donnent pas un résultat au format SAISIR (.i, .v, .d). Seules les fonctions de haut niveau utilisent ce format.

Cette annexe contient également des fonctions spécifiques pour traiter des images provenant du système d'acquisition multispectrale MUWI (*MU*lti*W*ay *I*mager), développé à l'ENITIAA-INRA de Nantes. Le nom de ces fonctions commence par les lettres MUWI

Voir aussi la liste thématique ci-dessous.

BROWSE IMAGE

browse a series of image

```
function browse_image(image_list)
Display the images whose filename is in 'image_list'
image_list must contain a valid list of file names
"return" to continue
```

Exemple d'utilisation : examine toutes les images bitmap (extension '.bmp') :

```
files=dir('*.bmp')
list=char(files.name);
browse_image(list)
```

BROWSE PLAN

-Browse a series of plans

```
function browse_plan(cube_3D)
Display the channels of a cube image
Right button to go down, Left button to go up, Ctrl C to exit
```

BROWSE GRAY

-Browse a series of plan in gray values

```
function browse_plan(cube_3D,(reverse),(channelnames))
Display the channels of a cube image
Right button to go down, Left button to go up, Ctrl C to exit
```

CO OCCURRENCE FEATURES

Assess some texture features by the method of co-occurrences

```
function [feature]=co_occurrence_features(im)
im is a matrix of rgb values (from 0 to 255) as obtained from im=extract_imshape (first
output argument)
feature contains the parameter of texture characterisation as obtained by
the co-occurrence method
10 parameters are evaluated for each channel:1 Max probability;
2 row average ; 3 column average; 4 row std; 5 column std; 6 second angular moment
7 contrast; 8 correlation; 9 entropy ; 10 inverse differential moment
```

see Chtioui's thesis page 98

Exemple d'utilisation

```
a=imread('luz1.bmp');  
[prop, labelled_im]=extract_shape(a);%extract some properties and label the image  
sel=5;  
[im1, pixel_list,im]=extract_imshape(a, prop, sel);% extract a given shape (here : sel)  
feature=co_occurrence_features(im1);% result associated with the 5th shape  
feature est ici un vecteur de 10 x 3 valeurs numériques associées successivement à chacun  
des canaux rvb.
```

CONCAT

build a single feature matrix from a set of images

```
function res=concat();
```

Images with extension ".bmp" must be in a single directory

Creates the matrix of features of all the shapes isolated from the images.

Highest level function for extracting shapes

Fonction encapsulant toutes les fonctions d'extraction de paramètres d'objets. Tous les objets isolés dans chacune des images avec extension « .bmp » présents dans le répertoire forment les lignes de la matrice de résultat. Les colonnes sont les variables extraites des images selon la logique définie dans la thèse de Chtioui.

CUBE2SAISIR.M

Transforms a cube_image (from CRAGX Gembloux) into a saisir structure

```
function x=cube2saisir(cube_image,(code))
```

Unfold the cube_image into a matrix, giving the identifier 'code' to every pixel.

returns a saisir matrix having n*p rows and q columns,

where n, p q are the number of image row, image column and channels.

x.row contains the initial row position of the pixel in the cube image

x.column contains the initial column position

EXTRACT_IMSHAPE

Extracts the image of a single shape as isolated by extract_shape

```
function [im1,pixel_list,im]=extract_imshape(rgb, prop, number)
```

rgb is the original image

prop is the property structure obtained from extract_shape (applied on rgb)

number is the rank number of the shape to be extracted

results:

pixel_list is a MATRIX dimensioned npixel X 5, in which npixel is the number of pixels in the shape

pixel_list(:,1) and pixel_list(:,2) contain the row and col index of each pixel position

pixel_list(:,3:5) contains the RGB values of the pixel

im1 is the isolated rectangular image containing the single object.

im (useful only for representation) is the same image converted into an intensity image using `mat2gray`.

we have the relationship $im1 = im * 255$;

EXTRACT SHAPE

Extracts shapes and their features from an RGB image

function `[prop, labelled_im, nshape]=extract_shape(im,(channel))`

given an RGB image, find a threshold, and gives the properties of the shapes

return `prop` (properties of shapes), `labelled_im` : labelled image

`channel` (from 1 to 3) : choses the channel for thresholding the image. Default: all

`nshape`: number of detected shapes

GRAY LEVEL FEATURES

Assess some features of the gray level histograms

function `[feature]=gray_level_features(pixellist)`

`pixellist` is a list of pixels as obtained from `extract_imshape`:

the 2 first columns are the x y pos of the pixel, and the column 3:5 are

the luminances of the RGB channels.

Chtioui 's thesis page 96

result: `feature` contains 6 parameters for each channel: mean, variance, entropy,

energy, dissymetry, flattening, corresponding to each channel (R,G B) i.e 18 values

Exemple :

```
a=imread('luz1.bmp');
```

```
[prop, labelled_im]=extract_shape(a);%extract some properties and label the image
```

```
sel=5;
```

```
[im1, pixel_list,im]=extract_imshape(a, prop, sel);% extract a given shape (here : sel)
```

```
[feature]=gray_level_features(pixel_list);
```

HAAR

- compute haar transform

```
function H=haar(x);
```

See function `thaar` for the inverse Haar transform

source :J. Lifermann. Les méthodes rapides de transformation du signal,

Walsh, Hadamard, Haar. Masson Publisher, 1980.

Compute the Haar transform of each row of x.

if the number of columns in x is not a power of 2, extracolumns are added

in order to reach the closest power of 2.

H.v contains the name of the haar function such as "number1*number2"

number1 indicates the window size of the considered fonction (large number=small window).

number2 the position of this window in x.

Example : image 256 x 256 (after zero filling)

$256=2^8$

variable 7*4 (number1=7; number2=4

Window size = $256/(2^7) = 2$ consecutive data points

position= $2*4=8$: data points 7 and 8

HAAR2

- two-dimensional haar transform

```
function h = haar2(x);
```

x can be either a one-plan image or a saisir file

x is zero-filled in order to reach a power of 2 such as 2, 4, 8 ... on rows and col

HAAR MULTISCALE

- show the plan through haar filtering

```
function im=haar_multiscale(h,xpower);
```

h is the output argument of fonction haar2

Cette fonction permet de donner une représentation à différents degrés de détail d'une image.

Elle effectue la transformation de Haar inverse en ne gardant que les fonctions définies par xpower

HAAR TEXTURE

- Compute image texture feature from an image

```
function res=haar_texture(filenamees,xpower,nreplicate);
```

accept only RGB or gray images

Warning : zero filling if the number of rows/columns are not power of 2 !

This not recommanded to have number of rows/columns not power of 2 !

Computes all the texture characteristics until xpower.

For example, with power =6 computes textures associated with haar functions

61 62 63 64 65 65 51 52 53 54 55 56 41 42 43 44 45 46 31 32 33 34 35 36 21
22 23 24 25 26 11 12 13 14 15 16

nreplicate indicates the number of desired duplication of masks.

In many situation, it is not useful to have very large mask, which are indeed not repeated in the image !!

HISTOGRAM DIFFERENCE FEATURES

Assesses some texture features by the method of histogram difference

```
function [feature,status]=difference_feature(im)
```

im is a matrix of rgb values (from 0 to 255) as obtained from im=extract_imshape (first output argument)

feature contains the parameter of texture characterisation as obtained by the grey level histogram difference method

status indicates if the assesement was possible :1 or impossible:0

if the assesement impossible: feature is a vector of 0

see Chtioui's thesis page 98.

the paremeters are mean; variance; entropy; energy; contrast for each chanel ie 15 values

HISTOGRAM DIFFERENCE FEATURES1

- Texture features by the method of histogram difference

function [feature]=histogram_difference_feature(im,channel_range,displacement)

feature contains the parameter of texture characterisation as obtained by

the grey level histogram difference method

if the assessment impossible: 'feature' is a vector of 0

see Chtioui's thesis page 98.

% extracted features: mean, std, kurtosis

% ADAPTED FOR GRAY LEVEL = FLOATING VALUES

Input args:

im : a (possibly multivariate) image

channel_range: vector of integer values giving the plan in im to be processed

displacement : vector with two integer values: delta in x, delta in y

Output args

feature: matrix of features of the image

with mean, standard deviation and kurtosis.

This group of 3 features are given for each element in "channel_range" .

Cette fonction ne s'applique qu'à une seule image (éventuellement multivariée), et peut traiter tous les plans dont les numéros sont donnés par channel_range.

Le nombre de variables de sortie est égal au nombre d'éléments dans channel_range x 3 (moyenne, écart_type et kurtosis de l'histogramme des différences). La moyenne est calculée sur l'histogramme des valeurs absolues des différences.

IMAGE HISTOGRAM

Build the gray-level histogram on an RVB image

function histo=image_histogram(tab,point, delta_x, delta_y)

On the 3D table tab (supposed to be and RVB image in 256 grey_level values),

assess the histogram in the rectangle defined

by point (top left). The delta bottom right values are in delta x and delta y

only process the first pair of point (if there are many)

cumulates the histogram in the order 1 2 3 (normally RGB)

warning! the grey levels values are ranging from 0 to 255, but the Matlab indices are ranging from 1 to 256 for each histogram!

Retourne le vecteur des fréquences absolues des 3 histogrammes R, V, B à la suite

IMAGE OBJECT FEATURES

Extracts size, color and texture features from an RGB image

function [feature]=image_object_features(rgb, (channel),(which),(code))

The rgb image must be formed of coloured objects placed on a rather white background.

The objects in the image are first isolated, then the features of each object are estimated

Features are eventually gathered in the matrix "feature" (in the Saisir format).

channel (optional) indicates the RGB channel on which the thresholding is

performed. Default (for channel <0 or channel >3) : all

'which' contains the number of the labelled isolated object to be studied (default: all the objects).

'code' contains an identifier which is used to prefix the identifier of the

object in the 'feature' matrix.

The choice of objects (as achieved by 'which' argument) to be studied can be defined from the examination of the labelled image:

The numbers of the labelled objects can be found by using the two functions

```
[prop, labelled_im]=extract_shape(rgb);
```

```
show_labelled(rgb, prop);
```

If an object is not valid (for example, if it is a straight line)

some features are not assessed and replaced with 0

IMAGE OCCURRENCE

Assess texture features using cooccurrence method

```
function r=image_occurrence(tab,point, delta_x, delta_y,dir_x,dir_y)
```

On the 3D table tab (supposed to be an RGB image in 256 grey_level values),

assess 6 x 3 co-occurrence features according to

Gonzales and Wintz, "image processing", Addison-Wesley, page 416 on each channel.

The area is defined by point (top left). The delta bottom right values are in delta_x

and delta_y. Only process the first pair of point (if there are many)

cumulates the features in order 1 2 3 (normally RGB)

warning ! dir_x and dir_y define the translation operator.

Do not confound with delta_x and delta_y which simply define the region of interest.

the rectangle area is supposed to be surrounded by places where the pixels are still

interesting : the starting or ending pixels may lay out of the rectangle

current version : no security

Travaille sur l'image entière et non sur une forme isolée.

INVARIANT MOMENT

Assesses the first 7 invariant moments from a binary image

```
function [feature]=invariant_moment(pixellist)
```

pixellist contains the list of the xy positions of the object and can be obtained as a result of

```
prop= regionprops(labelled_im,'All') or prop=extract_shape(im)
```

```
prop(sel).PixelList contains the pixel list of the (sel)ected binary image
```

as labelled by regionprops.

the invariant moments are assessed according to Chtioui thesis, page 92 and following.

LOAD BINARY MATRIX

loads a matrix saved in binary format

```
function res=load_binary_matrix(name,nrow,ncol,precision)
```

example:

```
res=load_binary_matrix('seg.ima',720,576,'int8')
```

see the "help" of fread for the possible values of "precision"

The binary matrix must be a succession of numbers in binary format.

Only limited use.

MANY IMAGE HISTOGRAMS

-build histograms of many images and many areas

function saisir=many_image_histograms(list,point, delta_x, delta_y)

load RGB images from 'list' and assess the histogram in all the areas defined by point, delta_x, delta_y

cumulates the histogram in the order 1 2 3 (normally RGB)

numbers of areas (as given in 'point') are in the identifiers of rows as first code

Cette fonction est utile pour établir les histogrammes de niveaux de gris dans plusieurs rectangles positionnés sur plusieurs images RVB.

MANY IMAGE OCCURRENCES

Calculates occurrence features on areas of several images

function saisir=many_image_occurrences(list,point, delta_x, delta_y, dir_x, dir_y)

load RGB images from 'list' and assesses the occurrences in all the areas defined by point, delta_x, delta_y

cumulates the occurrences in the order 1 2 3 (normally RGB)

numbers of areas (as given in 'point') are in the identifiers of rows as first code

Cette fonction est utile pour établir les caractéristiques de texture dans plusieurs rectangles positionnés sur plusieurs images RVB.

MUWI browse

Display the images at the MUWI format

function MUWI_browse(dos_ext);

Les images MUWI (Multiway imager) proviennent du système d'acquisition développé par ENITIAA-INRA.

La fonction montre successivement (*feuillette*, ou *browse* en anglais) toutes les images du répertoire courant qui sont définies par la chaîne de caractère DOS `dos_ext`. Les images ainsi sélectionnées doivent être des images MUWI, dont l'extension est normalement '.don'. Par exemple, l'ordre `MUWI_browse('II*.don')` déclenchera l'affichage de toutes les images contenues dans des fichiers dont le nom commence par 'II' et portant l'extension '.don'. On appuie sur la touche retour chariot pour passer à l'image suivante, ou `ctrlC` pour arrêter l'affichage. On peut naturellement n'afficher qu'une image par la commande

`MUWI_browse('nom')` où `nom` est une chaîne de caractère donnant le nom d'un fichier image.

La fonction affiche 4 images. Les trois premières, désignées par **channel #1, 2 ou 3** représente les plans-image (canaux) rouge, vert, bleu. Le canal vert (normalement dupliqué par MUWI) n'est pas représenté. Ces images sont représentées en fausses couleurs. Toute l'échelle des fausses couleurs est utilisée. C'est à dire que la valeur minimale des pixels du plan considéré est codée en bleu profond, et la valeur maximal en rouge vif. La quatrième image, dont le titre est le nom du fichier-image correspondant, est une représentation en couleurs par la combinaison des canaux RVB de MUWI.

La fonction utilisée sans argument (soit `MUWI_browse()`) feuillette toutes les images portant l'extension '*.don' dans le répertoire courant.

MUWI protocol

Build a protocol for MUWI project.

```
function str=MUWI_protocol(protocol_name,protocol_structure)
```

Protocol_structure must be an array of structure, each of them having the following field (example)

name: 'dos1.don' %% name of the resulting image

TIC: 0.020 %% integration time (Milli seconds)

gain: 1 %% Gain

TSC: 200 %% TimeStarting camera

LED: [1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] %% Illuminated diodes

TSL: 0 %% time of starting led

TEL: 1000 %% Time of Illumination LED (Milli seconds)

FEL: 0 %% Frequency of illumination (0: permanent)

Pour pouvoir utiliser MUWI de manière à acquérir une série d'image, il faut utiliser, au niveau système DOS, une commande du type `project2 c:\manip\proto`

Cette commande lance l'acquisition d'une suite d'images MUWI selon une liste de commande contenue dans le fichier texte `proto`.

On peut en principe créer ce fichier texte « manuellement » à partir d'un éditeur de texte comme NotePad ou WordPad. Cette tâche est cependant fastidieuse, et la fonction `MUWI_protocol` tente de la rendre plus aisée.

Dans cette logique, une acquisition d'image est définie par une structure Matlab contenant les champs suivants (supposons que la structure s'appelle `test`).

`test.name` : chaîne de caractère contenant le nom du fichier image résultant, portant l'extension '.don'. Par exemple ('`cette_image.don`')

`test.TIC` : Nombre indiquant le temps d'intégration de la caméra, en millisecondes

`test.gain` : Nombre indiquant le gain de la caméra (valeur conseillée : 1)

`test.TSC` : Nombre indiquant le temps de démarrage de la caméra, en millisecondes (time starting camera)

`test.TSL` : Nombre indiquant le moment d'allumage des leds.

`test.LED` : vecteur formé de 32 éléments (0 ou 1) donnant la nature des LED allumé
1 indique une LED allumée.

Les LED sont codées dans l'ordre suivant :

32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Blanc	UV 400	Bleu	Vert	Ambre	Rouge	IR 875	IR 950	Blanc	UV 400	Bleu	Vert	Ambre	Rouge	IR 875	IR 950	Blanc	UV 400	Bleu	Vert	Ambre	Rouge	IR 875	IR 950	Blanc	UV 400	Bleu	Vert	Ambre	Rouge	IR 875	IR 950
Ouest								Sud								Est								Nord							

Les lignes suivantes indiquent comment former un vecteur d'éclairage de tous les cotés :

```
led=zeros(8,32);
for i=1:8;
    led(i,aux)=1;
    aux=aux+1;
end
```

La matrice `led` contient les positions d'allumage des led dans l'ordre :

```
{'IRA','IRB','RED','AMB','GRE','BLU','UV$','WHI'});
```

soit IR950, IR875, rouge,ambre, vert, bleu, UV et blanc.

Ainsi, dans la structure de l'exemple `test`,

la ligne `test.LED=led(3, :)` déclanchera l'allumage de la led rouge.

`test.TEL` : Nombre indiquant le temps d'illumination des diodes en millisecondes

`test.FEL` : Nombre indiquant la fréquence de clignotement des LED en Hertz (de 0 à 100 000. La valeur 0 (conseillée) indique que les LED ne clignote pas

Pour chaque image à acquérir en séquence, on crée une structure de ce genre (par exemple `test`, `test1`, `test2...`)

On forme un vecteur de structure par un order du type
`sequence(1)=test ; sequence(2)=test1 ; sequence(3)=test2 ...`

La commande (par exemple)

```
mon_protocole=MUWI_protocol('mon_fichier', sequence) ;
```

crée et sauvegarde sur le disque dur un fichier texte appelé '`mon_fichier`' qui contient les commandes d'acquisition des images, telles qu'elles sont définies dans `sequence`.

`mon_protocole` contient les mêmes ordres dans l'environnement Matlab, sous la forme de chaînes de caractères.

Ainsi, après ce travail, au niveau système dos, la commande (par exemple)

```
project2 C:\manip\mon_fichier
```

déclenche la série d'acquisition d'images précédemment définie.

`project2` demande le chemin complet explicite de `mon_fichier`.

`project2` est le nom du programme, écrit en C++ qui commande le système MUWI.

Toutes les images résultantes, au format MUWI, sont sauvegardées dans le répertoire courant.

MUWI read

read image under the MUWI Format

```
function [res,header]=read_MUWI(name)
```

Cette fonction charge une image MUWI, de manière à former un cube-image. L'image est récupérée dans un tableau de `double` à trois indices, par exemple `res(i, j, k)`. Le premier indice correspond au numéro de ligne de l'image, le deuxième au numéro de colonne, le troisième aux 4 canaux des images MUWI. Les luminances (niveau de gris) sont codés par des nombres en double précision variant de 0 (pas de signal) à 1 : saturation.

Les 4 canaux (plans-image) de MUWI sont : 1 rouge, 2 : vert ; 3 : bleu ; 4 : deuxième canal vert. Au niveau du binaire, le canal vert est en fait dupliqué de manière à permettre un stockage rapide de blocs codés binaires. En principe, un canal vert unique peut être reconstitué en faisant la moyenne des canaux 2 et 4.

L'argument de sortie optionnel header (entête) donne l'entête contenu dans les images codées binaires. Il n'y a pas d'usage direct de cette entête, sauf pour vérifier le bon fonctionnement de MUWI_read.

MUWI sampling

Batch sampling pixels in MUWI images

fonction[res,header,coord]=MUWI_sampling(dos_ext,top_left,bottom_right,grid_point)

Le program executable au niveau système project2 permet d'acquérir des séries d'images MUWI du même objet. Par exemple la commande

project2 c:\manip\mon_protocole tente d'exécuter le protocole MUWI mon_protocole et de créer les images MUWI correspondantes dans le répertoire manip.

Pour pouvoir effectuer des études statistiques, il est nécessaire de pouvoir échantillonner (sampling) des pixels appartenant à chacune des images MUWI obtenues.

La fonction MUWI_sampling exécute ce travail. La chaîne de caractère dos_ext indique le critère de choix DOS (filtre DOS), conduisant à la sélection de fichiers image MUWI (extension .don).

Les paramètres top_left (en haut à gauche), bottom_right (en bas à droite) désignent les coordonnées du rectangle sous la forme de vecteurs à deux dimensions.

Le paramètre grid_point indique le nombre de points de quadrillage (grid) que l'on souhaite avoir sur chaque coté du rectangle. Par exemple si grid_point vaut 10, on échantillonne 10x10 = 100 pixels.

La fonction renvoie dans le premier argument de sortie une matrice SAISIR de dimensions nimage x (ngrid x ngrid x 4), où nimage indique le nombre d'images sélectionné par le filtre dos_ext. Les colonnes correspondent aux pixels à chacune des coordonnées d'échantillonnage : tous les canaux « rouge » d'abord (soit ngrid x ngrid valeurs), suivi de tous les canaux « vert », puis « bleu », puis « deuxième vert ».

Les noms des lignes (res.i) sont les noms des fichiers images.

Une valeur de res.d indique la luminance de l'image pour le pixel considéré.

res.v contient une indication de canal et des coordonnées pixels.

MUWI show

Display a MUWI image

fonction MUWI_show(name)

Affiche à l'écran une image MUWI vue comme une image RGB. Les deux canaux verts (2 et 4) sont mélangés pour l'affichage.

La fonction admet comme argument soit un cube image MUWI (obtenu par MUWI_read), soit une chaîne de caractère donnant le nom d'un fichier-image sur le disque dur, dans le répertoire courant.

ONE IMAGE HISTOGRAMS

Build the histograms of rectangle areas of one image

function `saisir=one_image_histograms(name,point, delta_x, delta_y)`

load a RGB image and assess the histogram in all the areas defined by

`point, delta_x, delta_y`

cumulates the histogram in the order 1 2 3 (normally RGB)

numbers of areas (as given in `point`) are in the identifiers of rows as first code

ONE IMAGE OCCURRENCES

Calculates texture features on several areas of a single image

function `saisir=one_image_occurrences(name,point, delta_x, delta_y, dir_x, dir_y)`

load a RGB image and assess the occurrence texture values in all the areas defined by

`point, delta_x, delta_y`

cumulates the occurrences in the order 1 2 3 (normally RGB)

numbers of areas (as given in `point`) are in the identifiers of rows as first code

PERIM FFT

Calculates the fourier coeff. of the boundary of a binary form

function `[feature,status]=perim_fft(bwim)`

return a vector of the 10 first moduli of the fourier coefficient of the boundary (in `feature(1:10)`)

and the number of pixels composing the perimeter in `feature(11)`;

`status` indicates if the assessment is 1: feasible; 0: impossible

If impossible `feature` is a vector of 11 zeros

READ FSM

reads fsm files

function `[cube info]=read_fsm(filename);`

Le format fsm est utilisé par des fabricants d'instruments scientifiques pour manipuler des images multivariées.

`info` donne des infirmations sur la nature de l'image, notamment les plages de longueurs d'onde ou de nombre d'onde.

RUN LENGTH FEATURES

Calculates some texture features by the run length method

function [feature]=run_length_features(im)

im is a matrix of rgb values (from 0 to 255) as obtained from im=extract_imshape (first output argument)

feature contains the parameter of texture characterisation as obtained by the grey level run_length method

feature is a 15 elements vectors giving

short_run_emphasis, long_run_emphasis, grey_level_distribution, run_length_distribution, proportion_of_runs

for successively the R, G, B channels.

see *Chtioui's thesis page 99*.

or *Galloway M.M "texture analysis using gray level run lengths". Computer graphics and image processing, 4, pp 172-179, 1975.*

For the sake of rapidity, only the case direction = 45 degree treated here

If status = 1: the assesment is possible; 0 otherwise.

If status = 0: the program return a vector of 0 in "feature"

SAISIR2CUBE

Transforms a saisir file into a cube_image

Function cube=saisir2cube(x,(nrow),(ncolumn),(nplan))

Refolds a matrix in order to form a cube image

returns a cube having nrow rows ncolumn columns and nplan plans

Example of utilization :

data=saisir2cube(pca.score,240,320,81);

default values (Gembloux format) nrow=240, ncolumn=320, nplan=81

Utile pour « replier » des facteurs après une analyse multidimensionnelle.

SELECT PIXEL

Select pixel vectors in an image

function[res]=select_pixel(image,(noplan),n,xcode))

returns the values of the pixels at positions

selected by clicking the mouse

code is an optional value added to the identifiers of rows

n is an optional number of pixels (exit if n pixels where selected)

for exiting press <Carriage return>

SHOW LABELLED

Displays and rgb image with numbers associated with shapes

function `show_labelled(rgb, prop, charsize)`

`rgb` is the original image

`prop` is the properties obtained from `extract_shape`

SHOW PLAN

Displays a plan of a cube image

function `h=show_plan(cube_image,noplan)`

with :

`cube` the cube image

`noplan` the number of the plan (channel) to be displayed

THAAR

- compute the inverse haar transform

function `X=thaar(H)`;

The number of columns of `H` must be a power of 2 (2, 4, ... 256 ...)

See function `haar` for the direct Haar transform

source :J. Lifermann. Les méthodes rapides de transformation du signal, Walsh, Hadamard, Haar. Masson Publisher, 1980.

Annexe A (suite):

LISTE THEMATIQUE

1 Affichage d'images

[BROWSE_IMAGE](#)

browse a series of image

[BROWSE_PLAN](#)

-Browse a series of plans

[SHOW_LABELLED](#)

Displays and rgb image with numbers associated with isolated shapes

[SHOW_PLAN](#)

Displays a plan of a cube image

2 Manipulation de « cube images »

[CUBE2SAISIR.M](#)

Transforms a cube_image (from CRAGX Gembloux) into a saisir structure

[SAISIR2CUBE](#)

Transforms a saisir file into a cube_image

3 Binarisation et extraction de formes connexes

[EXTRACT_IMSHAPE](#)

Extracts the image of a single shape as isolated be extract_shape

[EXTRACT_SHAPE](#)

Extracts shapes and their features from an RGB image

[PERIM_FFT](#)

Calculates the fourier coeff. of the boundary of a binary form

4 Analyse de la texture d'image

[CO_OCCURRENCE_FEATURES](#)

Assess some texture features by the method of co-occurrences

[IMAGE_OCCURRENCE](#)

Assess texture features using cooccurrence method

[MANY_IMAGE_OCCURRENCES](#)

Calculates occurrence features on areas of several images

[ONE_IMAGE_OCCURRENCES](#)

Calculates texture features on several areas of a single image

[HAAR_TEXTURE](#)

- Compute image texture feature from an image

[HISTOGRAM_DIFFERENCE_FEATURES](#)

Assesses some texture features by the method of histogram difference

[HISTOGRAM_DIFFERENCE_FEATURES1](#)

- Texture features by the method of histogram difference

[RUN_LENGTH_FEATURES](#)

Calculates some texture features by the run length method

5 Manipulation d'histogrammes des niveaux de gris

[IMAGE HISTOGRAM](#)

Build the gray-level histogram on an RVB image

[MANY IMAGE HISTOGRAMS](#)

-build histograms of many images and many areas

[ONE IMAGE HISTOGRAMS](#)

Build the histograms of rectangle areas of one image

[GRAY LEVEL FEATURES](#)

Assess some features of the gray level histograms

6 Extraction de caractéristiques d'objets isolés

[IMAGE OBJECT FEATURES](#)

Extracts size, color and texture features from an RGB image

[CONCAT](#)

build a single feature matrix from a set of images

[INVARIANT MOMENT](#)

Assesses the first 7 invariant moments from a binary image

[PERIM FFT](#)

Calculates the fourier coeff. of the boundary of a binary form

7 Commandes de l'environnement MUWI

MUWI est l'acronyme de *Multiway imager*, système de vision multivoie développé à l'ENITIAA-INRA de Nantes.

[MUWI browse](#)

Display the images at the MUWI format

[MUWI protocol](#)

Build a protocol for MUWI project.

[MUWI read](#)

read image under the MUWI Format

[MUWI sampling](#)

Batch sampling pixels in MUWI images

[MUWI show](#)

Display a MUWI image

8 Divers

[HAAR](#)

- compute haar transform

[HAAR2](#)

- two-dimensional haar transform

[HAAR MULTISCALE](#)

- show the plan through haar filtering

[SELECT PIXEL](#)

Select pixel vectors in an image

THAAR

- compute the inverse haar transform

Annexe B : Module olfactogramme

(état au 30 mars 2003)

Ce module donne l'ensemble des procédures qui ont été créées pour manipuler et exploiter des olfactogrammes (chromatogrammes d'odeur). Pour bien comprendre ce module, il est nécessaire de comprendre l'enchaînement logique de certaines procédures. Cette annexe donne un exemple de script MATLAB commenté, montrant la suite des procédures à mettre en oeuvre pour exploiter une collection d'olfactogrammes (voir annexe **B 2**). De plus, certaines procédures d'entrée (notamment `find_descriptor`) et `excel2aroma` dépendent très fortement de la structure des fichiers disponibles, et sont à réécrire dans chaque situation particulière. En cas de difficulté, contacter bertrand@nantes.inra.fr.

B 1 Liste alphabétique des commandes

[aroma_curve](#)

Represents an aromagram with the names of identifiers

`h=aroma_curve(chroma, alias, number)`

Warning! number correspond to the chromatogram. It is half of the real position of the intensity vector

function `h=aroma_curve(chroma, dico, number)`

Donne une représentation graphique des olfactogrammes.

`chroma` est le résultat de `excel2aroma`, tandis que `alias` est le résultat de `create_alias`.

[aroma2excel](#)

Saves an olfactogram file under an excel format.

function `aroma2excel(chroma,aliased, filename)`

The file to be saved must respect the format as described in function `excel2wine` or in `excel2aroma`

`aliased` is the dictionary of synonyms

`filename` is the .CSV resulting file (readable by Excel).

The resulting file is transposed in order to respect the Excel constraints: 255 columns and no limits on number of rows.

Fails if there are more than 122 chromatograms

[assigned2excel](#)

saves assigned peaks under Excel

function `assigned2excel(chroma,aliased, filename)`

Warning: gives only the identifiers (and not the intensity)

Fails if the number of retention times (columns) is larger than 122

See also `AROMA2EXCEL`

Proche de `aroma2excel`, mais les données ne sont pas transposées, et les intensités ne sont pas conservées.

Utile pour examine les attributions du point de vue du sens des descripteurs.

[browse_aroma](#)

`browse_aroma(chromato,alias)`

Représente les aromagrammes les uns après les autres.

Right button to go down, Left button to go up, Ctrl C to exit

[check_alias](#)

Permet de vérifier la liste des synonymes créée par `create_alias`

Affiche la liste des synonymes.

Par exemple:

```
check_alias(res)
```

```
l'alias de '?' ' est 'fruit rouge' '
```

```
ans =
```

```
166
```

```
l'alias de '?' (blanc d'œuf) ' est '?' '
```

```
ans =
```

```
1
```

```
l'alias de '?' (riz) ' est '?' '
```

```
ans =
```

```
1
```

[create_alias](#)

-interactively build a series of alias

```
function [dico1] = create_alias(dico,(alias),(indice))
```

from a given list of descriptors, in dico, interactively build a series of alias

in order to reduce the number of names

indice is the place of the file actually processed

Permet de créer une liste de synonymes, de manière à réduire le vocabulaire initial.

dico est le résultat de la fonction `find_descriptor`

La commande la plus simple est de la forme:

```
[dico1] = create_alias(dico)
```

```
>> alias or <RETURN> or 'prec' or 'fin'
```

La fonction présente l'un après l' autre chaque descripteur initial. L'utilisateur a le choix de le garder tel quel, de le modifier, de "remonter" au précédent (en tapant `prec`), ou de finir en tapant `fin`. On peut sortir même sans avoir fini la liste.

Dans ce cas, seuls les descripteurs déjà traités possèdent éventuellement un synonyme. Pour reprendre le travail, on utilise alors la commande (par exemple):

```
[dico2] = create_alias(dico,dico1,10)
```

qui signifie que l'on veut continuer la liste dans dico2, à partir de la liste de synonymes débutée dans dico1, et en commençant au 10^{ème} descripteur.

A la fin, on dispose d'une liste de synonymes.

Les synonymes peuvent être des descripteurs déjà présents dans le dictionnaire initial, ou nouveaux. Cela est signalé par le message "*found*" (trouvé) ou "*new*", nouveau descripteur. Le message donne également un nombre qui est le nombre d'occurrences du descripteur en cours. Cela peut aider l'utilisateur à créer ou non un nouveau descripteur: il paraît moins réducteur de donner un synonyme banal à un descripteur "rare" qu'à un descripteur très couramment utilisé.

Le fichier résultat donne les synonymes. Par exemple

list(res)

1 ?	1	
2 ? (blanc d'œuf)	343	
3 ? (désagréable)	1	
4 ? (riz)	293	
5 ? (végétal-amande amère)	335	
6 ? (végétale)	6	

Le descripteur '?' (inconnu) de numéro d'ordre 1 est sans synonyme: le nombre est le même que son numéro d'ordre.

Le descripteur '?' (*blanc d'œuf*) est remplacé par un autre descripteur qui est dans la liste en position 343. En affichant le contenu de la 343^{ème} ligne de res, par la commande `res.i(343,:)`, on s'aperçoit que l'utilisateur a remplacé ce descripteur par le nouveau descripteur '*œuf*'. Ainsi
>> `res.i(343,:)`

ans =

œuf

De même '?' (*riz*) est remplacé par `res.i(293,:)`, soit '*riz*' dans l'exemple.

[descri2](#)

Builds the table of frequency descriptors

function [res,occur] = descri2(chroma,alias,std)

Creates the table of frequency of descriptor in given areas of chromatograms

In order to take into account of the neighbouring, the peaks are replaced by a Gaussian curve, with a standard deviation equal to std (data point)

alias is the dictionary of synonyms

the chromatograms are obtained from previous procedures.

Here the intensities are not taken into account and are set equal to 1 for every peak.

this makes it possible to know the local frequencies

res.rank contains the indice of the identifier in the dictionary.

therefore `res.i(4,:)` is identical with `alias(rank(4),:)`

this is necessary to keep the actual code in chromatogram, as obtained by `decode_aroma1/2`

[descriptor list](#)

`function [res] = descriptor_list(alias,dico)`
create a list of descriptors ACTUALLY used
when knowing a list of alias (synonyms)

Exemple:

```
[res] = descriptor_list(alias,dico)
```

Cette fonction donne la liste des descripteurs finaux, ainsi que leurs occurrences, après remplacement des descripteurs originaux dans la collection d'olfactogrammes.

`alias` est le résultat de la fonction `create_alias`, et `dico` est le résultat de la fonction `find_descriptor`

Attention! Les structures `alias` et `dico` ne doivent pas avoir été modifiées par des outils tels que `deleterow`.

Cette fonction est nécessaire dans la logique de création de matrices de similitude. En effet `res.i` contient les descripteurs finaux, qui peuvent être utilisés dans le logiciel `matsim`.

Ce logiciel demande un fichier texte contenant la liste des descripteurs finaux.

On peut les sauvegarder en ASCII par la commande `string2text`. Par exemple:

```
string2text(res.i,'demo')
```

[excel2aroma](#)

Builds the SAISIR file of aromagram

```
function [chromato] = excel2aroma(filename, alias,mrange,(startchro),(endchro))
```

The Excel files must be saved as ".CSV" files (delimiter: ";")

VERY SPECIFIC VERSION

gives in succession the intensity and the code as defined by `alias`

`alias` is the dictionnaire of synonyms as created by `create_alias`;

`mrange` is larger than the highest retention indice

`startchro` and `endchro` are the number of the first and last chromatogram in `chromato`.

(This is necessary for memory manipulation: no more than 2000 chromatograms can be processed in a single turn)

Exemple:

```
chromato=excel2aroma('t5',alias,3000);
```

`Chromato` contient les données numérisées avec un pas de 1 indice de rétention. L'indice de rétention maximal est de 3000.

Le résultat est une structure SAISIR dans laquelle les lignes donnent successivement l'intensité perçue et le code descripteur final. `alias` permet d'établir la correspondance entre le descripteur original et le descripteur final (synonyme).

Si le fichier de départ est très grand, la mémoire disponible peut être insuffisante. Dans ce cas, il faut décoder par morceau. Les arguments optionnels (nombres entiers) `startchro` et `endchro` permettent de réduire la lecture aux seuls olfactogrammes dont les numéros de rangs sont compris entre `startchro` et `endchro`. Les fichiers résultats peuvent être concaténés ensuite par `appendrow`.

Le champ `chromato.v` donne les indices de rétention par pas de 1, et `chromato.i` les noms des olfactogrammes. Ces noms apparaissent deux fois, puisque un olfactogramme est formé de deux lignes.

Cette fonction est très dépendante de la manière dont les données ont été enregistrées. Elle doit être réécrite à chaque changement de format! La création d'une fonction de ce genre est indispensable pour l'exploitation des données d'aromagrammes.

[excel2wine](#)

Decodes excel files of wine olfactograms

`function res=excel2wine(filename, mrange, (nchar))`

Decodes olfactogram data according to the Laboratoire des aromes (Dijon) format

`filename` is the ".CSV" Excel file to be decoded

`mrange` is the largest retention indice possibly encountered

`nchar` (optional) the largest number of characters in a descriptors (extra ignored)

return:

`res.occurrence`: the descriptors and their occurrences

`res.dico` : the dictionnary (here the descriptors have no synonyms)

`res.chromato` : chromatogram data in the saisir format (successin intensity/code number)

The number of rows is then twice the number of chromatogram

DB 31/03/2003

Dans les essais de Dijon (mars 2003), les utilisateurs ne souhaitent pas créer un dictionnaire de synonymes. Pour cette raison, `res.dico` forme un dictionnaire où chaque descripteur est son propre synonyme. Ainsi, toutes les procédures qui demande un dictionnaire de synonyme (**alias**) peuvent fonctionner en passant `dico` comme argument.

Les olfactogrammes sont dans `chromato`.

Par exemple, pour utiliser `aroma_curve`, l'ordre est:

`aroma_curve(res.chromato, res.dico, 3)`

(représentation du 3^{ème} olfactogramme, contenu dans `res.chromato`).

[find_descriptor](#)

`function [dico] = find_descriptor(chaine)`

Extrait les descripteurs d'un fichier d'aromagrammes.

La première étape de l'exploitation des données d'olfactogramme consiste à faire l'inventaire de tous les descripteurs utilisés par les expérimentateurs.

La fonction renvoie une liste complète des descripteurs utilisés, quelle que soit leur position dans les olfactogrammes.

`chaine` indique le fichier sur disque dur contenant l'information.

Exemple:

```
res=find_descriptor('t5');
```

```
res =
```

```
  i: [342x30 char]  
  d: [342x1 double]  
  v: 'occurrence'
```

```
  ndescri: 2513
```

```
  npic: 2513
```

```
  nchromato: 105
```

'T5' est le nom du fichier sur le disque dur.

res.i contient la liste des descripteurs, en ordre alphabétique, en tenant compte des accents, mais pas de la différence majuscule/minuscule. Les espaces et les accents sont significatifs. **res.d** contient le nombre d'occurrences de ces mots (c'est à dire du nombre de fois que ce descripteur a été utilisé, dans l'ensemble des aromagrammes).

ndescri et **npic** contiennent le nombre de pics nommés et le nombre de pics (qui normalement sont identiques).

nchromato est le nombre d'olfactogrammes du fichier décodé.

Cette fonction est l'une des plus dépendantes de la manière dont les données ont été enregistrées. Elle doit être réécrite à chaque changement de format! La création d'une fonction de ce genre est indispensable pour l'exploitation des données d'aromagrammes.

[kassign_peak1](#)

Attempt to identify relevant peaks in olfactogram data

(KCmeans approach)

```
function [res] =
```

```
kassign_peak(chroma,alias,similitude,(niter),(void_group),(semantic_factor),(window_size),(intensity_factor))
```

Cette fonction est fondamentale puisqu'elle effectue l'opération complète de recalage des pics.

Arguments d'entrée:

chroma: fichier des alfactogrammes, résultat de excel2aroma

alias: fichier des synonymes, résultat de create_alias

similitude: matrice de similitude, créée avec le logiciel Matsim, chargée par matsim2saisir

niter: nombre d'itérations avant de finir, défaut: 20. La procédure s'arrête plus tôt si aucun pic ne change d'attribution pendant l'itération.

void_group: (nombre entier) taille d'un groupe considéré comme vide. Au cours des itérations, un pic de référence auquel est associé moins de void_group pics disparaît, défaut : 5.

semantic factor: importance donné au sens des descripteurs dans le recalage. Une valeur élevée veut dire que l'on refuse de mettre dans le même groupe de pic des descripteurs ayant des sens très différents, défaut: 5.

window_size: éloignement initial des pics de référence. Devrait être de l'ordre de grandeur de la dérive incontrôlée des indices de rétention. Une petite valeur veut dire que l'on a au début beaucoup de pics de référence, défaut : 10.

intensity_factor: indice permettant de favoriser l'agglomération des pics dont les intensités sont voisines. (On suppose qu'un pic est peu intense, quel que soit le produit). Ne donne pas des résultats intéressants, laisser la valeur par défaut (0.0001). Attention ! La valeur de cet indice est en relation avec l'échelle des intensités utilisée. .

Exemple d'utilisation:

picstest=kassign_peak1(chromato,alias,SA,15,5,20,8,0)

Applique la procédure sur le fichier chromato, avec le dictionnaire de synonymes alias, la matrice de similitude SA, nombre d'itération=15, la taille des groupes considérés comme vide fixée à 5, le facteur sémantique fixé à 20, la taille des fenêtres à 8, et le facteur d'intensité à 0.

Début de l'essai:

Total number of peaks = 2484
Iteration number 1 Moved = 1000
Number of reference peaks = 376
Iteration number 2 Moved = 2484
Number of reference peaks = 146
Iteration number 3 Moved = 2484
Number of reference peaks = 144
Iteration number 4 Moved = 2017
Number of reference peaks = 143

Dans la première itération, il y a 376 pics de référence qui correspond à un écartement régulier entre les pics égal à window_size ($376 = 3000/8 + 1$)

Les pics sont déplacés ("*Moved*") de manière à se rapprocher des pics de référence.

Au cours des itérations suivantes, les pics "peu attirants" disparaissent.

Au bout de 15 itérations on a l'affichage suivant:

Iteration number 15 Moved = 7
Number of reference peaks = 142

Il n'y a plus que 7 pics qui se sont "déplacés" (parmi 2484 pics initiaux), et ce n'est pas utile de continuer.

Les résultats sont dans picstest

```
picstest =  
  assigned: [1x1 struct]  
  ref: [1x1 struct]  
  merged: [1x1 struct]  
  merged_ref: [1x1 struct]
```

Le champ **assigned** (attribué) contient la liste des pics recalés, suivant la même structure que **chromato**. Un pic se réduit à sa position de référence.

Le champ **ref** (référence) donne la liste des pics de référence. **ref.d** contient 4 lignes, qui correspondent aux informations suivantes:

start : indice de rétention de départ de pic
end : indice de rétention de fin de pic
inten : intensité moyenne rencontrée.
code : code final associé au dictionnaire des synonymes.

On a ainsi tous les noms des descripteurs de référence par la commande:

```
alias.i(picstest.ref.d(4,:),:)
```

Les champs **merged** (concaténés) et **merged_ref** ont exactement la même logique.

Ils donnent les pics de référence, après que la fonction **kassign_peak1** ait fusionné les pics proches qui ont un descripteur identique.

[merge peak](#)

merges two peaks in one in aroma chromatograms

```
function res = merge_peak(assigned,alias,vector)
```

Assigned : peaks assigned in experiment. **alias**: dictionary of synonyms.

vector: retention indice of the first (left) peak to be merged.

For example, if **vector** contains value: 561, and that the following index is at 570, this two peaks will be merged.

Autre exemple : supposons les positions de référence (dans **assigned**) à :

```
561 570 582 612 614 700 800
```

si **vector** possède les deux éléments 561 612,

-Les pics à 561 et 570 sont fusionnés pour former le pic à $(561+570)/2 = 567$ (arrondi)

-Les pics à 612 et 614 sont fusionnés pour former le pic à $(612+614)/2 = 613$

Le fichier resultat contient donc les pics

```
567 582 613 700 800.
```

Il ya des règles logiques pour tenir compte de fusions difficiles (2 descripteurs différents). Le nombre de cas de ce genre est donné par la valeur de la variable **nproblem** (affichée à la fin des calculs).

B 2 Exemple de script simple

Premier script (très simple)

```
chro=excel2wine('es11',3000);  
browse_aroma(chro.chromato,chro.dico) ;  
[res,occur]=descri2(chro.chromato,chro.dico,5) ;  
browse(res) ;
```

Explications :

```
chro=excel2wine('es11',3000);
```

La structure `chro` est créée à partir du fichier `es11.csv` (créé sous Excel, et respectant la structure 'Dijon').

Cette structure contient :

`chro =`

`occurrence: [1x1 struct] % occurrences des descripteurs`

`chromato: [1x1 struct] % olfactogrammes au format standard`

`dico: [1x1 struct] % dictionnaire : aucun synonyme`

```
browse_aroma(chro.chromato,chro.dico) ;
```

Affichage des olfactogrammes. On passe en argument les olfactogrammes (dans `chro.chromato`) et le dictionnaire (dans `chro.dico`). En plaçant le curseur dans l'image, et en cliquant avec le bouton droit de la souris, on peut examiner les olfactogrammes les uns après les autres.

```
[res,occur]=descri2(chro.chromato,chro.dico,5) ;
```

Etablissement du profil des descripteurs.

Le paramètre '5' indique l'écart-type de la gaussienne.

`res` contient le profil des descripteurs, au format SAISIR. Chaque ligne correspond à un descripteur.

```
browse(res) ;
```

Affichage du profil de chaque descripteur (dans l'ordre alphabétique).

Annexe C : Pourquoi ça plante ?

N'oubliez pas de signaler des bugs ou des problèmes à christophe.cordella@fsaa.ulaval.ca

Cette annexe tente de donner des idées pour rechercher la cause d'un dysfonctionnement de SAISIR dans l'environnement MATLAB

1) Aucune des fonctions de SAISIR n'est reconnue

Exemple :

Je tente d'exécuter la fonction <xxxx>, et je reçois l'avertissement Matlab
?? Undefined function or variable 'xxxx'.

Diagnostic :

Tapez help saisir

Si la réponse est

?? Undefined function or variable 'saisir'.

Cela veut dire que le répertoire de SAISIR n'est pas dans le chemin des répertoires de Matlab.

Dépannage :

Aller dans le menu File ; set path ; add folder de Matlab, et ajouter le répertoire SAISIR qui doit exister et contenir les fonctions de SAISIR. Demander à ce que cette modification soit conservée d'une session à l'autre.

2) Problème de format

Exemple :

J'ai un fichier X , je tape X1=pca(X,) (acp sur X) et je reçois l'avertissement Matlab :

```
> x1=pca(x)
```

```
??? Reference to non-existent field 'd'.
```

```
Error in ==> c:\matlab\saisir\pca.m
```

```
On line 17 ==> [n,p]=size(saisir.d);
```

Diagnostic :

Utilisez l'ordre saisir_check pour identifier l'erreur de format

```
>> saisir_check(x)
```

```
FATAL ERROR: the field 'd' does not exist. See the manual for more information,  
or/and function MATRIX2SAISIR
```

```
The manual is in the Word file 'SAISIR.DOC'
```

Dépannage :

Tenter d'examiner les champs qui forment **X**. Si **X** est une matrice Matlab (et non une structure SAISIR), on peut rapidement le mettre au format par la commande **X=matrix2saisir(X)**. Dans ce cas, les identificateurs **X.i** et **X.v** sont simplement les numéros des indices. C'est rustique mais ça marche.

3) Problème de chargement de fichier

Ces problèmes sont (hélas) les plus difficiles à résoudre

3.1 Chargement avec excel2saisir

Supposons que l'on veuille charger le fichier 'donnee.xls'

La commande SAISIR est alors par exemple : `res=excel2saisir('donnee')` ;

Faites les vérifications suivantes et les corrections appropriées :

- Est ce que le fichier `donnee.xls` a bien été mis au format `donnee.csv`, qui est le seul format reconnu par SAISIR (voir la commande `excel2saisir`).

- Est ce que le fichier '`donnee.csv`' est bien dans le répertoire de travail de Matlab actuel ?

- Est-ce que le séparateur décimal dans Excel est bien le point et non la virgule ?

- Est ce que le format d'entrée est bien celui donné dans le manuel (voir paragraphe 2.1.1). Il faut impérativement que la première ligne soit formée d'identificateurs, de même que la première colonne. La feuille excel ne doit pas comprendre de lignes ou de colonnes vides, ni des commentaires ou des dessins.

- Les descripteurs dans Excel ne doivent pas contenir de retour chariot (saut de ligne), ni de virgule.

Lorsque le fichier est chargé, utiliser éventuellement `saisir_check` pour vérifier le format.

3.2 Chargement divers

Lorsque l'on a, par une commande Matlab ou par un ordre « copier coller », réussi à charger une matrice X Matlab, on peut la transformer en fichier SAISIR par l'ordre `X=matrix2saisir(X)`.

Pour avoir des identificateurs spécifiques, on peut écrire des ordres du type :

`donnee.d=X` ;

`donnee.i=nom_ligne` ;

`donnee.v=nom_colonne` ;

Dans cet exemple, les noms des lignes ou des colonnes sont créés en dehors de l'environnement SAISIR.

Attention ! les champs `.i` et `.v` sont des matrices de caractères et non des cellules.

On peut éventuellement utiliser l'ordre `char` pour convertir une cellule en chaîne de caractère.

Par exemple :

`donnee.i=char(nom_ligne)`;

Les données provenant d'appareils de mesure (chromatographe, spectromètres ...) demandent en général que l'utilisateur écrive des fonctions spécifiques ! Voir l'ordre Matlab `dload`.

4.1 difficultés associées à certaines commandes saisir

D'une manière générale, l'ordre `saisir_check` indique si le format des matrices SAISIR est correct. Les « warnings » (avertissements) de cette fonction doivent être examinés avec attention.

Par exemple, la présence de données manquantes n'est généralement pas tolérée par les commandes de SAISIR. Seules les difficultés spécifiques aux fonctions sont présentées ici.

[anovan1](#)

Il faut vérifier dans le champ `singular` de l'argument de sortie si les données ne sont pas singulières, c'est à dire qu'il y a assez de répétitions pour utiliser le modèle avec les interactions demandées. Si il y a une singularité, il faut réduire le degré des interactions à tester. Il s'agit du deuxième paramètre d'entrée (`model`).

[appendcol, appendcoll,appendrow,appendrow1](#)

Confusion entre les lignes et les colonnes.

Nombre de lignes ou de colonnes non identiques dans les arguments. Matrice totalement vide !

[comdim](#)

Les collections de signaux sont dans un vecteur de cellule (accolade `{}`) et non parenthèse `()`. Les tableaux regroupés doivent avoir le même nombre de ligne

[cormap](#)

En changeant l'ordre des arguments d'entrée, on change les variables qui sont placées en lignes ou en colonnes. Il faut en général choisir l'ordre le plus pratique

[create_group1](#)

L'ordre des groupes ne dépend pas de l'ordre alphabétique ou numérique des codes. Par exemple, les codes '1', '2', '3' ne sont pas forcément attachés aux groupes 1, 2, 3.

[d2_factorial_map](#)

Un paramètre d'entrée est bien la distance au carré.

[deletecol, deleterow](#)

Confusion entre lignes et colonnes

[excel2saisir](#)

`excel2saisir(filename)`

filename est bien une chaîne de caractère (entre ' '). Le fichier à charger doit être un fichier avec l'extension `.csv`, selon le format impérativement demandé. (voir le détail dans la liste des fonctions et dans le début de ce manuel).

[isi2saisir](#)

`isi2saisir1`

`isi2saisir2`

`isi2saisir3`

`isi2saisir4`

Les fabricants d'appareils ont tendance à changer de format sans prévenir !

Hélas, il faut réétudier le chargement au cas par cas.

Un bug connu : absorbances négatives, que `isi2saisir4` sait traiter.

[labelled_hist](#)

Ne marche que pour des tableaux ayant assez peu de lignes (moins de 500)

Il faut tester des valeurs de `nclass` et `charsize` pour avoir un graphique lisible.

[list](#)

Ne fonctionne que sur de très petits tableaux. Sinon, il faut sauvegarder les données par `saisir2excel` et les examiner sous Excel.

[msc](#)

Si un spectre a un écart type nul, la ligne sera formée de NaN.

[norm_col](#)

Résultats incertains si une colonne a un écart-type nul

[normed_pca](#)

Résultats incertains si une colonne a un écart-type nul

[nuee](#)

Le nombre de groupes en résultats peut-être inférieur à celui demandé, si des groupes se sont trouvés être vides au cours d'une itération

[pca](#)

Plante si le fichier contient des valeurs NaN.

[plotmatrix1](#)

Ne donne des résultats lisibles qu'avec des matrices ayant peu de lignes et de colonnes

[random_select](#)

Retourne un vecteur MATLAB, et non une structure SAISIR.

[readexcel](#)

Sauf exception, ne pas utiliser cette commande directement, passer par `excel2saisir`.

[reorder](#)

Renvoie une erreur si l'une des matrices argument d'entrée contient des identificateurs de lignes identiques. Si c'est le cas, il faut retirer les observations ayant le même idetntificateurs. Si la moyenne des observations de même nom est une solution acceptable, on peut sans sortir par l'ordre `split_average`.

[ridge_regression](#)

`krange` est un vecteur MATLAB et non une structure saisir.

`krange` est difficile à choisir. Voir `pca_ridge_regression`.

[saisir2excel](#)

Le premier argument est une matrice SAISIR. Le deuxième est une chaîne de caractère.

Le résultat est un fichier « .CSV ». Pour le lire correctement, il faut le charger depuis Excel, et non cliquer sur son icône à partir de l'explorateur windows (c'est là faute à Excel !).

[saisir_check](#)

Les *warnings* sont à considérer avec attention ! Les programmes de régression ou d'analyse discriminante ne supportent pas les colonnes d'écart-type nul. La plupart des fonctions de SAISIR refuse les données manquantes.

[show_vector](#)

Donne un graphique illisible si les identificateurs des variables sont très longs, ou qu'il y a trop de variables. On peut toujours faire des graphiques partiels en découpant (par `selectcol`) le vecteur à examiner.

[split_average](#)

Les moyennes sont dans le champ `average` de la sortie de cette fonction.

[stepwise_regression](#)

La fonction retourne un argument de sortie qui contient un champ `res`. Ce champ est lui-même un vecteur de cellules (*array of cells*) . Chacune des cellules correspond à un pas de

l'algorithme stepwise. De plus le dernier élément de **res** est parfois vide, c'est à dire que l'introduction des variables s'est arrêtée à l'étape précédente.

[xcomdim](#)

Pas d'utilisation directe, passer par **comdim**.