Module Optique Solaire
Mastère M1 Observatoire

Une courte introduction à IDL

Jean-Marie Malherbe

Documents de référence:

- le manuel de l'utilisateur d'IDL, en anglais
- Le cours de Jean Aboudarham, en français

La philosophie du langage IDL

IDL est issu du logiciel interactif de traitement de données solaires ANA conçu par le groupe de physique solaire de la NASA au début des années 1980 pour le traitement des données du satellite Solar Maximum Mission (Goddard Space Flight Center, Washington)
Aujourd'hui, IDL s'est imposé comme LE système de traitement universel utilisé en physique solaire, et son succès déborde désormais vers tous les horizons astronomiques.

IDL tourne sur toutes plateformes: Windows, Mac, UNIX.
C'est un langage TRES CONCIS qui manipule les images 2D ou 3D aussi simplement que des scalaires. Mais il ne faut pas programmer comme en Fortran au risque d'écrouler les performances. IDL est coûteux… Un clone gratuit, GDL, est en cours de développement: c'est donc GDL que vous allez utiliser.

Comment écrire une procédure IDL (fichier .pro)

Un programme IDL se rédige à l'intérieur d'un éditeur de texte indépendant selon :

```
Pro nom_programme
  Suite d'Instructions
end
```

Si ce fichier programme porte le nom *nom_programme.pro* on le compile en tapant sur le prompt IDL la commande suivante (ne pas oublier le point devant run):
IDL> .run *nom_programme*
puis on l'exécute en appelant le programme directement par son nom :
IDL> *nom_programme*

On peut passer des variables à une procédure en entrée sortie. Par exemple avec var_in en entrée et var_out en sortie:

```
Pro nom_programme, var_in, var_out
  Suite d'Instructions
End
```

puis on exécute la procédure en appelant le programme directement par son nom :
IDL> *nom_programme, var_in, var_out*

Les variables dans IDL

IDL travaille avec des variables de tout type :

- octets 8 bits de 0 à 255 (type BYTE)
- entiers courts 16 bits signés de –32768 à 32767 (type FIX ou INT)
- entiers courts 16 bits non signés de 0 à 65535 (type UINT)
- entiers longs 32 bits signés (type LONG)
- entiers longs 32 bits non signés (type ULONG)
- réelles 32 bits dits simple précision (type FLOAT), 7 chiffres significatifs entre $\pm 10^{38}$
- réelles 64 bits dites en double précision (type DOUBLE), 14 chiffres significatifs
- complexes 64 bits (type COMPLEX), 7 chiffres significatifs entre $\pm 10^{38}$
- complexes 128 bits (type DCOMPLEX), 14 chiffres significatifs entre $\pm 10^{308}$
- chaînes de caractères (type STRING)

exemples :

a = 400 définit un entier court

a = 100000 définit un entier long, et a = long(400) force à définir un entier long

a = 3.4e-7 définit un réel simple précision (en maths 3.4 10^{-7})

a = -3.1d-7 définit un réel double précision (en maths -3.1 10^{-7})

a = double(2.8) force à définir un réel en double précision

a = complex(1.5, 3.5) définit un complexe (partie réelle, partie imaginaire) en simple précision, ou en double précision par a = dcomplex(1.5, 3.5)

a = ‘GDL en M1’ définit une chaîne de caractères

Conversions de types :

a = byte(b) conversion en octet 8 bits
a = fix(b) conversion en entier court 16 bits
a = uint(b) conversion en entier court non signé
a = long(b) conversion en entier long 32 bits
a = ulong(b) conversion en entier long non signé
a = float(b) conversion en réel 32 bits (flottant simple précision)
a = double(b) conversion en réel 64 bits (flottant double précision)
a = complex(b,c) conversion en complexe 64 bits (2 flottants simple précision)
a = dcomplex(b,c) conversion en complexe 128 bits (2 flottants double précision)
a = string(b) conversion en chaîne de caractères

Les vecteurs (1D), tableaux ou matrices (2D ou plus)

Si dimx, dimy, … sont les dimensions du vecteur 1D ou du tableau (2D ou plus) :

Tab=bytarr(dimx,dimy,…) définit une matrice de type octet (BYTE)
Tab=intarr(dimx,dimy,…) définit une matrice de type entier court (FIX)
Tab=lonarr(dimx,dimy,…) définit une matrice de type entier long (LONG)
Tab=fltarr(dimx,dimy,…) définit une matrice de type réel (FLOAT)
Tab=dblarr(dimx,dimy,…) définit une matrice de type réel (DOUBLE)
Tab=complexarr(dimx,dimy,…) définit une matrice de type complexe (COMPLEX)
Tab=dcomplexarr(dimx,dimy,…) définit une matrice de type complexe (DCOMPLEX)

Attention aux indices d'adressage d'un tableau qui partent toujours de 0:
***Tab(x,y) est défini pour* $0 \leq x \leq dimx-1$ et $0 \leq y \leq dimy-1$**

Ces fonctions sont dynamiques, contrairement au FORTRAN. Les dimensions dimx, dimy,… peuvent être des variables résultant d'un calcul, et les tableaux peuvent être créés et dimensionnés en cours de programme. Les éléments sont initialisés à 0 lors de la création.

Informations sur une variable ou un tableau

help, tab *fournit les dimensions du tableau et son type de données*

Règles d'adressage des éléments d'un tableau et règles de redimensionnement

Tab ou Tab(*,*) représente la totalité du tableau de valeurs
Tab(50:100,*) est constitué des colonnes 50 à 100 et de toutes les lignes
Tab(50:100,25 :*) est constitué des colonnes 50 à 100 et des lignes à partir de la ligne 25
Tab(50,*) est un vecteur qui n'adresse que la colonne numéro 50
Tab(x,y) est l'élément situé à l'intersection de la colonne x et de la ligne y

Exemples :
T = 2.* Tab définit un nouveau tableau T égal au double du tableau Tab, de même dimension.
T = Tab(30:55,150:200) définit un nouveau tableau T contenant les colonnes 30 à 55 et les lignes 150 à 200 de Tab. Les dimensions de T sont donc 26 x 51. Les éléments de T sont donc numérotés de 0 à 25 (colonnes) et de 0 à 50 (lignes).
T = Tab(*,10) est un vecteur qui contient la ligne 10 de Tab.
T = Tab(15,23) est un scalaire

Lorsqu'on extrait d'un tableau à n dimensions un sous tableau de dimensions n-1 ou moins, la fonction **reform** permet de supprimer la (les) dimension(s) devenue(s) inutile(s). Exemple :
Tab = fltarr(10,20)
T = Tab(*,2) va créer un tableau T de dimensions (10,1) correspondant à la 2ème ligne de Tab.
IDL n'a pas supprimé la seconde dimension devenue inutile.
T = reform(T) réorganise T en supprimant la dimension inutile;
T devient donc un vecteur de dimension (10)

Créer un tableau en l'initialisant

Les fonctions :
indgen (dimx, dimy,…) pour entiers courts
lindgen (dimx, dimy,…) pour entiers longs
findgen (dimx, dimy,…) pour réels 32 bits
dindgen (dimx, dimy,…) pour réels 64 bits

génèrent des matrices de dimension (dimx, dimy,…) et les initialisent avec les valeurs de leurs indices. Exemple :
Tab = findgen(100)
Génère un vecteur de 100 valeurs réelles initialisées de 0. à 99. par pas de 1.

Connaître les dimensions d'un tableau lu dans un fichier : fonction size

dim = size(Tab) retourne un vecteur dim
Ndim = dim(0) donne le nombre de dimensions du tableau
Dimx = dim(1) donne la première dimension, etc...
Dimy = dim(2)
Dimz = dim(3) …

Opérations arithmétiques courantes : +, -, *, / , ^ (élévation à la puissance)

Ces opérations courantes fonctionnent sur des scalaires ou sur des matrices, dans le cas des matrices les opérations sont réalisées **terme à terme.**
Si a et b sont deux matrices de dimensions et nombre d'éléments identiques, alors l'opération c = a * b réalise un produit terme à terme, et non pas un produit matriciel.
Il ne faut surtout pas utiliser de boucle comme en FORTRAN !

Multiplication matricielle : ##

a = b ## c est la multiplication matricielle lignes par colonnes des matrices b et c, à condition que le nombre de colonnes de b soit égal au nombre de lignes de c

Fonctions diverses sur les scalaires ou les matrices

abs(a) donne la valeur absolue ou le module de a (si a complexe)
real_part(a) donne la partie réelle de a si a complexe
imaginary(a) donne la partie imaginaire de a si a complexe
Ces opérations sont réalisées terme à terme si a est une matrice, et le résultat est une matrice

max(a) donne la valeur maximale de la matrice a
min(a) donne la valeur minimale de la matrice a
mean(a) donne la valeur moyenne de la matrice a
moment(a) donne les moments de divers ordres de la matrice a

Intégration selon les lignes ou les colonnes d'un tableau : fonction total

tabx = total(tab,1) intègre dans le sens de la première dimension
taby = total(tab,2) intègre dans le sens de la seconde dimension, etc…
tabt = total(tab) somme tous les éléments du tableau

tabmoy = mean(tab) *est identique à total(tab)/(dimx * dimy) si dimx, dimy sont les dimensions de tab*

Opérateurs logiques et de comparaison

AND, OR, EQ (=), NE (≠), LE (≤), LT(<), GE(≥), GT(>) *comme en FORTRAN*

Il faut y ajouter les opérateurs **<=, <, >, >=,** qui permettent d'affecter à une nouvelle variable la plus grande ou la plus petite de deux valeurs comparées.

Exemple :
a = b > c renvoie dans a la plus grande des deux valeurs b et c
a = b < c renvoie dans a la plus petite des deux valeurs b et c

Les opérateurs peuvent s'appliquer aux matrices. Par exemple, si tab est un tableau de valeurs, écrire tab = tab > 10 signifie que l'on affecte terme à terme à tab « la plus grande valeur entre 10 et tab », donc toute valeur inférieure à 10 est mise à 10 dans tab.

Fonction where

Cette fonction évite la réalisation de tests bouclant sur les indices d'un tableau, qui sont extrêmement pénalisants sous IDL. La fonction where renvoie les indices de tableau dans un vecteur lorsque le test qui suit la fonction est vérifié :

x = where (condition logique sur un tableau, count)

Renvoie dans le vecteur x les positions (indices) où le test est concluant et dans count (option facultative) le nombre d'occurrences concluantes

Exemple :
x = where (tab LT 10.)
Renvoie dans le vecteur x les positions où les éléments de tab sont inférieurs à 10, tab est ici un tableau de dimensions quelconques. On peut prendre ensuite une action comme tab(x) = 50.

x est un vecteur d'adresses.

Les chaînes de caractères

a = ‘Voici une chaine’ définit une chaîne de caractères
a = ’’ est une chaîne le longueur nulle
a = b + c effectue la concaténation des deux chaînes a et b
a = strarr(dim) définit un vecteur de chaînes de caractères de dimension dim

Quelques fonctions indispensables :

len = strlen(a) renvoie la longueur de la chaîne de caractères a
a = strupcase(a) convertit la chaîne a en majuscules
pos = strpos(a,b) renvoie la position de la sous chaîne b dans la chaîne a ; si b n’est pas une sous chaîne de a, on obtient –1
b = strmid(a, n, len) extrait len caractères de la chaîne a à partir de la position n. Le premier caractère de a est en position n=0.
Strput,a,b,pos place la chaîne de caractères b à la position pos dans la chaîne a en écrasant ce qui peut exister en dessous. Le premier caractère de a est en position 0.
a = strtrim(a) élimine tous les caractères blancs inutiles en fin de chaîne

Les structures de contrôle

- boucle FOR

```
for i=deb,fin,step do begin
  instructions
endfor
```

- boucle WHILE

```
while (test logique) do begin
  instructions
endwhile
```

Attention! Le test logique (initialisé à TRUE sinon on n'entre pas dans la boucle) doit être modifié à l'intérieur pour devenir à un moment donné FALSE, sinon on ne peut pas sortir !

- boucle REPEAT

```
repeat begin
      instructions
endrep until (test logique)
```

Attention! Le test logique est exécuté au moins une fois. Il doit être modifié dans la boucle, et devenir TRUE à un moment donné pour pouvoir en sortir !

•test IF

```
if (test logique) then begin
   instructions
Endif
```

•test IF/ELSEIF

```
if (test logique) then begin
   instructions
endif
else begin
   instructions
Endelse
```

•structure CASE simple

```
case (expression) of
   valeur1: instruction
   valeur2: instruction
endcase
```

•structure CASE/ELSE

```
case (expression) of
   valeur1: instruction
   valeur2: begin
                        instructions
  end
   valeur3: instruction
else: begin
                instructions
        end
endcase
```

•GOTO, label

Cette instruction est parfois utile pour sortir d'une boucle FOR.

```
Goto, label
instructions
label :
```

Entrées/sorties clavier/écran

- •afficher une valeur var à l'écran : print, var
- •saisir une valeur var au clavier : read, var

Entrées/sorties fichiers de données

En astronomie, les fichiers de données suivent un standard très stricte, le FITS (Flexible Image Transport System). Un fichier FITS se compose d'un en tête (ou HEADER) qui décrit les données sous forme de blocs de 2880 caractères, à l'aide de mots clef. Les mots clef de base sont :

BITPIX : le nombre de bits par pixel (exemple 16 si nombres entiers courts)
NAXIS : le nombre de dimensions
NAXIS1 : le nombre de pixels dans la première dimensions etc…
NAXIS2
NAXIS3…
L'en tête se termine toujours par le mot clef END.

Il est suivi par les données elles mêmes, sous forme de tableau de valeurs. Les fichiers FITS peuvent être constitués de nombres entiers courts, longs, réels flottants, sous forme de tableaux 1D, 2D, 3D, etc…

Les procédures de lecture/écriture de fichiers FITS ne font pas partie intégrante d'IDL : elles figurent dans IDLASTRO.

•lister un répertoire de fichiers FITS

fich = findfile(‘chemin/*.fts', count = nf)

*renvoie tous les noms de fichiers de type *.fts trouvés dans le répertoire vers le vecteur chaîne fich dont la dimension vaudra nf, nombre de fichiers présents. Fich est du type* STRARR *et a pour dimension nf.*

•lire un fichier FITS

tab = readfits(‘filename’)

ou tab = readfits(‘filename’, header)

range le contenu du fichier dans le tableau tab, et son en tête dans la variable header (chaîne de caractères), dont la lecture est optionnelle. Les données sont souvent des entiers courts issus de caméras CCD (16 bits, -32767 à +32767). Il est recommandé avant d’envisager tout traitement de les convertir tout d’abord en entiers non signés (0 à 65535) par la fonction :

tab=uint(tab)

puis de les convertir en nombres flottants (nombres réels 32 bits) par la fonction :

tab=float(tab)

On obtient les dimensions du tableau par la fonction :

dim = size(tab)

nd = dim(0) donne le nombre de dimensions

dimx = dim(1) donne la première dimension, etc…

dimy = dim(2)…

•écrire un nouveau fichier FITS

writefits,‘filename’,tab *écrit le tableau tab (2 ou 3 dimensions) dans le fichier* ‘filename’

•fichiers JPEG 8 bits

Ce type de fichier est restreint à 8 bits, de sorte que toute image devra être recadrée sur 8 bits en utilisant la commande bytscl de deux manières possibles:

a) Image=bytscl(tab,min=valeur_min,max=valeur_max)
Si tab(x,y) < valeur_min alors la fonction bytscl retourne 0
Si tab(x,y) > valeur_max alors la fonction bytscl retourne 255
Entre valeur_min et valeur_max, la fonction bytscl a une réponse linéaire de 0 à 255

b) Image=bytscl(tab)
Dans ce cas, la fonction bytscl calcule elle même automatiquement valeur_min et valeur_max à partir des valeurs de tab

Ecriture JPEG :
Write_jpeg,'filename',image

Inversement, on lit une image JPEG 8 bits par:
Read_jpeg,'filename',image

On ne traitera pas ici les fichiers JPEG 24 bits couleur.

•fichiers PNG 8 bits

Ce type de fichier est également restreint à 8 bits, de sorte que toute image devra être recadrée sur 8 bits en utilisant la commande bytscl comme pour le JPEG

Ecriture PNG :

Write_png,'filename',image

Inversement, on lit une image PNG 8 bits par:

Read_png,'filename',image

•fichiers binaires de sauvegarde réutilisables sous IDL uniquement (très utile) création d'un nouveau fichier avec sauvegarde des variables de tous types var1, var2…qui peuvent être des tableaux de données :

Save,var1,var2,filename='filename'

Récupération des variables dans une session IDL ultérieure (elles porteront le même nom, auront le même type et les mêmes dimensions que lors de la sauvegarde):

Restore, 'filename'

Ré-échantillonnage des tableaux

- ré-échantilonnage d'un tableau de dimensions initiales dimx, dimy par sommation des pixels

tab = rebin(tab,xdim,ydim)

attention ! xdim, ydim sont des sous multiples de dimx, dimy (procéder à un redimensionnement si ce n'est pas le cas avant de ré-échantilonner). Les pixels étant regroupés, la fonction rebin travaille par sommation et améliore le rapport signal/bruit de l'image initiale.

- ré-échantilonnage d'un tableau de dimensions initiales dimx, dimy par interpolation

tab = congrid(tab,xdim,ydim)

xdim, ydim sont quelconques (plus grands ou plus petits que dimx, dimy), donc la fonction congrid travaille par interpolation, et entraîne une perte de qualité : utiliser rebin quand l'amélioration du rapport S/B est requise.

Visualisation d'images

•créer une fenêtre de dimensions (dimx, dimy)

window, n, xs=dimx, ys=dimy *créer la fenêtre numéro n de la dimension du tableau*

n est le numéro de la fenêtre (à partir de 0)

sélectionner la fenêtre m : wset, m

détruire la fenêtre m : wdelete, m

•visualiser un tableau de dimensions (dimx,dimy) en niveaux de gris

window, n,xs=dimx, ys=dimy

tvscl, tab ou tv,bytscl(tab) *pour voir un tableau à 2 dimensions*

tvscl, tab(*,*,i) ou tv, bytscl(tab(*,*,i)) *pour voir le i-ème plan d'un tableau à 3 dimensions*

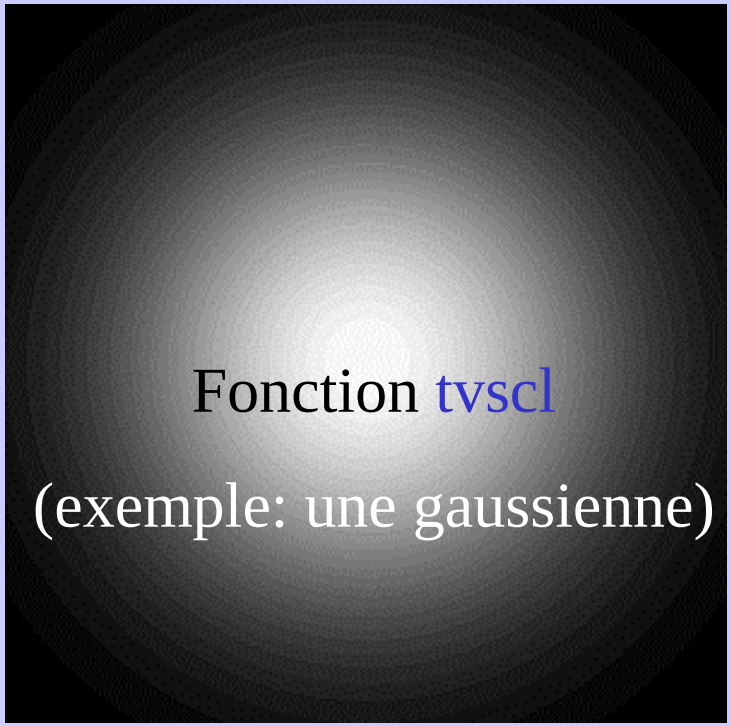
tv,bytscl(tab, min=valeur_min, max=valeur_max) *pour contraindre l'affichage des valeurs entre les valeurs min et max spécifiées par le programmeur. Dans ce cas :*

Si tab(x,y) < valeur_min alors la fonction bytscl retourne 0

Si tab(x,y) > valeur_max alors la fonction bytscl retourne 255

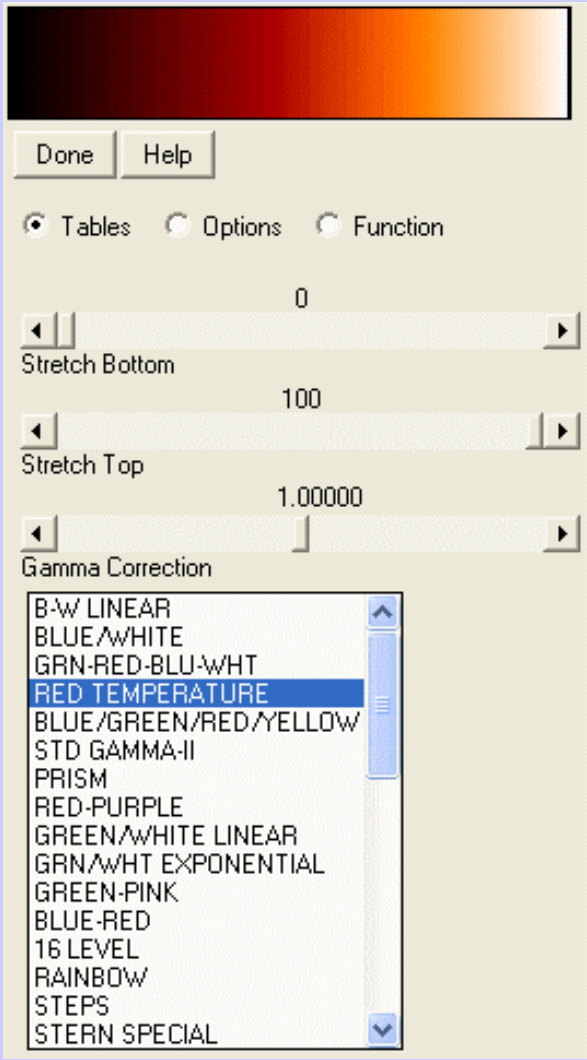
Entre valeur_min et valeur_max, la fonction bytscl a une réponse linéaire de 0 à 255

Quand on utilise tvscl, tab (ou tv, bytscl(tab), ce qui est strictement équivalent), la fonction bytscl calcule elle même automatiquement les valeur_min et valeur_max à partir des valeurs de tab.



Commande

xloadct →



•fausses couleurs en mode affichage 8 bits

Une palette de fausses couleurs peut être appliquée à la visualisation, pourvu qu'on soit en affichage 8 bits, par :

Loadct, n

Charge en mémoire la palette IDL prédéfinie numéro n (0 = niveaux de gris)

Choix interactif d'une palette prédéfinie par IDL: commande xloadct

Lire une palette pour la modifier (rend 3 vecteurs rouge, vert, bleu de type BYTE):
Tvlct,rouge,vert,bleu,/get

Charger une nouvelle palette utilisateur (3 vecteurs BYTE rouge, vert, bleu de dimension 256):
Tvlct,rouge,vert,bleu

•relever la position du curseur dans la fenêtre courante

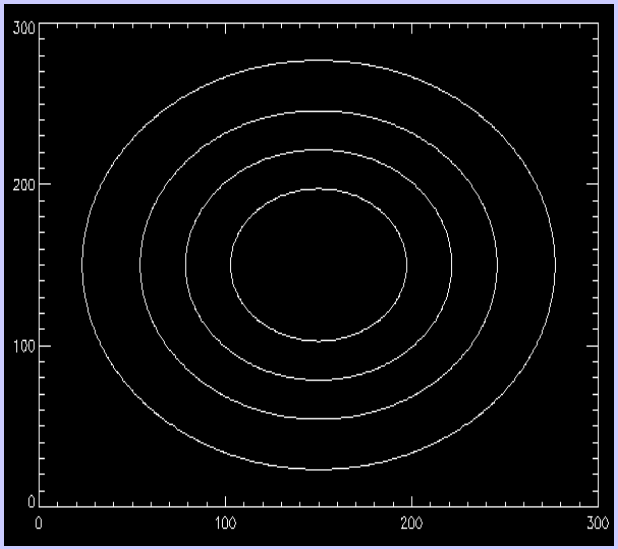
wset,n *sélectionner la fenêtre numéro n*
cursor, x, y, /device *renvoie les coordonnées du curseur (x,y)*

•Isocontours ou lignes de niveau d'un tableau

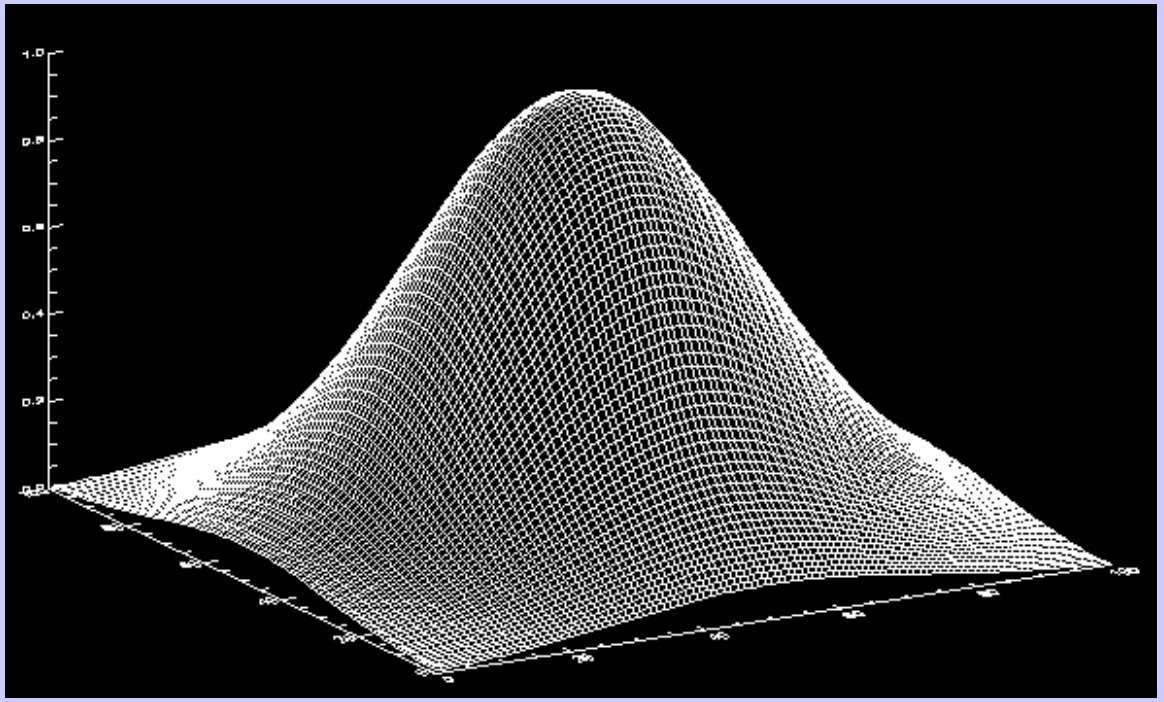
Contour,tab *pour un tableau à 2 dimensions*
Contour,tab(*,*,i) *pour le i-ème plan d'un tableau à 3 dimensions*

•Visualisation en 3D sous forme d'une surface composée d'un grillage

Surface,tab *pour un tableau à 2 dimensions*
Surface,tab(*,*,i) *pour le i-ème plan d'un tableau à 3 dimensions*



Fonction contour (exemple d'une gaussienne)



Fonction surface

•Superposition d'isocontours sur une image en niveaux de gris

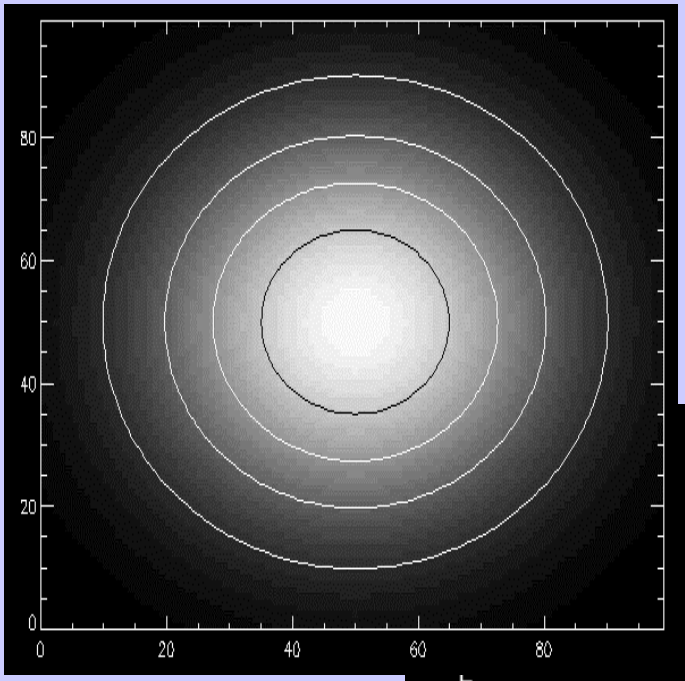
window,0,xs=dimx,ys=dimy

Image_cont,tab

•Visualisation en 3D sous forme d'une surface continue ombrée

Shade_surf,tab *pour un tableau à 2 dimensions*

Shade_surf,tab(*,*,i) *pour le i-ème plan d'un tableau à 3 dimensions*

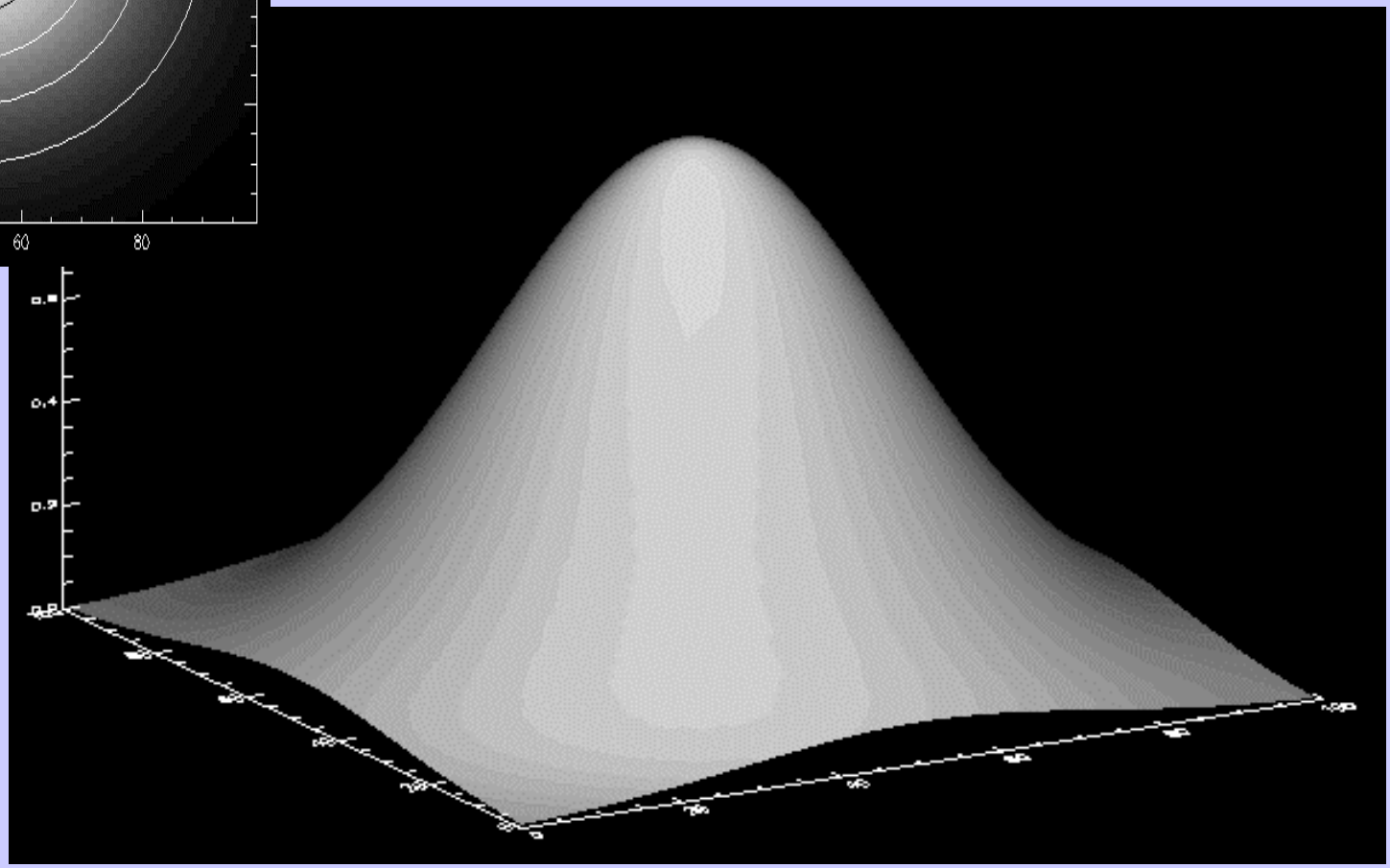


← Fonction image_cont

(exemple d'une gaussienne)

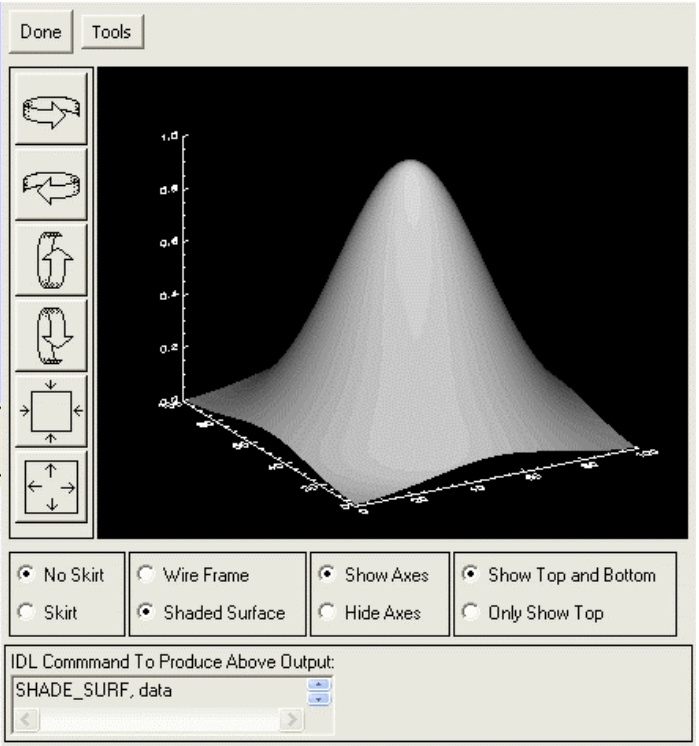
Fonction
shade_surf

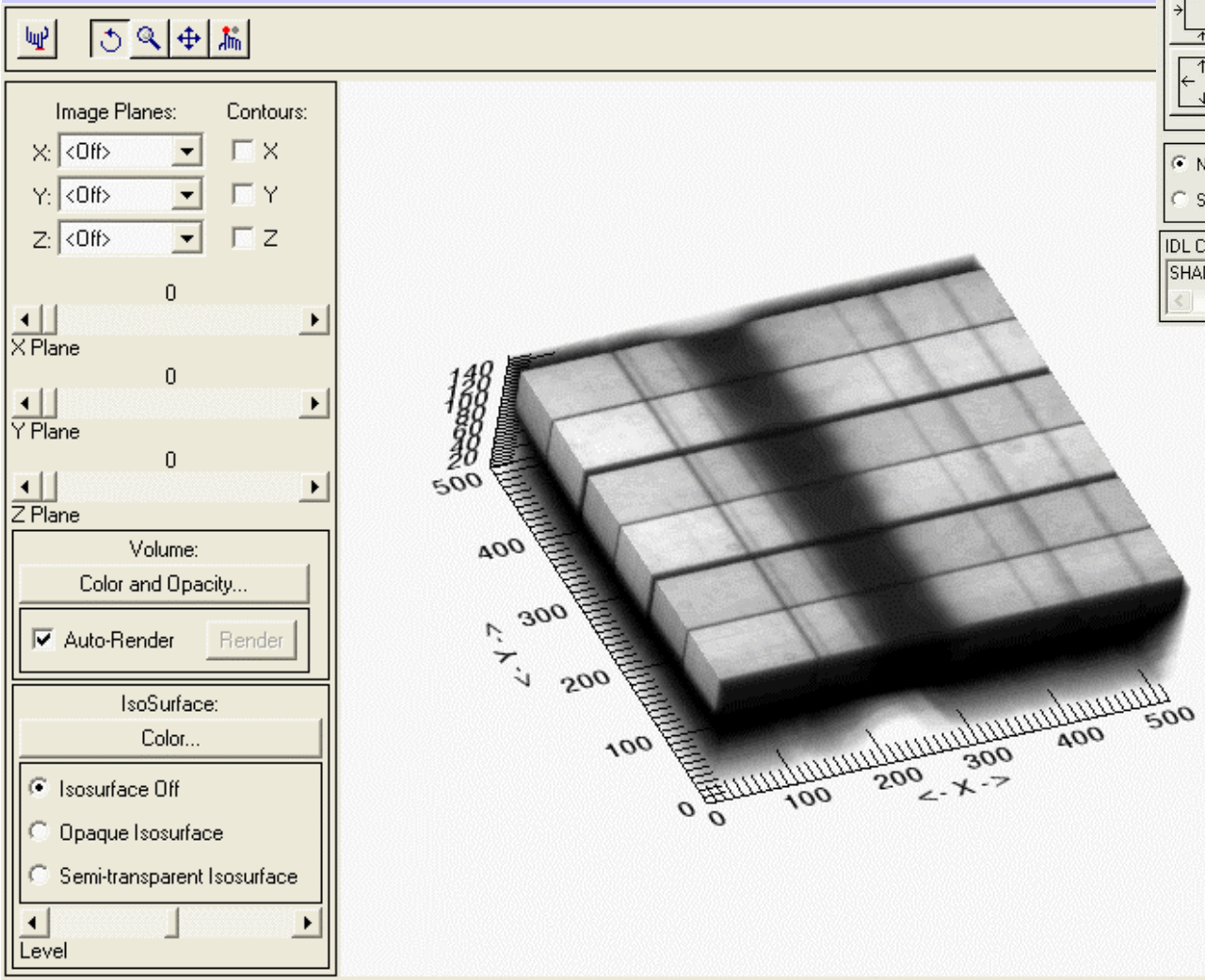
→



Visualisation de tableaux 2D →

xsurface,tab

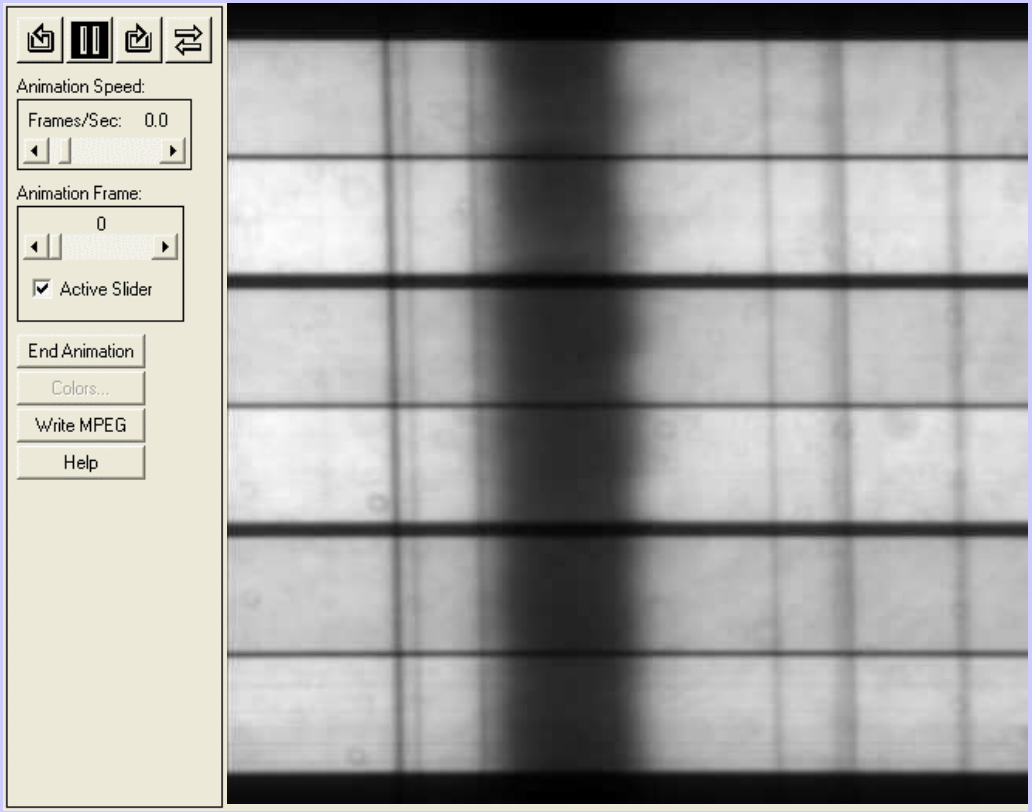




←Visualisation de tableaux 3D

xvolume,tab

Animation d’un tableau 3D




```
tab=readfits('ha.fts')

tab=float(tab)

dim=size(tab)

dimx=dim(1)

dimy=dim(2)

dimz=dim(3)
```

Initialiser XINTERANIMATE:

```
XINTERANIMATE, SET=[dimx, dimy, dimz], /SHOWLOAD , $
MPEG_QUALITY=100
```

Charger les images dans XINTERANIMATE:

```
FOR l=0,dimz-1 DO XINTERANIMATE, FRAME = l, IMAGE = bytscl(tab[*,*,l])
```

Jouer l'animation:

```
XINTERANIMATE, 10, /KEEP_PIXMAPS
```

Représentation graphique d'une fonction y = f(x)

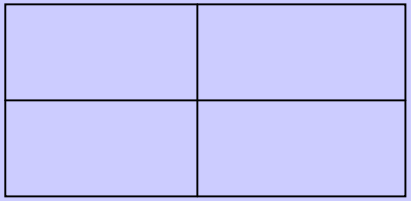
Les abscisses et ordonnées son contenues dans les vecteurs x et y.
window, n, xs=640, ys=480 *créer la fenêtre numéro n de la dimension souhaitée*

plot,x,y *où x = vecteur des abscisses, y = vecteur des ordonnées*
plot,tab(*,y) *pour faire une coupe horizontale d'ordonnée y dans un tableau tab*
plot,tab(x,*) *pour faire une coupe verticale d'abscisse x dans un tableau tab*

Quelques options intéressantes :
plot,x,y,linestyle=ls, thick=epaisseur,color=couleur,background=couleur, $
xrange=[xmin,xmax] ,yrange=[ymin,ymax], $
xtitle='Label sous axe Ox',ytitle='Label sous axe Oy',title='Titre du graphique', $
charsize=taille,charthick=epaisseur
ls = type de trait : 0 — 1 ····· 2 ----- 3 -.-.-.- 4 -…-…- 5 – – –
epaisseur du trait ou des caractères: nombre décimal (1 par défaut)
couleur : valeur entre 0 et 255 qui adresse la couleur d'une palette chargée par xloadct

On peut tracer plusieurs courbes sur le même graphique. Dans ce cas, la première courbe sera tracée avec la fonction PLOT. Les suivantes le seront avec la fonction OPLOT :
plot, x, y, linestyle=0, xtitle='Label sous axe Ox', ytitle='Label sous axe Oy'
oplot, x, y1, linestyle=1
oplot, x, y2, linestyle=2

On peut aussi tracer plusieurs graphiques dans une même fenêtre, en la divisant en N_{col} colonnes x N_{lign} lignes. A chaque nouvelle instruction PLOT, on passe à la division suivante de la fenêtre.



!P.multi = [0, N_{col} , N_{lign}]

Pour revenir au mode standard, faire !P.multi = 0

Sortie des graphiques en format POSTSCRIPT pour impression

Avant l'exécution des instructions graphiques, il faut spécifier le périphérique de sortie sous la forme d'un fichier postscript, car la sortie se fait sur l'écran par défaut :

```
Set_plot,'PS'
Device,filename='nom_du_fichier.ps',bits=8,/color
Instructions graphiques
Device,/close
```

L'impression du fichier se fait à l'aide d'une commande (hors IDL) du système d'exploitation.

On revient ensuite à l'affichage sur l'écran « X window » par :

```
Set_plot,'X'
```

Les fonctions utilisateur

```
Function function_name (var1, var2…)
  Instructions
Return, var
End
```

La fonction reçoit en entrée les variables *var1*, *var2*, … et fournit en sortie une unique variable var (contrairement aux procédures PRO qui peuvent en retourner plusieurs)

Appel: y = function_name (var1, var2, …)

Exemple:

```
Function sinc (x)
   Var = sin(x)/x
Return, var
End
```

Les fonctions peuvent être écrites et compilées dans des fichiers indépendants (d'extension .pro), ou placées à la suite ou au début du programme principal.
Il est possible de partager une grande quantité de variables entre les procédures et les fonctions sans les faire passer en argument en utilisant l'instruction COMMON :

```
Common nom, var1, var2, var3, …..varN
```

Caractères spéciaux dans une procédure

$ est le caractère de continuation d'une ligne à la ligne suivante. Un Exemple:

```
Print, var1, var2, $
var3, var4
```

& est le caractère qui sépare deux instructions tapées sur la même ligne. Exemple :

```
x = 12.5 & y = -34.6 & z = 56.9
```

; est le caractère qui précède un commentaire. Exemple :

```
x = 10.5 ; on initialise  la variable x
```

Documentation en ligne : taper sur le prompt IDL le caractère **?**

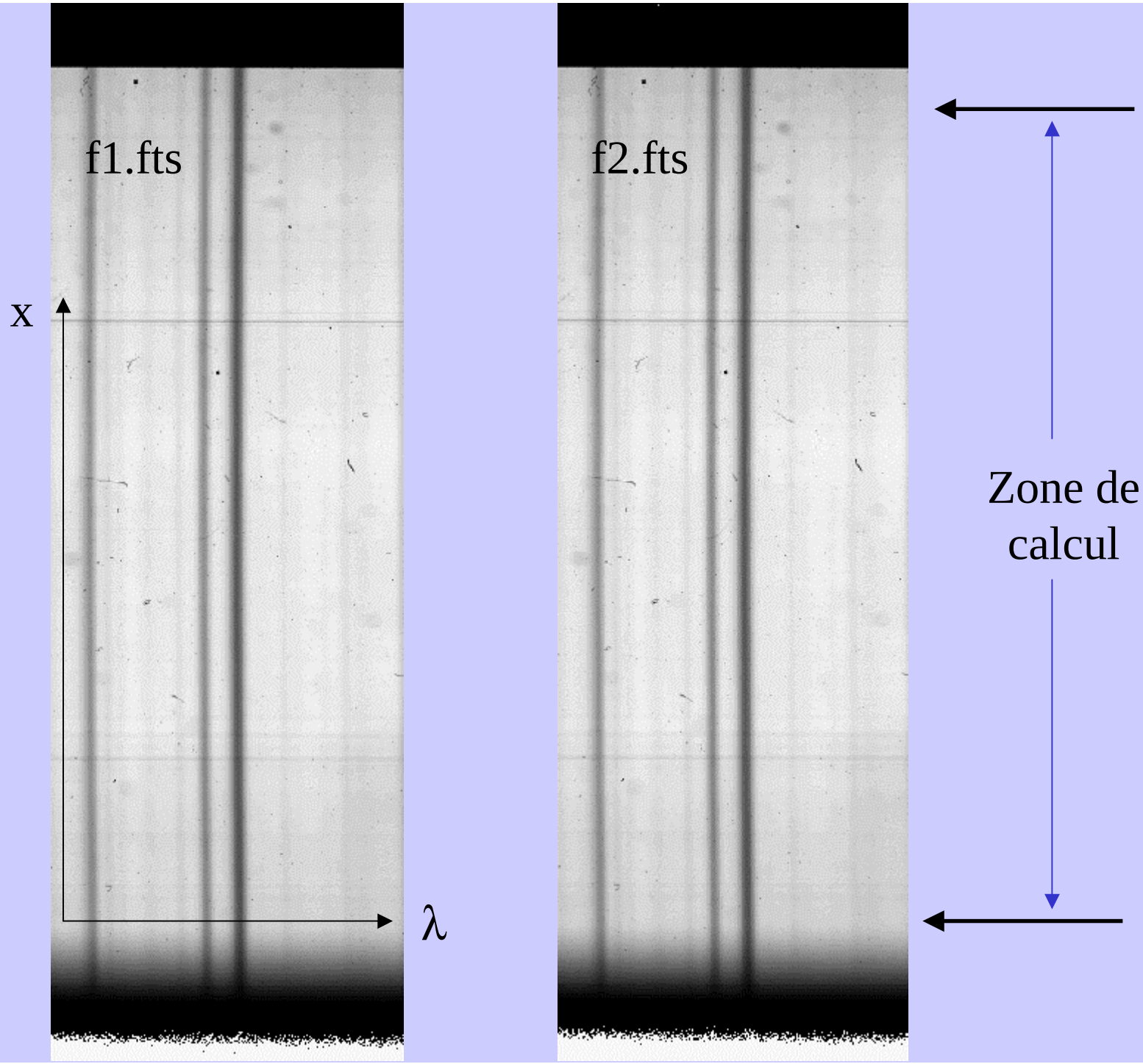
Variable d'environnement !PATH

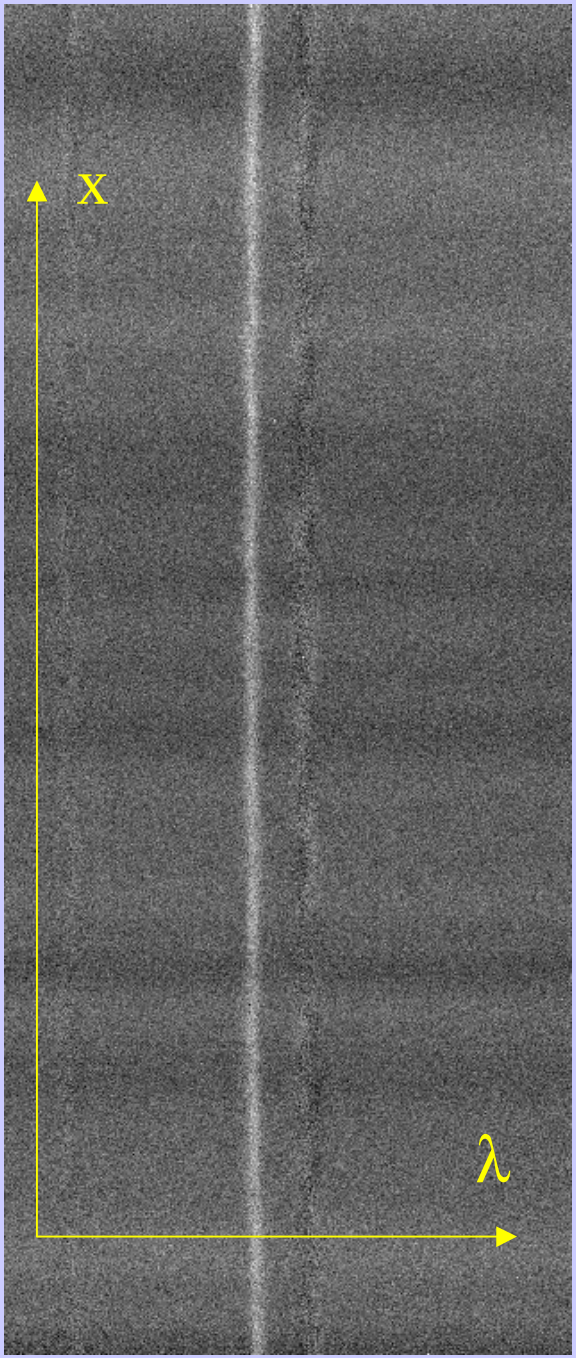
Cette variable permet de spécifier un chemin de recherche de procédures spécifiques à chaque utilisateur, le chemin par défaut étant le répertoire courant. Exemple :

```
!PATH = '/usr/project/myprograms:' + !PATH
```

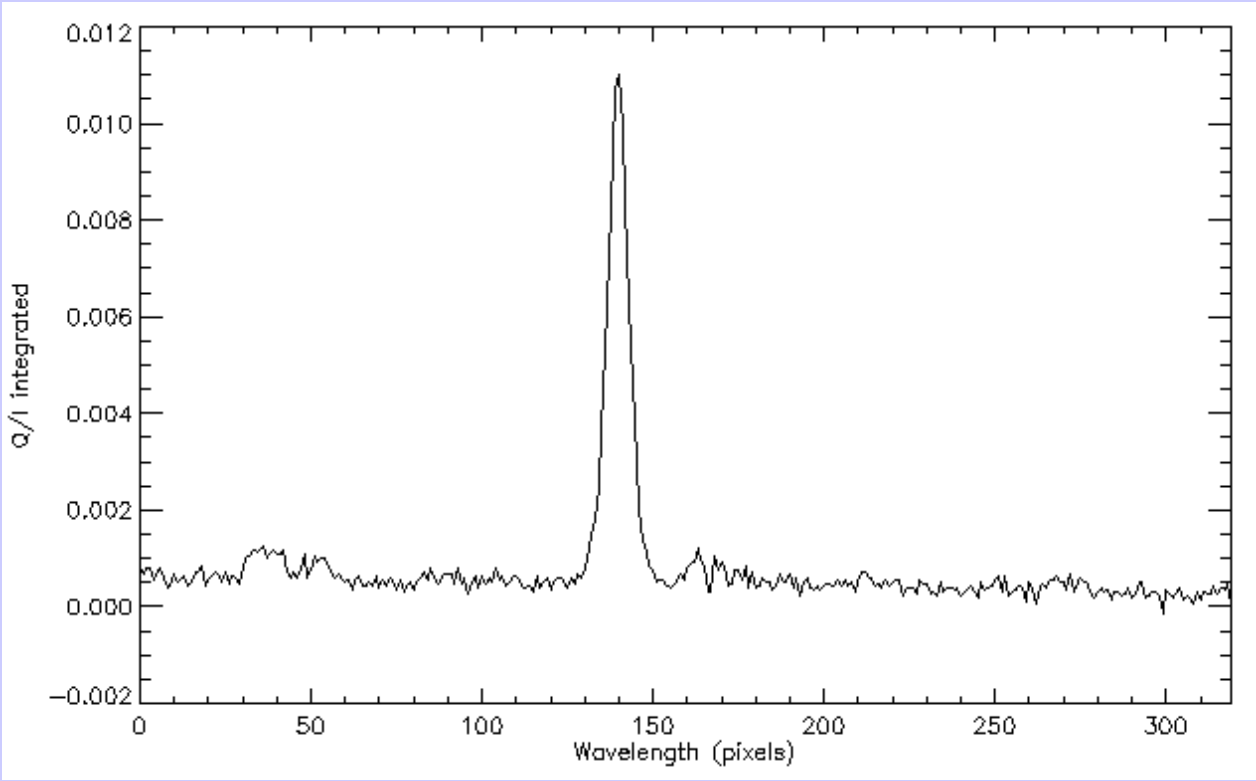
Cette instruction ajoute au chemin de recherche standard l'exploration du répertoire utilisateur : /usr/project/myprograms

```
Pro spectropol
;Auteur du programme :
;Date de création :
;Date de révision :
;Fonction réalisée par le programme :
;-----------------------------------------------
;répertoire de lecture
chemin='/usr/local/Master_TPinfo/M1_TPinfo/OSS2008-M1/'
;lecture des deux fichiers d'entrée
t1=readfits(chemin+'f1.fts')
t2=readfits(chemin+'f2.fts')
;conversion en flottant 32 bits
t1=uint(t1)
t1=float(t1)
t2=uint(t2)
t2=float(t2)
;calcul de l'image Q/I(lambda,x)
i=t1+t2
q=t1-t2
qsuri=q/i
;taille de l'image Q/I(lambda,x)
dim=size(qsuri)
dimx=dim(1)
dimy=dim(2)
;couper 100 lignes en haut et en bas
qsuri=qsuri(*,100:dimy-100)
dimy=dimy-200
;visualisation de l'image Q/I(lambda,x)
window,0,xs=dimx,ys=dimy
tvscl,qsuri
;intégration dans le sens spatial de Q/I(lambda,x)
qi=total(qsuri,2)
;moyenne
qi=qi/float(dimy)
;visualisation graphique de l'intégration Q/I(x)
window,1
plot,qi,xtitle='Wavelength (pixels)',ytitle='Q/I integrated',xstyle=1
end
```


f1.fts
f2.fts
x
λ
Zone de
calcul



Taux de polarisation linéaire Q/I



Intégration le long de x