Table des matières

1 Préambule

IDL ne fait pas de distinction entre les lettres majuscules et minuscules. En conséquence, si on fait

```
1 q = 2
2 Q = 3
3 print,q
```

On obtiendra la valeur 3. Il faut donc faire extrêmement attention.

2 Les bases

2.1 Les boucles

2.1.1 case

```
1   case nplots of
2      1: begin
3         !p.multi=0
4      end
5
6      2: begin
7         !p.multi=0
8      end
9
10     3: begin
11        !p.multi=[0,1,2]
12     end
13  endcase
```

2.1.2 if

```
1 if [boolean] then begin
2   [code]
3 endif
```

```
1 if [boolean] then begin
2   [code]
3 endif else begin
4   [code]
5 endelse
```

Pour de multiples valeurs d'une même variable, préférer une boucle `case`.

2.1.3 for

```
1 for i=0,n_elements(a)-1 do begin
2   [code]
3 endfor
```

2.1.4 where

```
a=indgen(10)
ind=where(a lt 5,count)
print,a[ind]
```

grâce à la commande **where**, on stocke dans `ind` les indices des valeurs de a qui vérifient $a < 5$. Ça permet de sélectionner rapidement et efficacement certaines données pour les isoler.

`count`, quant à lui, est une variable qui va stocker le nombre d'éléments vérifiant la condition que l'on a donné. En particulier, ça permet de ne faire des opérations que si `count` est différent de 0.

2.2 Les booléens

2.2.1 Comparateurs

Pour comparer deux entiers a et b on a un certain nombre d'opérateurs binaires listés dans [Table 1, p3]

Commande : Signification
`a eq b` : $a = b$
`a ne b` : $a \neq b$
`a lt b` : $a < b$
`a le b` : $a \leqslant b$
`a gt b` : $a > b$
`a ge b` : $a \geqslant b$

TABLE 1 – Liste des comparateurs mathématiques les plus usuels, notamment pour les tests.

Opérateur : Signification
`~` : non
`&&` : et
`||` : ou

TABLE 2 – Liste des operateurs sur booléens

2.2.2 Opérateurs booléens

Sur [TABLE 2, p3] sont représentés les opérateurs sur booléens disponibles. On les utilise de la façon suivante :

```
1 if (~(a lt c) && (a eq b)) then begin
2       [code]
3   endif
```

signifie qu'on ne veut pas que a soit inférieur strictement à c et on veut que a soit égal à b.

2.3 fonctions mathématiques standard

nom : fonction
alog : logarithme népérien (ou naturel)
alog10 : logarithme décimal
exp : exponentiel
factorial : factoriel (n !)
abs : valeur absolue
sqrt : racine carrée (pour l'anglais square root)
cos : cosinus (attention, tous les angles sont en radians : p radians = 180 °)
sin : sinus
tan : tangente (rappelons que cotangente = 1/tangente)
acos : arc cosinus (fonction inverse de cosinus)
asin : arc sinus (fonction inverse de sinus)
atan : arc tangente (fonction inverse de tangente)
sinh : sinus hyperbolique (sinh(x) = [exp(x)-exp(-x)]/2)
cosh : cosinus hyperbolique cosh(x) = [exp(x)+exp(-x)]/2)
tanh : tangente hyperbolique (tanh(x)=sinh(x)/cosh(x))

2.4 Les tableaux

Il existe plusieurs types de tableaux, celui que j'utilise le plus est :

```
a=dblarr(12,3)
b=dblarr(10)
```

qui définit a comme un tableau de 12 colonnes et 3 lignes et b comme une liste de 10 éléments, les éléments de ces deux tableaux étant des réels double précision.

 La numérotation commence à 0, ainsi, `b[0]` est le premier élément de b, et `b[9]` est le dernier élément

Il existe un moyen très pratique de générer des listes, notamment quand on veut tracer des courbes, c'est à l'aide de la fonction suivante :

```
f=indgen(10)
dx=1.d-2
x=dx*indgen(10)
```

Ceci définit f comme une liste contenant les entiers de 0 à 9, et x comme la même chose, sauf qu'on a définit un pas, dx qui nous permet de resserrer les points. On peut ainsi rajouter une valeur minimale, faire une échelle logarithmique et plein de choses très pratique.

Remarque : À noter que si on exécute :

```
a=0.1*!dpi*indgen(10)
b=sin(a)
```

alors on va faire le sinus de chaque élement de a et le stocker dans b qui sera un tableau de même taille que a. `!dpi` est la variable qui contient la valeur de π en réel double précision.

Il existe des commandes spéciales pour manipuler des tableaux, qu'il ne faut surtout pas reprogrammer soit même, vu que le temps de calcul serait plus long :

```
total(a)
max(a)
min(a)
n_elements(a)
size(a)
```

qui définissent respectivement la somme, le maximum, le minimum, le nombre d'éléments et les dimensions (je ne connais pas le détail de cette dernière commande, vu que je ne l'utilise jamais). À noter que ce sont des fonctions, donc on les définit de la façon suivante :

```
somme=total(a)
```

Remarque : On peut aussi récupérer l'indice de l'élément le plus grand ou le plus petit du tableau. Pour celà, il suffit d'écrire :

```
energie_max=max(energie,indice_max)
energie_min=min(energie,indice_min)
print,energie[indice_max],energie[indice_min]
```

Dans le cas présent, ce n'est pas très utile, mais ça permet, dans d'autres cas, quand on a plusieurs tableaux, de récupérer cet indice, et d'afficher les autres caractéristiques correspondant à ce même indice, c'est à dire rayon, masse et vitesse d'un objet dont l'énergie est maximale, par exemple.

La commande sort(a) permet de sortir une liste d'indice qui permet de trier la liste dans l'ordre croissant. Ainsi

```
print,a[sort(a)]
```

affiche la liste a avec les éléments triées.

2.5 Les variables aléatoires

Il existe au moins deux types de variables aléatoires implémentées dans GDL (i) distribution uniforme (ii) distribution normale que l'on appelle respectivement par :

```
a=randomu(seed,10)
b=randomn(seed,10)
```

où 10 est le nombre de valeurs que l'on veut, et `seed` le nombre qui sert à générer la séquence aléatoire. On peut tout à fait (et c'est conseillé si on veut des résultats reproductible pour vérification) spécifier la valeur de seed, du style :

```
a=randomu(182751824562,10)
```

Remarque : Pour un `seed` donné, la séquence de nombre aléatoire générée est toujours la même.

Pour éviter les correlations entre les générations de nombres aléatoires, il faut en fait utiliser le même seed quand on fait plusieurs tirages, et les nombres générés sont à la suite (quand les générations se font au sein du même programme, sans réinitialisation du seed.

Exemple :

```
seed=1001L
print,randomu(seed,10)
print,randomu(seed,10)
```

les deux suites de nombres seront différentes.

```
seed=1001L
print,randomu(seed,10)
seed=1001L
print,randomu(seed,10)
```

les deux suites de nombres seront les mêmes.

3 Les plots

3.1 Rendu correct des textes

Dans les plots, pour avoir un rendu agréable et propre des polices, il faut mettre l'option

```
1 !p.font = 0
```

3.2 Multi plots

Pour celà il faut définir une grille (ligne et colonne) afin de déterminer le nombre de plots que l'on pourra faire dans la même fenêtre.

```
1 !p.multi = [0,2,3]
```

Avec la ligne précédente, on définit 3 lignes et 2 colonnes pour une fenêtre graphique[1]. Ensuite, chaque commande `plot` qui suivra occupera une cellule de cette grille, la grille étant remplir à partir du coin en haut à gauche, ligne par ligne.

3.3 Forcer le recadrage d'une figure

Avec les options `xrange=[`*xmin*`,`*xmax*`]` et `yrange=[`*ymin*`,`*ymax*`]` on peut recadrer la figure, mais IDL n'en fait parfois qu'à sa tête et ne limite pas exactement le graphique aux bornes qu'on lui soumet. Afin de le forcer à obéir, il suffit de rajouter `/xstyle` et/ou `/ystyle` pour respectivement forcer les bornes horizontales ou verticales qu'on lui entre via `xrange` et `yrange`.

3.4 Enlever les axes

Il faut utiliser les mots clés `xstyle` et `ystyle`. Par exemple :

```
1 plot,x,y,xstyle=4,ystyle=4
```

C'est quelque chose que j'utilise pour faire des contours de disque de gaz et ne pas avoir les axes afin d'avoir une image plus présentable.

Les options sont :

Remarque : Ainsi, pour supprimer les axes, et afficher les données en forçant le zoom au range fixé, il faut utiliser :

```
1 plot,x,y,xstyle=5,ystyle=5
```

1. Je ne sais pas à quoi correspond le 0, mais je le laisse

Valeur	Description
1	force l'étendue de l'axe à être exacte
2	étendre le *range* de l'axe
4	Supprimer l'axe
8	Supprimer le style en boite (l'axe ne sera affiché que d'un seul coté du graphique)
16	empêcher la valeur minimale 0 pour l'axe Y (uniquement pour l'axe Y)

TABLE 3 – Valeurs possibles pour les mots clés `[XYZ]STYLE`. À noter que l'on peut superposer plusieurs options. Par exemple si on veut que l'étendue de l'axe soit exacte ET supprimer le style en boite, on peut mettre le mot clé à $8 + 1 = 9$

3.5 Exporter en .png

Il faut pour cela spécifier la taille de l'image. L'autre problème peut venir des couleurs. Quand je veux des .png avec des couleurs assez différentes (pour afficher les orbites de plusieurs planètes par exemple), je fais :

```
filename = "plot.png"
image_width = 640
image_height = 640

loadct,39,/silent

set_plot,'z'
device,set_resolution=[image_width,image_height],set_pixel_depth=24,decomposed=0
!p.background = 255
!p.color = 0

plot,x,y

write_png,filename,tvrd(/true)
device,/close
```

C'est utile quand on veut ensuite utiliser les .png pour faire une vidéo, avec **avidemux** par exemple.

3.6 Exporter en postscript

Le plus simple à faire, mais quelques adaptations me semblent quand même indispensables. La première est d'utiliser l'astuce pour avoir un rendu correct des textes [§ 3.1, p5]. La deuxième est d'exporter en .eps et non pas en .ps. L'eps est un postscript encapsulé, c'est à dire, grossièrement (de ce que j'en comprends), qu'il stocke aussi l'information sur la taille de la figure et le cadre qu'il faut utiliser pour avoir la figure, et rien que la figure. En postscript simple, il arrivera souvent qu'on ait une figure de taille A4 avec d'énormes blancs sur les cotés.

La 3^{e} et dernière astuce n'est pas indispensable mais je la trouve pratique. Elle consiste à rajouter systématiquement à mes scripts une option `/write` qui permet d'écrire la figure dans un fichier ou non. L'affichage graphique à l'écran se faisant dans tous les cas.

J'ai alors quelque chose du style :

```
pro test,write=write
  filename = "plot.eps"

  if keyword_set(write) then begin
    isps = 1
  endif else begin
    isps = 0
  endelse

  for j=0,isps do begin
    if (keyword_set(write) and (j eq 0)) then begin
      set_plot,'PS'
      device,/encaps,xsise=20,ysize=20*5./6.,$
          /color,bits=24,file=filename
    endif

    plot,1,2
```

```
18
19      if (keyword_set(write) and (j eq 0)) then begin
20        device,/close
21        set_plot,'X'
22      endif
23    endfor
24  end
```

4 Manipuler des fichiers

4.1 Tester l'existence d'un fichier

```
1 if file_test("toto.txt") then begin
2   print,"'toto.txt' existe"
3 endif
```

5 Astuces

5.1 Commandes bash

Pour appeler une commande du shell, du style pour lister les fichiers, il faut utiliser un $ devant la commande. Ainsi, pour lister les fichiers, on utilisera la commande :

```
GDL> $ls
back      cooling.pro~  energy.pro~ lhb.pro~    tutos     warm.pro~
cooling.pro  energy.pro    lhb.pro LHB_temp_ocean.ps  warm.pro
```

5.2 Les mots clés et comment les tester

```
1 pro ma_fonction,option=option
2
3 if keyword_set(option) then begin
4   print,"'/option' est active"
5 endif
6
7 end
```

5.3 Lecture de fichiers de paramètres

Il est pratique, pour lire un fichier de paramètre, de créer une fonction qui s'occupe de ça, afin de pouvoir facilement réutiliser et modifier le code. Il est encore plus pratique que cette fonction retourne non pas une liste de valeurs, mais une structure. Pour ceux qui sont familier avec la programmation objet, ça correspond plus ou moins à une instance de classe, l'appel des valeurs ce faisant par `nom_variable.valeur`.

Un exemple concret. Voici le code de ma fonction permettant de récupérer les valeurs du fichier de paramètres `genesis.out` :

```
1  function getGenesisOut
2      nProcAzim=0
3      nProcRad=0
4      radialRes_p=long(0)
5      azimRes_p=long(0)
6      npl=0
7      nstar=0
8      rmin=0.
9      rmax=0.
10     nOutput=0
11
12     openr,lun,"genesis.out",/get_lun
13     readf,lun, nProcAzim
14     readf,lun, nProcRad
15     readf,lun, radialRes_p
16     readf,lun, azimRes_p
17     readf,lun, npl
18     readf,lun, nstar
19     readf,lun, rmin, rmax
20     readf,lun, nOutput
21     free_lun,lun
22
23     radialRes = nProcRad * radialRes_p
24     azimRes = nProcAzim * azimRes_p
25
26     parameters = {nrad:uint(radialRes), nazim:uint(azimRes), npl:uint(npl),
27     nstar:uint(nstar), rmin:rmin, rmax:rmax, nout:nOutput}
28
```

```
29      return, parameters
30  end
```

Ensuite, pour utiliser et récupérer les informations, il suffit de faire :

```
1 genesis = getGenesisOut()
2 print,genesis.rmin,genesis.rmax
3 print,genesis.npl
```

L'avantage de cette méthode est qu'on peut modifier le fichier de paramètres, modifier la fonction, on pourra toujours appeler les anciens paramètres de la même manière, sans avoir à modifier les scripts.

5.4 Variables globales

Au lieu de s'embêter, dans une fonction ou une procédure, à rajouter des paramètres à n'en plus finir. On peut, comme en fortran, définir un *common block*, c'est à dire un ensemble de variables qui seront définies dans ce block et auxquelles on pourra accéder dans n'importe quelle fonction ou procédure à condition d'y définir, le même block.

Exemple : Je définis un block nommé *parameters* dans lequel j'inclue les variables `rmin` et `rmax` :

```
1 common parameters, rmin, rmax
```

Pour accéder à ces variables dans n'importe quelles fonctions ou procédures, il me suffit de rajouter la déclaration du block :

```
1  pro testation
2    common parameters, rmin, rmax
3    x = 3.
4    rmin = 0.6
5    rmax = 1.8
6
7    printvar(x)
8  end
9
10 function printvar,x
11   common parameters
12
13   print,x,rmin,rmax
14 end
```

6 FAQ

6.1 Impossible de lire un fichier

Il refusait catégoriquement de lire mon fichier, quoi que je fasse. Après plusieurs heures de recherches éfrenées, il s'est avéré que ça venait du fait que la fin de ligne de mon fichier était une fin de ligne mac. Après avoir changé et mis une fin de ligne Unix/windows, tout est rentré dans l'ordre.

6.2 Min ou Max changé sans crier gare

Les fonctions `min()` et `max()` sont des fonctions qui s'appliquent sur des listes et non sur des éléments.

J'avais un maximum *xmax* qui était changé sans que je comprenne pourquoi. En cours de script, j'avais la ligne

```
1 prefactor = max(ymax,xmax)
```

C'est cette ligne qui posait problème et même si je ne comprends pas le détail, j'ai résolu le soucis en remplaçant cette ligne par :

```
1 prefactor = max([ymax,xmax])
```