Manuel utilisateur de TRIP

version 1.4.120
24 October 2021

Gastineau - Laskar

TRIP est un programme de manipulation algébrique orienté vers les calculs de méthodes de perturbations et plus particulièrement vers la mécanique celeste.

Auteurs :

J. Laskar
Astronomie et Systèmes Dynamiques
Institut de Mécanique Céleste
77 avenue Denfert-Rochereau
75014 PARIS
email : laskar@imcce.fr

M. Gastineau
Astronomie et Systèmes Dynamiques
Institut de Mécanique Céleste
77 avenue Denfert-Rochereau
75014 PARIS
email : gastineau@imcce.fr

Avec des contributions de E. Paviot, F. Thire, D. Acheroff, M. To, A. Ceyrac, G. Rouault, V. Kelsch, F. Darricau, N. Brucy, F. Boschet.

Ce logiciel inclut une librairie développée par la fondation NetBSD et ses contributeurs.

Ce logiciel inclut les librairies LAPACK et 'SCSCP C Library'.

Ce logiciel est lié dynamiquement aux librairies LTDL, GMP, MPFR et MPC. Le code source de ces librairies peut être téléchargé depuis les sites internet indiqués dans le chapitre References (voir Chapitre 19 [References], page 211).

TRIP version 1.4.120

1 Introduction

Pour exécuter TRIP, il faut

- Sous Unix et MacOs X, taper trip dans un shell.
- Sous Windows, cliquer sur l'icone de trip pour la version console.
- Sous Windows, cliquer sur l'icone de tripstudio pour la version graphique.

Vous verrez à l'écran :

```
              **** BIENVENUE SUR TRIP v1.0.0 ****

Taper 'help;' ou '?;' pour obtenir de l'aide

>
```

TRIP est prêt à recevoir vos ordres. TRIP dispose d'un interpréteur sensible aux minuscules et majuscules.

Pour calculer $S = (X + Y)^2$, il suffit de faire :

```
              **** BIENVENUE SUR TRIP v1.0.0 ****

Taper 'help;' ou '?;' pour obtenir de l'aide

> S=(X+Y)^2;
S(X,Y) = 1*Y**2 + 2*Y*X + 1*X**2
```

Les touches de curseur permettent de modifier la ligne courante ou de rappeler les commandes précédentes.

Pour quitter TRIP, il suffit de taper `quit` ou `exit`.

Les mots réservés du langage TRIP sont spécifiés dans les annexes. Ces mots se répartissent en trois catégories :

- Variables globales, constantes de TRIP et symboles mathématiques.
- Fonctions : commande retournant une valeur
- Procédures : commande ne retournant jamais de valeur.

Au démarrage de TRIP ou lors de l'exécution des commandes reset ou @@ , celui-ci lit les fichiers suivants dans l'ordre indiqué :

'$TRIPDIR/etc/trip.ini'	où $TRIPDIR est le répertoire où est installé TRIP.
'$HOME/.trip'	où $HOME est le répertoire racine de l'utilisateur.
'trip.ini'	dans le répertoire courant.

Ces fichiers peuvent contenir n'importe quelles instructions.

Recommendations :

- Il est souhaitable de configurer les variables d'environnements suivantes :
  - TMPDIR : répertoire où seront stockés les fichiers temporaires.
  - LC_MESSAGES : langue utilisée pour les messages affichés.

2 Semantique

2.1 expression

Une expression peut faire intervenir tous les opérateurs définis sur les identificateurs, ainsi que des appels de fonctions ou de commandes. Une expression doit se terminer par le caractère `;` ou `$` . Le `;` entraine le calcul et l'impression du resultat, alors que le caractère `$` n'effectue que le calcul. Plusieurs instructions séparées par `;` ou `$` peuvent se trouver à la suite. Elle seront effectuées en séquence.

Un commentaire sur une ou plusieurs lignes est défini par `/* .... */`, comme dans le langage C. Un commentaire de fin de lignes est défini par les caractères `//` , comme dans le langage C++.

```
Exemple :
> // calcule 1+t et affiche le resultat
> s=1+t;
s(t) =
                          1
 +                        1*t

> // calcule 1+t mais n'affiche pas le resultat
> s1=(1+x+y)^2$
> // affiche le contenu de s1
> s1;
s1(x,y) =
                          1
 +                        2*y
 +                        1*y**2
 +                        2*x
 +                        2*x*y
 +                        1*x**2

>
```

2.2 identificateur

Un identificateur doit commencer par une lettre et se composer de lettres (majuscules ou minuscules) et/ou de chiffres (09), ainsi que du caractere _ ou ' . Par défaut, un identificateur est global, c'est-à-dire qu'il est visible depuis n'importe quelle instruction. Un identificateur peut être local à une macro, c'est-à-dire qu'il est visible uniquement depuis cette macro et sera détruit à chaque fois que l'exécution de cette macro se terminera. Un mot réservé de TRIP ne peut pas être utilisé comme identificateur. Un identificateur peut être de type :

- variable
- série
- constante
- tableau de séries
- tableau de variables
- vecteur numérique
- matrice numérique
- chaine de caractères
- structure

```
Exemple :
> ch="file1"$
> s=1+x$
> dim t[1:2];
> z=1+2*i;
z = (1+i*2)
> bilan;
ch  CHAINE
s   SERIE
t   TAB
x   VAR
```

La description des fonctions, procédures et variables utilisent les types suivants :

<identificateur>	identificateur
<entier>	nombre entier naturel ou une opération retournant un entier
<réel>	nombre réel ou une opération retournant un réel
<complexe>	nombre complexe ou une opération retournant un complexe
<variable>	variable
<constante>	une opération retournant un nombre entier, réel ou complexe
<série>	série
<opération>	une opération retournant une série ou un nombre (une constante)
<tableau>	tableau de série ou de constantes
<tableau de variables>	tableau de variables
<(tableau de) variables>	variable ou tableau de variables
<vec. num.>	vecteur numérique de réels ou de complexes
<vec. réel>	vecteur numérique de réels
<vec. complexe>	vecteur numérique de complexes
<tableau de vec. réel>	tableau de vecteurs numériques de réels
<tableau de vec. complexe>	tableau de vecteurs numériques de complexes
<tableau de vec. num.>	tableau de vecteurs numériques
<(tableau de) vec. réel>	(tableau de) vecteur numérique de réels
<(tableau de) vec. num.>	(tableau de) vecteur numérique
<constante ou vec. num.>	constante ou vecteur numérique de réels ou de complexes
<constante ou matrice>	constante ou matrice numérique de réels ou de complexes
<réel ou vec. réel>	nombre réel ou vecteur numérique de réels
<matrice>	matrice numérique
<matrice reelle>	matrice numérique de réels
<matrice complexe>	matrice numérique de complexes
<nom fichier>	nom de fichier
<fichier>	fichier
<macro>	macro
<dimension d'un tableau>	deux entiers séparés par un :
<liste_dimension>	liste de <dimension d'un tableau> séparée par une virgule
<2 dimensions d'une matrice>	liste de 2 dimensions séparée par une virgule. Chaque dimension est composée deux entiers séparés par un :
<condition>	comparaison entre deux opérations
<chaine>	une opération retournant une chaine de caractères

<(tableau de) chaine>	une opération retournant un tableau de chaine de caractères ou une seule chaine de caractères
<liste de variables>	liste de noms de variables ou tableau de variables
<liste_parametres>	liste de paramètres d'une macro
<corps>	liste d'expression trip
<client scscp>	connexion vers un serveur SCSCP
<objet distant>	objet stocké sur un serveur SCSCP distant
<session Maple>	connexion vers une session Maple locale
<session Mathematica>	connexion vers une session Maple locale

2.2.1 serie

Une série est un polynôme de degré n (n étant un entier). Elle est donc fonction d'une ou plusieurs variables.

```
Exemple :
> s=1+x;
s(x) = 1 + 1*x
> s2=(1+x+y);
s2(x,y) = 1 + 1*y + 1*x
> bilan;
s   SERIE
s2  SERIE
x   VAR
y   VAR
```

2.2.2 constante

Une constante est un nombre entier, réel, complexe, rationnel ou un intervalle . En mode numérique rationnel (`_modenum = NUMRAT` ou `_modenum = NUMRATMP`), la saisie de 0.5 crée un réel et non le rationnel 1/2.

```
Exemple :
>x = 2;
2
>y = 2.25;
9/4
>z = 1+2*i;
(1+i*2)
>bilan;
x     CONST
y     CONST
z     CONST
> _modenum=NUMDBLINT;
        _modenum   =    NUMDBLINT
> s = <1|2>;
 s = [+1.00000000000000E+00,+2.00000000000000E+00]
```

2.2.3 chaine de caracteres

C'est une suite de caractères entre " ". Elle est utilisée pour définir le `_path`, donner des commentaires, définir des noms de fichiers, etc... Elles ne sont pas limitées en taille. Par contre, pour le `_path`, sa longueur est tronquée à 256 (valeur dépendant du système). Pour produire un guillemet (") dans une chaine, il suffit de le doubler.

```
Exemple :
> ch = "file" + "1";
```

```
ch = "file1"
> sch = "../" + ch;
sch = "../file1"
> sch1 = sch + "." + str(12);
sch1 = "../file1.12"
> ch3="nomsuivide""fin";
ch3 = "nomsuivide"fin"
```

2.2.4 reel

<nom> = <réel> ;

Un nombre réel est toujours affiché ou saisi en base 10. Le symbole `d`, `e`, `E` ou `D` est l'opérateur d'exponentiation.

```
Exemple :
> a=2.125E6;
a = 2125000
> q=3D2;
q = 300
> r=0.123554545;
r = 0.123554545
```

2.2.5 complexe

<nom> = <réel> + i * <réel> ;

Les nombres complexes sont reconnus. Il suffit de les écrire avec un i minuscule ou avec un I majuscule.

```
Exemple :
>  x=2+i*3;
x = (2+i*3)
> y=-5+4*i;
y = (-5+i*4)
```

2.2.6 nom de fichier

C'est une chaine de caractères . Cette chaine doit etre encadrée par des " " si la chaine contient les caractères espaces " [ ] { } ( ) ou plusieurs points. Un nom de fichier peut être le contenu d'un identificateur de type chaine de caractères.

Ces noms de fichiers sont toujours relatifs à la variable globale *_path*, excepté ceux construits à l'aide de la fonction `file_fullname` (voir Section 18.20 [file_fullname], page 208).

```
Exemple :
> read("fichier.1.dat",T);
> s="ell."+str(10)+".asc";
s = "ell.10.asc"
> read(s,T);
```

2.2.7 fichier

C'est un objet désignant un fichier ouvert en lecture ou en écriture.

```
Exemple :
> f = file_open("fichier.1.dat",read);
> file_close(f);
```

2.3 Visibilité

2.3.1 private

`private` [Commande]

`private <identificateur> x ,...;`

Cette commande indique que les identificateurs suivants seront locaux à une macro, à une boucle (for, while, sum) ou à un fichier.

Les identificateurs locaux sont détruits à la fin de l'exécution d'une macro ou à la fin de chaque itération de la boucle (for, while, sum).

Si l'identificateur est local à un fichier, celui-ci est uniquement visible par les macros de ce fichier ou pendant l'exécution de ce fichier.

Si un identificateur local porte le même nom qu'un identificateur global, l'identificateur local sera alors utilisé automatiquement. Dans ce cas, on dit que l'identificateur local cache l'identificateur global.

`private _ALL` [Commande]

`private _ALL;`

Cette commande indique que tous les identificateurs utilisés dans la macro ou dans le fichier seront locaux à cette macro ou à ce fichier.

Le comportement est identique à la déclaration précédente.

Pour accéder à un objet global, il faut utiliser la commande `public` (voir Section 2.3.2 [public], page 8).

```
Exemple :
> // S, SY, SXY, SXX, DEL sont des identificateurs locaux
> macro least_ab[TX,TY,a,b] {
 private S, SY, SXY, SXX, DEL;
 S=size(TX)$
 SX=sum(TX)$
 SY=sum(TY)$
 SXY=sum(TX*TY)$
 SXX=sum(TX*TX)$
 DEL=S*SXX-SX*SX$
 a = (S*SXY-SX*SY)/DEL$
 b = (SXX*SY-SX*SXY)/DEL$
};
>
> tx=1,10;
tx  Vecteur de reels double-precision : nb reels =10
> ty=3*tx;
ty  Vecteur de reels double-precision : nb reels =10
> %least_ab[tx,ty,[a],[b]];
> bilan;
SX CONST
a CONST
b CONST
tx VNUMR
ty VNUMR
>
> // T2 est un identificateur local
```

```
> macro detval[T] {
 private _ALL;
 dim T2[1:3,1:3]$
 T2[:,::]=0$
 T2[1,1]=T**2$
 T2[2,2]=1-T$
 T2[3,3]=T$
 return det(T2);
};
> %detval[5];
                        -500
> bilan;
SX CONST
a CONST
b CONST
tx VNUMR
ty VNUMR
>
```

2.3.2 public

`public` [Commande]

`public` <identificateur> `x ,...;`

Cette commande ne doit être utilisée qu'après la commande `private _ALL;`. Elle indique que les identificateurs de cette liste seront globaux alors que tous les autres seront locaux à cette macro ou à ce fichier.

Exemple :

```
> // w est local et z est global
> macro myfunc[T] {
  private _ALL;
  public z;
  w = cos(T);
  z = T;
};
> %myfunc[5];
w =         0.2836621854632262
z =                          5
> bilan;
z CONST
>
```

3 Variables globales

L'affichage de la valeur d'une variable globale est realisée en donnant le nom de la variable suivi de ; . L'affichage de la valeur de l'ensemble des variables globales s'effectue avec la commande `vartrip` (voir Section 18.12 [vartrip], page 203). Les variables globales réelles sont stockées sous la forme d'un réel double-précision.

3.1 _affc

`entier _affc` [Variable]

`_affc` = {1, 2, 3, 4, 5, 6, 7, 8, 9};

Elle indique le format souhaité pour l'affichage des coefficients numériques dans les séries ou constantes.

Valeur par défaut = 4.

= 1 : %G (format court standard)

= 2 : %.8G (8 chiffres après la virgule)

= 3 : %.10G (10 chiffres après la virgule)

= 4 : %20.15G (15 chiffres ou plus après la virgule)

= 5 : pseudo rationnel (les doubles sont convertis en rationnels par fractions continues quand c'est possible, sinon %.8G)

= 6 : même chose que 5 mais %.15G (15 chiffres après la virgule)

= 7 : pseudo rationnel (format de longueur fixe)

= 8 : %15.6E (6 chiffres après la virgule)

= 9 : même chose que 4 avec diamètre de l'intervalle

```
Exemple :
> _affc=1;
    _affc = 1
> 1/3;
    0.333333
> _affc=2;
    _affc = 2
> 1/3;
    0.33333333
> _affc=3;
    _affc = 3
> 1/3;
    0.3333333333
> _affc=4;
    _affc = 4
> 1/3;
       0.333333333333333
> _affc=5;
    _affc = 5
> 1/3;
    1/3
> _affc=6;
    _affc = 6
> 1/3;
    1/3
```

```
> _affc=7;
    _affc = 7
> 1/3;
         1/3
> _affc=8;
    _affc = 8
> 1/3;
       3.333333E-01
> _affc=9; _modenum=NUMDBLINT;
    _affc = 9
    _modenum   =    NUMDBLINT
> 1/3;
    [+3.333333333333333E-1,+3.333333333333334E-1 (   5.551115E-17)]
```

3.2 _affdist

`entier _affdist` [Variable]

`_affdist` = {0, 1, 2, 3, 4, 5, 6, 7, 8};

indique le type d'affichage souhaité pour les séries

Valeur par défaut = 2.

- = 0 : récursif.
- = 1 : distribué.
- = 2 : distribué avec un terme par ligne.
- = 3 : distribué sous forme de (co)sinus.
- = 4 : distribué sous forme de (co)sinus avec un terme par ligne.
- = 5 : distribué et aligné sous forme de (co)sinus avec un terme par ligne.
- = 6 : distribué et factorisé par rapport aux variables de `_affvar`.
- = 7 : distribué et factorisé par rapport aux variables de `_affvar`, avec un terme par ligne pour les 2 parties.
- = 8 : distribué et factorisé par rapport aux variables de `_affvar`, avec un terme par ligne pour la partie distribuée et sur une même ligne pour la partie factorisée .

Remarque : Si `_affvar` est vide, l'affichage de `_affdist` = 6, respectivement 7 ou 8, est équivalent à `_affdist`=1, respectivement 2.

```
Exemple :
>_affdist = 2;
>x = 1+y;
x(y) =
1
+ 1*y
```

3.3 _comment

`booleen _comment` [Variable]

`_comment` {`on`,`off`};

Active ou désactive l'affichage des commentaires.

Valeur par défaut = `off`.

```
Exemple :
> _comment;
_comment OFF
> /* série à deriver : /* s=1+x */  : */;
> _comment on;
> /* série à deriver : /* s=1+x */  : */;
série à deriver :  s=1+x   :
> _comment off;
```

3.4 _cpu

`entier _cpu` [Variable]

`_cpu` = <entier> ;

Spécifie le nombre de processeurs utilisables pour les calculs suivants.

Il est impossible de spécifier un nombre supérieur au nombre de processeurs ou coeurs utilisables.

Valeur par défaut : nombre de processeurs ou coeurs disponibles sur la machine.

```
Exemple :
> _mode=POLP;
_mode      =    POLP
> _cpu=1;
_cpu = 1
> time_s; s=(1+x+y+z+t+u)^30$ time_t(usertime, realtime);
03.321s  -  ( 99.44% CPU)
> msg("the real time is %g\n", realtime);
the real time is 3.33976
> _cpu=4;
_cpu = 4
> time_s; s=(1+x+y+z+t+u)^30$ time_t(usertime, realtime);
03.339s  -  (322.89% CPU)
> msg("the real time is %g\n", realtime);
the real time is 1.03414
```

3.5 _echo

`booleen _echo` [Variable]

`_echo` = {0, 1};

`_echo` {on,off};

Active ou désactive l'affichage de l'écho des commandes exécutées.

Valeur par défaut = `off`.

= 1 : Affiche l'écho des commandes (similaire à on)

= 0 : N'affiche pas l'écho des commandes (similaire à off)

```
Exemple :
>_echo = 1;
>msg "exemple";
exemple
cde>>
msg "exemple";
```

3.6 _endian

`nom _endian` [Variable]

`_endian` = {little , big};

Indique l'ordre des octets (big-endian ou little-endian) dans les fichiers binaires.

Valeur par défaut = endianness de la machine.

= big : big-endian

= little : little-endian

```
Exemple :
> _endian=big;
                _endian    =    big
> vnumR ieps;
> readbin(file1.dat,"%u",ieps);
```

3.7 _graph

`nom _graph` [Variable]

`_graph` = `gnuplot`;

`_graph` = `grace`;

Indique si les commandes plot, replot, plotps, plotf utiliseront les logiciels grace ou gnuplot (voir Chapitre 12 [Graphiques], page 111) pour réaliser les graphiques.

Valeur par défaut = `gnuplot`.

= gnuplot : gnuplot réalise les graphiques

= grace : grace réalise les graphiques

```
Exemple :
> _graph = grace;
                _graph      = grace
> _graph = gnuplot;
                _graph      = gnuplot
```

3.8 _hist

`booleen _hist` [Variable]

`_hist` {on,off};

Indique si le fichier `history.trip` est ouvert ou non.

Si le fichier est ouvert, les commandes sont alors enregistrées dans ce fichier.

Valeur par défaut = `on`.

```
Exemple :
>_hist;
_hist = ON
```

3.9 _history

`chaine _history` [Variable]

`_history` = <chaine> ;

Indique le répertoire où sera stocké le fichier `history.trip`.

Valeur par défaut = `""`.

```
Exemple :
> _history="/users/toto/";
        _history = /users/toto/
```

3.10 _info

booleen _info [Variable]

_info {on,off};

Active ou désactive l'affichage de messages d'informations.

Valeur par défaut = on.

```
Exemple :
> _modenum=NUMQUAD;
_modenum   =    NUMQUAD
> x1=0,2*pi,2*pi/59$
> x2=0,3,0.5$
> _info off;
_info off
> sl=interpol(LINEAR,x1,cos(x1),x2)$
> _info on;
_info on
> sl=interpol(LINEAR,x1,cos(x1),x2)$
 Information : interpol effectue l'interpolation numerique en double-precision.
```

3.11 _integnum

booleen _integnum [Variable]

_integnum = {DOPRI8,ODEX1,ADAMS};

D'efinit l'intégrateur utilisée lors de l'intégration numérique par integnum (voir Section 17.4 [integnum], page 185) et integnumfcn (voir Section 17.5 [integnumfcn], page 190).

Valeur par défaut = DOPRI8.

= DOPRI8 : Runge-Kutta 8(7) ("A family of embedded Runge-Kutta formulae", Journal of Computational and Applied Mathematics, Dormand et Prince, 1980).

= ODEX1 : méthode d'extrapolation d'ordre 1 ("Solving ordinary differential equations I", Hairer et al., 1987).

= ADAMS : méthode Adams prédicteur-correcteur d'ordre 12 ("Solving ordinary differential equations I", Hairer et al., 1987).

```
Exemple :
> _integnum=ODEX1;
_integnum  =    ODEX1
> _integnum=DOPRI8;
_integnum  =    DOPRI8
```

3.12 _language

_language [Variable]

_language = {fr, en };

Change la langue utilisee pour les messages.

valeur par défaut =

– Sous Unix, Linux et MacOS X, depend de la variable d'environnement LC_MESSAGES. Si elle n'existe pas, la languge francaise est utilisee.

– Sous Windows, depend des préférences du système.

```
Exemple :
> _language = en ;
                _language    =    en
> Exp(x, y);
TRIP[ 3 ] : This parameter must be an integer
        Command :' Exp(x, y);'
                           ^
> _language = fr;
                _language    =    fr
> Exp(x, y);
TRIP[ 3 ] : Cet argument doit etre un entier
        Commande :'Exp(x, y);'
```

3.13 _mode

_mode [Variable]

_mode = {POLY, POLP, POLYV, POLPV };

Indique la représentation utilisée pour le stockage séries et pendant les calculs sur les séries.

Ce mode peut être changé en cours de route.

Valeur par défaut = POLY.

– POLY : Séries sous forme récursive creuse. Ce mode est le mode par défaut. Il est efficace pour les degrés très élevés.

– POLP : Séries sous forme récursive pleine. Souvent le plus rapide. A utiliser dès que les degrés partiels des séries sont peu élevés.

– POLYV : Séries sous forme récursive creuse. Ce mode est le mode par défaut. Les listes terminales sont optimisées pour le mode numérique courant. Il est efficace pour les degrés très élevés.

– POLPV : Séries sous forme récursive pleine. Souvent le plus rapide. Les vecteurs terminaux sont optimisés pour le mode numérique courant. A utiliser dès que les degrés partiels des séries sont peu élevés.

```
Exemple :
> _mode=POLY;
_mode      =    POLY
> s1=(1+x)^100$
> s2=1+x^100$
> bilan mem;
          Nom            Memoire (octets)         Nb de termes
------------------- -------------------- --------------------
                 s1                 4848                  101
                 s2                   96                    2
------------------- -------------------- --------------------
                                    4944                  103

Memoire physique utilisee = 7118848 octets
Memoire maximale utilisee = 2783817728 octets
> _mode=POLP;
_mode      =    POLP
> s1=(1+x)^100$
```

```
> s2=1+x^100$
> bilan mem;
          Nom            Memoire (octets)         Nb de termes
-------------------- -------------------- --------------------
                  s1                 3232                  101
                  s2                 3232                    2
-------------------- -------------------- --------------------
                                     6464                  103

Memoire physique utilisee = 7208960 octets
Memoire maximale utilisee = 2784210944 octets

> _mode=POLY$ time_s; s1=(1+x)^100$ time_t;
00.240s  -  ( 47.74% CPU)
> _mode=POLP$ time_s; s1=(1+x)^100$ time_t;
00.013s  -  ( 40.57% CPU)
```

3.14 _modenum

`_modenum` [Variable]

`_modenum = {NUMDBL, NUMRAT, NUMRATMP, NUMQUAD, NUMFPMP};`

Indique le type choisi pour les coefficients numériques, des vecteurs numériques et des matrices numériques.

Ce mode peut être changé en cours de route.

Valeur par défaut = `NUMDBL`.

Les coefficients sont codés sous les formes suivantes :

- NUMDBL : réel double-précision. (par défaut)
- NUMQUAD : réel quadruple-précision. Ce mode n'est pas supporté sous Windows.
- NUMRAT : rationnel si possible, sinon d'un réel double-précision.

  La valeur absolue du numérateur et du dénominateur est comprise entre 0 et 2**62 sur la plupart des machines. Si le numérateur ou le dénominateur devient trop grand, le rationnel est compris en réel double-précision.

  Les nombres réels sont toujours stockés en double-précision. Par exemple, le coefficient "2.0" sera stocké sous la forme réel double-précision alors que le coefficient "2" sera stocké sous la forme d'un rationnel.
- NUMRATMP : entier ou rationnel multi-précision, sinon d'un réel double-précision.

  La valeur absolue du numérateur et du dénominateur ne sont pas limités.

  Les nombres réels sont toujours stockés en double-précision. Par exemple, le coefficient "2.0" sera stocké sous la forme réel double-précision alors que le coefficient "2" sera stocké sous la forme d'un rationnel.

  Ces coefficients numériques utilisent la librairie GMP. (type mpz_t ou mpq_t). Sur les systèmes d'exploitation 64 bits, les entiers inférieurs à 2^63-1 en valeur absolue utilisent directement les instructions du processeur au lieu de la librairie GMP.
- NUMFPMP : réel multiprécision.

  Le nombre de chiffres significatifs est specifies par la variable globale `_modenum_prec` (voir Section 3.15 [_modenum_prec], page 16).

  Ces coefficients numériques utilisent la librairie MPFR.

Pour les modes NUMRAT, NUMRATMP, les vecteurs ou matrices numériques contiennent toujours des réels ou de complexes double-précision.

```
Exemple :
> _affc=4;
    _affc = 4
> _modenum=NUMDBL;
        _modenum   =    NUMDBL
> s=1/11;
s =   0.0909090909090909
> _modenum=NUMRAT;
        _modenum   =    NUMRAT
> s=1/11;
s =          1/11
> _modenum=NUMFPMP;
                _modenum   =    NUMFPMP
> pi;
        3.14159265358979323846264338327950288419716939937510582097494459 24
```

3.15 _modenum_prec

`entier _modenum_prec` [Variable]

`_modenum_prec = <entier> ;`

Variable utilisée si `_modenum = NUMFPMP` (voir Section 3.14 [_modenum], page 15).

Elle indique le nombre de chiffres significatifs des nombres réels multi-precision.

Valeur par défaut = 64.

```
Exemple :
> _modenum=NUMFPMP;
_modenum   =    NUMFPMP
> _modenum_prec= 20;
_modenum_prec = 20
> pi;
3.14159265358979323846
> _modenum_prec= 40;
_modenum_prec = 40
> pi;
3.1415926535897932384626433832795028841975
```

3.16 _naf_dtour

`réel _naf_dtour` [Variable]

`_naf_dtour = <réel> ;`

Variable utilisée par naf (voir Section 17.1 [Analyse en frequence], page 179).

Elle indique la "longueur d'un tour de cadran".

Valeur par défaut = 2*pi.

```
Exemple :
> _naf_dtour=360;
```

3.17 _naf_icplx

`entier _naf_icplx` [Variable]
`_naf_icplx` = {0, 1};

Variable utilisée par naf (voir Section 17.1 [Analyse en frequence], page 179).

Elle indique si la fonction est réelle ou complexe.

Valeur par défaut = 1.

Les valeurs possibles sont :

= 0 : fonction réelle

= 1 : fonction complexe

```
Exemple :
> _naf_icplx=0;
```

3.18 _naf_iprt

`entier _naf_iprt` [Variable]
`_naf_iprt` = {-1 , 0, 1, 2};

Variable utilisée par naf (voir Section 17.1 [Analyse en frequence], page 179).

Elle indique le niveau d'affichage pour `naf` et `naftab`. L'affichage des resultats intermédiaires sera stocké dans un fichier.

Valeur par défaut = -1.

Les valeurs possibles sont :

= -1 : aucun affichage.

= 0 : affichage très succint.

= 1 : affichage détaillé.

= 2 : affichage très détaillé.

```
Exemple :
> _naf_iprt  = 2;
```

3.19 _naf_isec

`entier _naf_isec` [Variable]
`_naf_isec` = {0, 1};

Variable utilisée par naf (voir Section 17.1 [Analyse en frequence], page 179).

Elle indique l'utilisation ou non de la méthode des sécantes.

Valeur par défaut = 1.

Les valeurs possibles sont :

= 0 : la méthode des sécantes n'est pas utilisée.

= 1 : la méthode des sécantes est utilisée.

```
Exemple :
> _naf_isec=1;
```

3.20 _naf_iw

`entier _naf_iw` [Variable]

`_naf_iw = <entier> ;`

Variable utilisée par naf (voir Section 17.1 [Analyse en frequence], page 179).

Elle indique la présence de fenêtre.

Valeur par défaut = 1.

Les valeurs possibles sont :

= -1 : fenêtre exponentielle $PHI(T) = 1/CE * \exp^{(-1/(1-T^2))}$
avec CE= 0.2219969080840397189 1E0

= 0 : pas de fenêtre

= N > 0 : $PHI(T) = C_N * (1 + \cos(\pi T))^N$ avec $C_N = 2^N * (N!)^2/(2N)!$

```
Exemple :
> _naf_iw=-1;
```

3.21 _naf_nulin

`entier _naf_nulin` [Variable]

`_naf_nulin = <entier> ;`

Variable utilisée par naf (voir Section 17.1.1 [naf], page 179).

Elle indique le nombre de lignes à ignorer en début du fichier de données.

Valeur par défaut = 1.

```
Exemple :
> _naf_nulin=0;
```

3.22 _naf_tol

`réel _naf_tol` [Variable]

`_naf_tol = <réel> ;`

Variable utilisée par naf (voir Section 17.1 [Analyse en frequence], page 179).

Elle indique la tolérance pour déterminer si deux fréquences sont identiques.

Valeur par défaut = 1E-10.

```
Exemple :
> _naf_tol=1E-4;
```

3.23 _path

`chaine _path` [Variable]

`_path = <chaine> ;`

`_path = "chemin du repertoire";`

Indique le répertoire utilisé pour la sauvegarde ou le chargement de fichiers. Toutes les commandes ou fonctions TRIP sauvegardant ou lisant des fichiers utilisent ce radical. Ce chemin est notamment utilisé pour le chargement de programme trip (include), la création de fichier postscript (plotps) et le tracé de fichier (plotf).

Valeur par défaut = "" (le répertoire courant).

Remarque :

- Ne pas oublier le "/" à la fin du path sous UNIX ou MACOS X.
- Ne pas oublier le "\" à la fin du path sous WINDOWS.

```
Exemple :
_path = "/u/gram/trip/"; /* UNIX */
_path = "\u\gram\trip\"; /* WINDOWS */
```

3.24 _quiet

`booleen _quiet` [Variable]

`_quiet` {on,off};

Supprime, respectivement réactive, les affichages ultérieurs pour les instructions suivies de `;`, uniquement dans la macro courante.

Valeur par défaut = `off`.

```
Exemple :
> macro silence{ s=1; _quiet on; s=pi; };
> %silence;
s =                          1
> stat(s);
 constante de nom s et de valeur    3.141592653589793
```

3.25 _read

`chaine _read` [Variable]

`_read` = ( );

`_read` = ("skipbrokenlines", `"skipemptylines"`);

`_read` est une liste d'aucune, une ou plusieurs chaines indiquant le comportement des fonctions `read` et *file_read* lorsque des lignes vides ou partielles sont lues dans le fichier.

`_read = ()` indique que ces fonctions génèreront une erreur en cas de ligne vide ou incomplète.

Les valeurs possibles sont :

- "skipbrokenlines" : les lignes partielles sont ignorées et un message d'information indique les numéros de lignes ignorées.
- "skipemptylines" : les lignes vides sont ignorées et un message d'information indique les numéros de lignes ignorées.

Valeur par défaut = `( )`.

```
Exemple :
> t1=1,3;
t1  Vecteur de reels double-precision : nb reels =3
> t2=11,13;
t2  Vecteur de reels double-precision : nb reels =3
> f=file_open(temp001, write);
f  = fichier "temp001" ouvert en ecriture
> file_writemsg(f,"# header line\n");
> file_write(f,t1);
> file_writemsg(f,"err...\n");
> file_write(f,t2);
> file_close(f);
> _read= ( "skipbrokenlines");
_read      =    ( "skipbrokenlines" )
> vnumR Q;
> read(temp001,  Q);
 Information : La ligne 1 est ignoree : elle est incomplete ou vide
```

```
 Information : La ligne 5 est ignoree : elle est incomplete ou vide
> writes(Q);
+1.0000000000000000E+00
+2.0000000000000000E+00
+3.0000000000000000E+00
+1.1000000000000000E+01
+1.2000000000000000E+01
+1.3000000000000000E+01
```

3.26 _read_history

`chaine _read_history` [Variable]

`_read_history` = <chaine> ;

`_read_history` indique le nom du fichier dans lequel les erreurs ignorées par *_read* sont enregistrées. Ce fichier est vidé lors de l'affectation de `_read_history`. Chaque ligne du fichier texte est constituée de deux colonnes: le numéro de la ligne et le nom du fichier où l'erreur s'est produite.

`_read_history = ""` indique que les erreurs ignorées par *_read* ne sont pas enregistrées dans un fichier.

Valeur par défaut = "".

```
Exemple :
> t1=1,3;
t1  Vecteur de reels double-precision : nb reels =3
> t2=11,13;
t2  Vecteur de reels double-precision : nb reels =3
> f=file_open(temp001, write);
f  = fichier "temp001" ouvert en ecriture
> file_writemsg(f,"# header line\n");
> file_write(f,t1);
> file_writemsg(f,"err...\n");
> file_write(f,t2);
> file_close(f);
> _read=( "skipbrokenlines");
_read      =    ( "skipbrokenlines" )
> _read_history="errors.dat";
_read_history = "errors.dat"
> vnumR Q;
> read(temp001,  Q);
 Information : La ligne 1 est ignoree : elle est incomplete ou vide
 Information : La ligne 5 est ignoree : elle est incomplete ou vide
> vnumR lerr;
> fmt="%g %s";
fmt = "%g %s"
> read("errors.dat", fmt, lerr, ferr);
> writes("%g\n",lerr);
1
5
> afftab(ferr);
ferr[1] =  "temp001"
ferr[2] =  "temp001"
```

3.27 _time

`booleen _time` [Variable]

`_time` {`on`, `off`};

active ou désactive l'affichage du temps partiel après chaque commande exécutée.

Lors de la commande `_time on`, un appel implicite à `time_s` est effectué et au retour de la commande, le temps 0.00s est affiché.

Valeur par défaut = `off`.

```
Exemple :
> macro temps
{
s=(1+x+y+z)**20$
_time on;
s=(1+x+y+z)**20$
q=s$
_time off;
};
> %temps;
00.0s
08.10s
00.12s
>
```

3.28 _userlibrary_path

`chaine _userlibrary_path` [Variable]

`_userlibrary_path` = <chaine> ;

`_userlibrary_path` = "chemin du repertoire des fonctions utilisateurs";

Lors de l'affectation de cette variable, TRIP lit récursivement dans le dossier spécifié tous les fichiers ayant une extension .t. Il charge toutes les macros de ces fichiers qui sont précédées du commentaire commencant par `//!trip extern_function` et les rend visible à l'utilisateur sous forme de fonction.

Le format du commentaire est :

`//!trip extern_function nom_de_la_fonction attribute="parametre_attribute" "parametres_inout"`

La chaine `parametre_attribute` peut contenir les valeurs suivantes séparées par des virgules. Si cette chaine est vide, alors la fonction est publique et visible par l'utilisateur.

- private : la fonction est interne et n'est pas visible pour l'utilisateur.

`parametres_inout` doit contenir autant d'éléments que la fonction. Chaque élément est s'eparé par une virgule. Un élément peut avoir les valeurs suivantes :

- in : le paramètre est en entrée seule.
- out : le paramètre est en sortie seule.
- inout : le paramètre est en entrée/sortie.

Plusieurs macros peuvent partager la même valeur `nom_de_la_fonction` si elles ont un nombre différent de paramètres.

Valeur par défaut = "".

```
Exemple :

_userlibrary_path="./myuserlibrary/";
userfunc1(3);
userfunc2(3,y);
```

Le dossier myuserlibrary contient le fichier suivant :

```
Exemple :
//!trip extern_function userfunc1  "in"
macro userfunc1[x]
{
    return 2*x+1;
};

 //!trip extern_function userfunc2  "in,out"
macro userfunc2[x,y]
{
    y=2*x+1;
};
```

3.29 I

`complexe i` [constante]

`complexe I` [constante]

$i^2 = -1$

```
Exemple :
>z = x + y*i;
z(x,y) = (0+i*1)*y+1*x
```

3.30 PI

`réel pi` [constante]

`réel PI` [constante]

Le symbole mathématique π

```
Exemple :
>z = pi;
 3.14159265358979
```

4 Variables

4.1 crevar

`crevar` [Fonction]

`crevar` (<chaine ou nom> *radical*, <entier> *indice1*, <entier> *indice2*,...);

Elle crée et retourne une variable de nom radical+"_"+ indice1 + ... + "_" + indicen.

`crevar` (<chaine ou nom> *radical*);

Elle crée et retourne une variable de nom radical. Cette fonction permet de créer des variables avec un nom non standard.

```
Exemple :
> dimvar t[1:3];
> t[1]:=crevar("t",1);
t[1] =  t_1 =                          1*t_1

> t[3]:=crevar("tab",3,2,1);
t[3] =  tab_3_2_1 =                          1*tab_3_2_1

> afftab(t);
t[1] =  t_1 =                          1*t_1

t[2] =
t[3] =  tab_3_2_1 =                          1*tab_3_2_1

> dimvar u[1:2];
> u[1]:=crevar("\bar{X}");
u[1] =  \bar{X} =                          1*\bar{X}

> u[1];
u[1] =  \bar{X} =                          1*\bar{X}

>
```

4.2 Operateurs

`:=` [Operateur]

<identificateur> `[...]` `:=` <variable> ;

Elle crée une référence vers une variable existante.

```
Exemple :
> // w est une reference vers x
> s=(1+x+y)**3$
> w:=x$
> stat(w);
Variable x type : 2     ordres :  2 2 2 2
 dependances :
 variables dependant de celle-ci :


> deriv(s,w);
```

```
                        3
 +                      6*y
 +                      3*y**2
 +                      6*x
 +                      6*x*y
 +                      3*x**2

>
```

5 Series

5.1 Operateurs

`+ - * /` [Operateur]

TRIP effectue l'addition (ou le plus unaire), la multiplication, la soustraction (ou moins unaire) et la division entre les séries, variables, constantes ou expressions.

Pour additionner, multiplier, soustraire et diviser terme à terme des tableaux, il existe l'opérateur +, *, - et /.

Les opérateurs +, *, -, / s'appliquent aux vecteurs numériques.

Le + ou - unaire s'applique à tous les types.

Remarque: La division par une série n'est pas supportée. Il faut utiliser la fonction `div` pour une division euclidienne.

```
Exemple :
> s=1+x-y*z+t/2;
s(x,y,z,t) = 1 + 1/2*t - 1*y*z + 1*x
> _modenum=NUMRATMP;
_modenum   =    NUMRATMP
> p=5/3;
p = 5/3
> _modenum=NUMQUAD;
_modenum   =    NUMQUAD
> p=5/3;
p =   1.66666666666666666666666666666667
```

`** ^` [Operateur]

<série> **<réel>

Elle effectue l'élevation à la puissance.

Elle accepte des exposants à valeurs entières.

La puissance **-1 sur un tableau à 2 dimensions égales effectue l'inversion matricielle.

Remarque : Par contre, la puissance terme à terme ** n'existe pas pour les tableaux.

```
Exemple :
> u=2**3;
8
> z=(1+i)**3;
(-2+i*2)

> s=(1+x+y)**2;
s(y,x) = 1 + 2*x + 1*x**2 + 2*y + 2*y*x + 1*y**2
```

5.2 Fonctions usuelles

`size` [Fonction]

`size(`<série> *s* `)`

Elle retourne le nombre de termes de la série *s*.

```
Exemple :
> s=1+x*y+z*t-i*p;
s(x,y,z,t,p) = 1 + (-0-i*1)*p + 1*t*z + 1*y*x
> size(s);
    4
```

5.3 Derivation et integration

5.3.1 deriv

`deriv` [Fonction]

`deriv(<série> , <variable> )`

Elle dérive la série par rapport à une variable .

```
Exemple :
> s=(1+x+y)**2;
s(x,y) = 1 + 2*y + 1*y**2 + 2*x + 2*x*y + 1*x**2
> deriv(s,x);
    2 + 2*y + 2*x
```

5.3.2 integ

`integ` [Fonction]

`integ(<série> , <variable> )`

Elle intègre la série par rapport à une variable .

```
Exemple :
>  s=(1+x+y)**2;
s(x,y) = 1 + 2*y + 1*y**2 + 2*x + 2*x*y + 1*x**2
> integ(s,x);
    1*x + 2*x*y + 1*x*y**2 + 1*x**2 + 1*x**2*y + 1/3*x**3
```

5.4 Division euclidienne

`div` [Commande]

`div(<série> f, <série> g, <identificateur> q, <identificateur> r )`

Elle calcule le quotient et le reste de la division de f par g tel que $f = q \times g + r$ et degre(r)<degre(g). Le quotient est stocké dans q et le reste dans r.

```
Exemple :
> div(x^3+x+1, x^2+x+1, q,r);
 > q;
q(x) =
 -                        1
 +                        1*x

> r;
r(x) =
                          2
 +                        1*x
```

5.5 Selection

5.5.1 coef_ext

`coef_ext` [Fonction]

`coef_ext(<série> , (<variable> ,<entier> ),...)`

Elle récupère le coefficient de la variable à la puissance désirée dans la série.

`coef_ext`(S,(X,n),(Y,m)) retourne le coefficient de X^n*Y^m dans la série S.

```
Exemple :
>  S= (1+x+y+z)**4 $
> coef_ext(S,(x,1),(y,2));
  12 + 12*z
```

5.6 Evaluation

5.6.1 coef_num

`coef_num` [Fonction]

`coef_num(<série> , (<variable> ,<constante> ),...)`

Elle substitue rapidement dans une série une (ou des) variable(s) par une (ou des) constantes numériques.

```
Exemple :
> S= (1+x+y+z)**4 $
> coef_num(S,(x,0.1),(y,2));
    923521/10000 + 29791/250*z + 2883/50*z**2 + 62/5*z**3 + 1*z**4
```

5.6.2 evalnum

`evalnum` [Fonction]

`evalnum(<série> , {REAL/COMPLEX} , (<variable> ,<vec. num.> ),...)`

Elle évalue la série en substituant les variables par leur valeur dans le vecteur numérique associé.

La fonction retourne un vecteur numérique de réels si `REAL` a été précisé sinon un vecteur numérique de complexes si `COMPLEX` a été précisé.

Les vecteurs numériques doivent être de taille identique.

```
Exemple :
> serie=sin(x+y)-2*y;
serie(y,_EXy1,_EXx1) =
   (                    -0+i*                   0.5)*_EXy1**-1*_EXx1**-1
 + (                     0-i*                   0.5)*_EXy1*_EXx1
 -                      2*y

> TABX=0,pi,pi/6;
TABX  Vecteur de reels double-precision : nb reels =7
> TABY=-pi,0,pi/6;
TABY  Vecteur de reels double-precision : nb reels =7
> TABRES=evalnum(serie,REAL,(x,TABX),(y,TABY));
TABRES  Vecteur de reels double-precision : nb reels =7
> writes(TABRES);
+6.2831853071795862E+00
+4.3699623521985504E+00
+3.3227648010019526E+00
+3.1415926535897931E+00
+2.9604205061776341E+00
+1.9132229549810371E+00
+1.2246467991473532E-16
> writes(TABX);
+0.0000000000000000E+00
```

```
+5.2359877559829882E-01
+1.0471975511965976E+00
+1.5707963267948966E+00
+2.0943951023931953E+00
+2.6179938779914940E+00
+3.1415926535897931E+00
> TABRES=evalnum(serie,COMPLEX,(x,TABX),(y,TABY));
TABRES  Vecteur de complexes double-precision : nb complexes =7
> writes(TABRES);
+6.2831853071795862E+00 +1.4997597826618576E-32
+4.3699623521985504E+00 +0.0000000000000000E+00
+3.3227648010019526E+00 +0.0000000000000000E+00
+3.1415926535897931E+00 +0.0000000000000000E+00
+2.9604205061776341E+00 -5.5511151231257827E-17
+1.9132229549810371E+00 +5.5511151231257827E-17
+1.2246467991473532E-16 +0.0000000000000000E+00
>
```

6 Constantes

6.1 Fonctions usuelles

La plupart des routines sont décrites dans (voir Chapitre 10 [Vecteurs numeriques], page 55).

6.1.1 factorielle

`fac` [Fonction]

`fac( <entier> n);`

Elle retourne $n!$ (fonction factorielle).

```
Exemple :
> fac(3);
                          6
> n=5;
n =                       5
> fac(n+1);
                        720
>
```

6.2 Entree/Sortie sur les reels

Ces routines assurent la lecture ou l'écriture séquentielle d'un fichier texte contenant uniquement des réels.

`ecriture` [Commande]

`ecriture(<nom fichier> );`

Cette fonction ouvre un fichier en écriture dans le répertoire spécifié par `_path`. Si le fichier n'existe pas, il est crée automatiquement.

```
Exemple : :
> ecriture("fichier1.dat");
```

`lecture` [Commande]

`lecture(<nom fichier> );`

Cette fonction ouvre un fichier en lecture dans le répertoire spécifié par `_path`.

```
Exemple : :
> lecture("fichier1.dat");
```

`print` [Commande]

`print(<réel> );`

Cette fonction écrit dans le fichier ouvert en écriture(avec `ecriture`) un réel en double-précision. Write a double-precision floating-point number in the file opened (with the command `ecriture`).

```
Exemple : :
> ecriture("fichier1.dat");
> print(atan(1)); /* on écrit pi/4 dans fichier1.dat */
```

`read` [Fonction]

`read;`

Cette fonction lit dans le fichier ouvert en lecture(avec `lecture`) et retourne un réel en double-précision.

```
Exemple : :
> ecriture("fichier1.dat");
> print(atan(1));
> close;
> lecture("fichier1.dat");
>  s=read;
s = 0.785398163397448
>  close;
```

`close` [Commande]

`close;`

Cette fonction ferme le fichier de réels s'il est ouvert (en écriture ou en lecture).

```
Exemple : :
> ecriture("fichier1.dat");
> print(atan(1));
> close;
```

7 Chaines de caracteres

7.1 Declaration et affectation

La déclaration de chaine de caractères est implicite. Elle est automatiquement effectuée lors d'une affectation. Pour produire un guillemet (") dans une chaine, il suffit de le doubler.

<nom> = <chaine> ;

```
Exemple :
/* L'exemple suivant declare les chaines ch et ch2. */
> ch="file";
ch = "file"
> ch2=ch+".txt";
ch2 = "file.txt"
```

7.2 Concatenation

<nom> = <chaine> + <chaine> ;

L'opérateur + concatène deux chaines de caractères. La longueur des chaines n'est pas limitée.

```
Exemple :
> ch = "file" + "1";
ch = "file1"
> sch = "../" + ch;
sch = "../file1"
> sch1 = sch + "." + str(12);
sch1 = "../file1.12"
```

7.3 Repetition

* [Operateur]

<chaine> * <entier>

<entier> * <chaine>

L'opérateur * répète, le nombre de fois spécifié par l'entier, la chaine de caractères. L'entier doit être positif ou nul. Si l'entier est 0, la chaîne retournée est vide.

```
Exemple :
> s=" %g";
s = " %g"
> format=4*s+"\n";
format = " %g %g %g %g\n"
> t=1,10$
> writes(format, t,t**2, t**3, t**4);
```

7.4 Extraction

[] [Operateur]

<chaine> [<entier> *j*]

Elle retourne le caractère ayant cet indice *j*. Les indices commencent à 1.

<chaine> [<entier> *binf* : <entier> *bsup*];

Elle retourne une chaine contenant uniquement les caractères situés entre les bornes inférieures et supérieures.

Si la borne inférieure est omise, alors sa valeur est 1. Si la borne supérieure est omise, alors sa valeur est la longueur de la chaine.

Remarque : toutes les combinaisons d'omissions sont permises.

```
Exemple :
> s="bonjour";
s = "bonjour"
> s1=s[4];
s1 = "j"
> s2=s[:3];
s2 = "bon"
> s3=s[4:5];
s3 = "jo"
>
```

7.5 Comparaison

`==` [Operateur]

<chaine> == <chaine>

L'opérateur == retourne vrai si les chaines de caractères sont identiques sinon elle retourne faux.

```
Exemple :
> s="monchemin";
s = "monchemin"
> if (s=="monchemin") then { msg "true"; } else { msg "false"; };
true
>
```

`!=` [Operateur]

<chaine> != <chaine>

L'opérateur != retourne faux si les chaines de caractères sont identiques sinon elle retourne vrai.

```
Exemple :
> s="monchemin";
s = "monchemin"
> if (s=="MON") then { msg "true"; } else { msg "false"; };
false
>
```

`<` [Operateur]

<chaine> *s1* < <chaine> *s2*

L'opérateur < retourne true si *s1* est plus petit que *s2* selon l'ordre lexicographique sinon elle retourne faux.

```
Exemple :
> s="abc";
s = "abc"
> if (s<"abd") then { msg "true"; } else { msg "false"; };
true
>
```

`<=` [Operateur]

<chaine> *s1* `<=` <chaine> *s2*

L'opérateur <= retourne true si *s1* est plus petit que ou égal à *s2* selon l'ordre lexicographique sinon elle retourne faux.

```
Exemple :
> s="abc";
s = "abc"
> if (s<="abe") then { msg "true"; } else { msg "false"; };
true
>
```

`>` [Operateur]

<chaine> *s1* `>` <chaine> *s2*

L'opérateur > retourne true si *s1* est plus grand que *s2* selon l'ordre lexicographique sinon elle retourne faux.

```
Exemple :
> s="abc";
s = "abc"
> if (s>"abd") then { msg "true"; } else { msg "false"; };
false
>
```

`>=` [Operateur]

<chaine> *s1* `>=` <chaine> *s2*

L'opérateur >= retourne true si *s1* est plus grand que ou égal à *s2* selon l'ordre lexicographique sinon elle retourne faux.

```
Exemple :
> s="abc";
s = "abc"
> if (s>="abe") then { msg "true"; } else { msg "false"; };
false
>
```

7.6 Conversion d'entier, de reel ou de serie en chaines

`str` [Fonction]

`str`(<entier>);

`str`(<réel>);

`str`(<série>);

`str`(<chaine> *format*, <réel>);

Elle convertit un entier, un réel ou un objet en chaine de caractères. Si *format* est spécifié, alors l'entier ou le réel est convertit suivant celui-ci. Le *format* est similaire à celui-ci de la fonction printf du langage C.

Les indicateurs de conversion sont

- g,G le nombre est converti sous forme de réel (notation xxxx.xxxx ou notation scientifique [-]d.dddE+-dd) si exposant trop grand).
- e,E le nombre est converti sous forme de réel (notation scientifique [-]d.dddE+-dd).
- f,F le nombre est converti sous forme de réel (notation [-]ddddd.ddddd).

– d,i le nombre est converti sous forme d'entier. Si l'argument est un réel, alors seule sa partie entière est convertit. En mode numérique NUMRAT ou NUMRATMP, les rationnels sont écrits sous la forme "numérateur/dénominateur".

Les modificateurs de longueur ne sont pas acceptées. Par exemple, le format "%lg" est refusé.

En mode numérique NUMRAT ou NUMRATMP, les rationnels sont écrits sous la forme "numérateur/dénominateur" si aucun format n'est spécifié.

```
Exemple :
> ch = str(1235);
ch = "1235"
> ch = str(int(2*pi));
ch = "6"
> s = str(2E3);
s = "2000"
> s = str("%.4g",pi);
s = "3.142"
> _modenum = NUMRATMP;
_modenum   =    NUMRATMP
> s = str("%d", 3/2);
s = "3/2"
> s = str("%g", 3/2);
s = "1.5"
> s = str(1+x**2);
s = "   1
 + 1*x**2

"
>
```

7.7 Conversion d'une liste d'entiers ou de reels en chaines

`msg` [Fonction]

`msg(`<chaine> *textformat*`,` <réel> *x*`, ... );`

Elle génère une chaine contenant le texte formaté accompagné de constantes réelles.

Le formatage est identique à celui de de la commande printf du langage C (voir Section 7.6 [str], page 33, pour les formats acceptés). Le format doit être une chaine et peut être sur plusieurs lignes.

Pour écrire un guillemet, il faut le doubler.

```
Exemple :
> ch = msg("pi=%g pi=%.8E",pi,pi);
ch = "pi=3.14159 pi=3.14159265E+00"
> _modenum=NUMRATMP;
_modenum   =    NUMRATMP
> s=msg("%d %g", 1/11, 1/11);
s = "1/11 0.0909091"
```

7.8 Conversion d'une chaine en entier, reel

`coef_num` [Fonction]

`coef_num(` <chaine>);

Elle convertit une chaine en un entier ou un réel.

```
Exemple :
> d = coef_num("-42");
d =                          -42
> r = coef_num("3.14E0");
r =                         3.14
>
```

7.9 Longueur de chaines

`size` [Fonction]

`size`(<chaine>);

Elle retourne la longueur de la chaine, c'est-à-dire le nombre de caractères.

```
Exemple :
> ch = "1235";
ch = "1235"
> size(ch);
 4
```

8 Tableaux

Il existe quatre sortes de tableaux dans TRIP :

- Les tableaux de séries et constantes.
- Les tableaux de variables.
- Les vecteurs numériques.
- Les matrices numériques.

Les matrices sont des tableaux de séries et de constantes. Ce chapitre ne discute pas des vecteurs et matrices numériques.

8.1 Declaration de tableaux de series

`dim` [Commande]

`dim` <nom> [<liste_dimension>], ...;

Elle déclare un ou plusieurs tableaux de séries en spécifiant le nombre de dimensions.

Chaque dimension est séparée par une virgule. Une dimension est composée de deux ou trois entiers séparés par des : qui indiquent l'indice du premier et du dernier élément, ainsi que le pas de cette dimension. Si le troisième entier est absent, alors le pas vaut 1. Les dimensions d'un tableau peuvent être stockées dans un identificateur.

Ce type de tableaux peut contenir des séries, des constantes, des troncatures, des chaines de caractères, ... mais pas de variables.

```
Exemple :
> // Ici, on declare un tableau tl a deux dimensions
> // de séries ou de constantes
> dim tl [1:22,-2:6];
> // on declare 3 tableaux t1, t2 , t3 avec des dimensions differentes
> bounds1 = 1:4;
bounds1 = bornes [ 1:4 ]
> dim t1[bounds1], t2[5:6], t3[-1:3];
> // on declare un tableau avec un pas different de 1
> dim t5[1:5:2];
> t5[:]=7;
> afftab(t5);
t5[1] =                          7
t5[3] =                          7
t5[5] =                          7
>
```

8.2 Initialisation d'un tableau de series

<nom> = [<série> ,... : <série> , ...];

<nom> = dim[<série> ,... : <série> , ...];

Cette commande crée et initialise un tableau de séries avec les séries ou constantes fournies.

Les : indiquent une nouvelle ligne et les , séparent les colonnes. Il doit y avoir le même nombre de colonne pour chaque ligne.

```
Exemple :
> // declare un tableau de series a 2 dimensions contenant des series
> tab=[1,2+2*x:(1+x)**2,(2+2*x)**2];
```

```
tab [1:2, 1:2 ] nb elements = 4

> stat(tab);
Tableau de series
 tab [ 1:2 , 1:2 ]
 liste des elements du tableau :
tab [ 1 , 1 ] =
 constante de nom tab et de valeur    1
tab [ 1 , 2 ] =
 serie tab (  x )
 nombre de variables : 1   taille du descripteur : 80 octets
 nombre de termes : 2   taille : 304 octets
tab [ 2 , 1 ] =
 serie tab (  x )
 nombre de variables : 1   taille du descripteur : 80 octets
 nombre de termes : 3   taille : 352 octets
tab [ 2 , 2 ] =
 serie tab (  x )
 nombre de variables : 1   taille du descripteur : 80 octets
 nombre de termes : 3   taille : 352 octets
> // declare un tableau de series a 2 dimensions contenant des constantes
> tab2=[1,2,3:4,5,6];

tab2 [1:2, 1:3 ] nb elements = 6

>
```

`dimvar` [Commande]

`dimvar` <nom> [<liste_dimension>], ...;

Elle déclare un ou plusieurs tableaux de variables en spécifiant le nombre de dimension.

Chaque dimension est séparée par une virgule. Une dimension est composée de deux entiers qui indiquent l'indice du premier et du dernier élément de cette dimension.

Ce type de tableaux ne peut contenir que des variables. Pour affecter une variable existante à un tableau de variables, on remplace le symbole `=` par `:=` .

```
Exemple :
> // on declare un tableau t2 de variables a  une dimension
> dimvar t2[1:5];
> // Maintenant, si on fait:
> t2[1] := x;
t2[1] =  x =                          1*x

> // on aura la derivee de S par rapport a x.
> S=1+3*x$
> deriv(S,t2[1]);
                          3
>
> // on declare 2 tableaux de variables tv1 et tv2
> dimvar tv1[1:2], tv2[1:3];
>
```

8.3 Initialisation d'un tableau de variables

<nom> = dimvar[<série> ,... : <série> , ...];

Cette commande crée et initialise un tableau de variables avec les variables fournies.

Les : indiquent une nouvelle ligne et les , séparent les colonnes. Il doit y avoir le même nombre de colonne pour chaque ligne.

```
Exemple :
> // on declare un tableau t2 de variables avec les variables x, y et z et u
> 1+x+y+z$
> t2 = dimvar[x:y:z:u];

t2 [1:4 ] nb elements = 4

>
```

8.4 Generation d'un tableau de variables

`tabvar` [Commande]

`tabvar`(<tableau de variables>);

Elle génère automatiquement les variables dans le tableau spécifié en utilisant comme nom des variables, le nom du tableau et leur indice.

Avant d'utiliser `tabvar`, il faut créer un tableau de variables à l'aide de `dimvar`.

```
Exemple :
> dimvar t[0:3];
> tabvar(t);
> afftab(t);
t[0] =  t_0 =                          1*t_0

t[1] =  t_1 =                          1*t_1

t[2] =  t_2 =                          1*t_2

t[3] =  t_3 =                          1*t_3

>
```

8.5 Affectation dans un tableau

`=` [Operateur]

<nom> = <tableau> ;

Elle affecte le tableau (opérations entre des tableaux) au nom de l'identificateur fourni.

<tableau> `[`<entier> , ...`]` = <opération> ;

Elle affecte à un élément du tableau l'opération. Cette opération peut être une série, une constante, un vecteur numérique, une troncature mais pas un tableau.

<tableau> `[` <entier> *binf* `:` <entier> *bsup* `:` <entier> *pas* `,...]=` <operation> ;

<tableau> `[` <entier> *binf* `:` <entier> *bsup* `,...]=` <operation> ;

Elle affecte une opération à une partie de tableau. Si cette opération est une série, une constante, un vecteur numérique, une troncature alors cette opération est copiée pour chaque élément du tableau. Si cette opération est un tableau, alors chaque élément de celui-ci est

affecté dans l'élément correspondant. Dans ce cas, elle vérifie que le nombre d'éléments et de dimensions est cohérent.

Si la borne inférieure est omise, alors sa valeur est 1. Si la borne supérieure est omise, alors sa valeur est la taille du tableau. Si le pas est omis, alors sa valeur est 1.

Lorsque le nombre de dimensions n'est pas connu ou variable, il est possible de mettre l'ellipsis ... en dernier argument pour indiquer une séquence de longueur quelconque :,:,: .

Remarque : toutes les combinaisons d'omissions sont permises.

```
Exemple :
> _affc=1$
> dim t[1:4,7:25];
> t[1,7]=1+x;
t[1,7] =    1
 + 1*x

> r=t;

r [1:4, 7:25 ] nb elements = 76

> dim t[1:4,7:25];
> dim t2[1:4];
> t2[:]=5;
> t[:,8]=t2;
> t[:,::2]=1+x;
> t[2,...]=3;
>
```

`:=` [Operateur]

<tableau> `[...]` `:=` <variable> ;

Elle affecte une variable à un élément du tableau de variables.

```
Exemple :
> dimvar X[1:4];
> X[1]:=crevar("e",1,1);
X[1] =  e_1_1 =                          1*e_1_1

> deriv(1+2*e_1_1,X[1]);
                           2
>
```

8.6 Taille d'un tableau

`size` [Fonction]

`size`(<tableau>);

`size`(<tableau> ,<entier>);

Elle retourne le nombre d'éléments du tableau contenu dans la première dimension ou dans la dimension spécifiée par l'entier.

Si la dimension spécifiée n'existe pas dans le tableau, alors la fonction retourne -1.

```
Exemple :
> dim t[1:4,7:25];
> size(t,2);
```

```
 19
> size(t);
 4
>
```

num_dim [Fonction]

num_dim(<tableau> t)

Elle retourne le nombre de dimensions du tableau t.

```
Exemple :
> dim t1[1:5];
> size(t1);
 5
> dim t2[1:22,-2:6];
> size(t2);
 22
>
```

8.7 Bornes d'un tableau

inf [Fonction]

inf(<tableau> ,<entier>);

Elle retourne la borne inférieure du tableau pour la dimension spécifiée par l'entier.

Si la dimension spécifiée n'existe pas dans le tableau, alors la fonction retourne -1.

```
Exemple :
> dim t[1:2,0:3];
> inf(t,2);
 0
> for k=inf(t,2) to sup(t,2) { t[:,k]= k$ };
> afftab(t);
t[1,0] =                        0
t[1,1] =                           1
t[1,2] =                           2
t[1,3] =                           3
t[2,0] =                        0
t[2,1] =                           1
t[2,2] =                           2
t[2,3] =                           3
>
```

sup [Fonction]

sup(<tableau> ,<entier>);

Elle retourne la borne supérieure du tableau pour la dimension spécifiée par l'entier.

Si la dimension spécifiée n'existe pas dans le tableau, alors la fonction retourne -1.

```
Exemple :
> dim t[1:2,0:3];
> sup(t,2);
 3
> for k=inf(t,1) to sup(t,1) { t[k,:]= k$ };
> afftab(t);
t[1,0] =                           1
```

```
t[1,1] =                        1
t[1,2] =                        1
t[1,3] =                        1
t[2,0] =                        2
t[2,1] =                        2
t[2,2] =                        2
t[2,3] =                        2
>
```

8.8 Affichage d'un tableau

`afftab` [Commande]

`afftab`(<tableau>);

Elle affiche le contenu du tableau de séries ou de variables.

```
Exemple :
> _affc=1$
> dim t[1:4];
> t[1]=1+x$
> t[2]=2*i$
> t[3]=({(x,y),2})$
> afftab(t);
t[1] =    1
 + 1*x

t[2] =    (0+i*2)
t[3] = ({(y, x), 2})
t[4] =
>
```

8.9 Extraction d'element

`[]` [Operateur]

<tableau> `[`<entier> ,...`]`;

Elle retourne l'élément ayant ces indices.

<vec. num.> `[` <entier> *binf* `:` <entier> *bsup* `:` <entier> *pas*,... `];`

<vec. num.> `[` <entier> *binf* `:` <entier> *bsup*, ... `];`

Elle retourne un tableau contenant uniquement les éléments situés entre les bornes inférieures et supérieures avec le pas spécifié.

Si la borne inférieure est omise, alors sa valeur est la borne inférieure de la dimension correspondante. Si la borne supérieure est omise, alors sa valeur est borne supérieure de la dimension correspondante. Si le pas est omis, alors sa valeur est 1.

Lorsque le nombre de dimensions n'est pas connu ou variable, il est possible de mettre l'ellipsis ... en dernier argument pour indiquer une séquence de longueur quelconque :,:,: .

Remarque : toutes les combinaisons d'omissions sont permises.

```
Exemple :
> _affc=1$
> dim t[1:4];
> t[1]=1+x$
> s=t[1];
```

```
s(x) =
   1
 + 1*x

> t1=t[3:];

t1 [3:4 ] nb elements = 2

> dim t[1:4,5:10];
> v2=t[:,6:];

v2 [1:4, 6:10 ] nb elements = 20

> v3=t[4,...];

v3 [5:10 ] nb elements = 6
```

`select` [Fonction]

`select ( <condition> , <tableau> );`

Elle retourne un tableau contenant uniquement les éléments du tableau où la condition est vraie.

Si la condition est toujours fausse, elle retourne la constante 0. Pour tester si la constante 0 a été retournée, le test `if (size(resultat)==0) then { ... } else { ... }` peut être utilisé

La condition et le tableau doivent avoir le même nombre d'éléments et une seule dimension.

```
Exemple :
> // extrait les elements de tm telque  t1[j]=1
> _modenum=NUMRATMP;
_modenum   =    NUMRATMP
> dim t1[1:5], tm[1:5]$
> for j=1 to 5 { t1[j]=abs(3-j)$ tm[j]=(1+x)**j$ };
> q = select(t1==1, tm);

q [1:2 ] nb elements = 2

> if (size(q)!=0) then { afftab(q); };
q[1] =    1
 + 2*x
 + 1*x**2

q[2] =    1
 + 4*x
 + 6*x**2
 + 4*x**3
 + 1*x**4

>
```

8.10 Operations sur les tableaux de series

La plupart des fonctions usuelles sont décrites dans (voir Chapitre 10 [Vecteurs numeriques], page 55). Les fonctions suivantes peuvent être appliquées aux tableaux de séries (comme sur les séries) pour éviter les boucles for :

- coef_ext voir Section 5.5.1 [coef_ext], page 26,
- coef_num voir Section 5.6.1 [coef_num], page 27,
- deriv voir Section 5.3.1 [deriv], page 26,
- integ voir Section 5.3.2 [integ], page 26,

8.10.1 Produit matriciel

`&*` [Operateur]

`<tableau> &* <tableau>`

Elle calcule le produit matriciel de deux tableaux à 1 ou 2 dimensions.

Elle calcule r=a&*b tel que $r[i,j] = \sum_{k=1}^{size(a,1)} a_{i,k} \times b_{k,j}$

Si le tableau a une seule dimension, alors il est considéré comme un vecteur-colonne.

Remarque: le nombre de colonnes du premier tableau doit être égal au nombre de lignes du second tableau.

```
Exemple :
> _affc=1$ _affdist=0$
> t=[x,y:x-y,x+y]$
> t2=[x+y,x-y:x,y]$
> r=t&*t2;

r [1:2, 1:2 ] nb elements = 4

> afftab(r);
r[1,1] =    2*y*x + x**2
r[1,2] =    y**2 - y*x + x**2
r[2,1] =  - y**2 + y*x + 2*x**2
r[2,2] =    2*y**2 - y*x + x**2
>
```

8.10.2 Determinant

`det` [Fonction]

`det( <tableau>` M)

`det( <tableau>` M`, <chaine>` *method*)

Elle calcule le déterminant d'un tableau à 2 dimensions (matrice carrée).

- si M est une matrice numérique, un algorithme LU est utilisé dans tous les modes numeriques. Lors de calculs en double-precision, la librairie Lapack est utilisée.
- si M est une matrice polynomiale, un algorithme de Bareiss est utilisé par défaut.

Si *method* est spécifié, elle utilise cette méthode pour calculer le déterminant si le contenu de la matrice le permet. Si le contenu ne le permet pas, elle utilise l'algorithme par défaut.

Les valeurs possibles de *method* sont :

- "minor" : algorithme des mineurs qui requiert une large mémoire. En pratique, cela n'est utilisable que pour les matrices de taille inférieure à 12.
- "bareiss" : algorithme de Gauss-Bareiss.

- "interpol" : algorithme basé sur l'interpolation. Elle n'est utilisable que si la matrice est polynomiale à coefficients entiers ou rationnels.
- "interpol_modular" : algorithme basé sur l'interpolation, l'arithmétique modulaire et le reste chinois. Elle n'est utilisable que si la matrice est polynomiale à coefficients entiers.

```
Exemple :
> t=[9,0,7
    :1,2,3
    :4,5,6];

t [1:3, 1:3 ] nb elements = 9

> det(t);
                        -48
>
```

8.10.3 Inverse

`**-1` [Fonction]

<tableau> **-1

Elle calcule l'inverse d'un tableau à 2 dimensions (matrice carrée).

Remarque : lors de calculs en double-precision, la librairie Lapack est utilisée. Un algorithme LU est utilisé dans tous les cas.

```
Exemple :
> _modenum=NUMRATMP$
> t=[9,0,7
    :1,2,3
    :4,5,6];

t [1:3, 1:3 ] nb elements = 9

> r=t**-1;

r [1:3, 1:3 ] nb elements = 9

> afftab(r);
r[1,1] =    1/16
r[1,2] =  - 35/48
r[1,3] =    7/24
r[2,1] =  - 1/8
r[2,2] =  - 13/24
r[2,3] =    5/12
r[3,1] =    1/16
r[3,2] =    15/16
r[3,3] =  - 3/8
>
```

8.10.4 Valeurs propres

`eigenvalues` [Fonction]

`eigenvalues`(<tableau>)

Elle calcule les valeurs propres d'un tableau à 2 dimensions (matrice carrée) en utilisant un algorithme QR (librairie lapack).

```
Exemple :
> t=[9,0,7
    :1,2,3
    :4,5,6];

t [1:3, 1:3 ] nb elements = 9

> l=eigenvalues(t);

l [1:3 ] nb elements = 3

> afftab(l);
l[1] =          13.76915041895731
l[2] =          4.084362034809442
l[3] =        -0.8535124537667539
```

8.10.5 Vecteurs propres

`eigenvectors` [Commande]

`eigenvectors`(<tableau> *MAT*, <tableau> *TVECT*, <tableau> *TVAL*)

`eigenvectors`(<tableau> *MAT*, <tableau> *TVECT*)

Elle calcule les vecteurs propres d'un tableau *MAT* à 2 dimensions (matrice carrée). Elle stocke les vecteurs propres dans le tableau *TVECT* et les valeurs propres dans le tableau *TVAL*.

Elle utilise un algorithme QR (librairie lapack).

Chaque colonne de *TVECT* correspond à un vecteur propre. Chaque vecteur est normalisé.

```
Exemple :
> t=[9,0,7
    :1,2,3
    :4,5,6];

t [1:3, 1:3 ] nb elements = 9

> eigenvectors(t,vectp,valp);
> afftab(vectp);
vectp[1,1] =        -0.8081690381494434
vectp[1,2] =        -0.7505769919446202
vectp[1,3] =        -0.4846638559537327
vectp[2,1] =         -0.209021306608322
vectp[2,2] =         0.3985224414326275
vectp[2,3] =        -0.5474094124384613
vectp[3,1] =         -0.550611386696964
vectp[3,2] =         0.5270806796287867
vectp[3,3] =         0.6822344772186747
> afftab(valp);
valp[1] =          13.76915041895731
valp[2] =          4.084362034809442
valp[3] =        -0.8535124537667539
> //  1er vecteur
> p=vectp[:,1]$
> afftab(p);
```

```
p[1] =        -0.8081690381494434
p[2] =         -0.209021306608322
p[3] =         -0.550611386696964
```

8.10.6 Arithmetique

Les tableaux doivent avoir le même nombre de dimensions et le même nombre d'éléments par dimension.

`+` [Operateur]

<tableau> + <tableau>

Elle retourne l'addition terme à terme de deux tableaux de séries.

`+` [Operateur]

<tableau> + <série>

<série> + <tableau>

Elle retourne la somme d'une série et de chaque élément du tableau de séries.

`*` [Operateur]

<tableau> * <tableau>

Elle retourne le produit terme à terme de deux tableaux de séries.

`*` [Operateur]

<tableau> * <série>

<série> * <tableau>

Elle retourne le produit terme à terme d'une série et d'un tableau de séries.

`-` [Operateur]

<tableau> - <tableau>

Elle retourne la soustraction terme à terme de deux tableaux de séries.

`-` [Operateur]

<tableau> - <série>

<série> - <tableau>

Elle retourne la différence entre une série et chaque élément du tableau de séries.

`/` [Operateur]

<tableau> / <tableau>

Elle retourne la division terme à terme de deux tableaux de séries.

`/` [Operateur]

<tableau> / <série>

<série> / <tableau>

Elle retourne la division entre une série et chaque élément du tableau de séries.

```
Exemple :
> _affc=1$
> t2=[x+y,x-y:x,y];

t2 [1:2, 1:2 ] nb elements = 4

> t=[x,y:x-y,x+y];
```

```
t [1:2, 1:2 ] nb elements = 4

> r=t*t2*i-t*x;

r [1:2, 1:2 ] nb elements = 4

> afftab(r);
r[1,1] =    (0+i*1)*x*y
 + (-1+i*1)*x**2

r[1,2] =    (-0-i*1)*y**2
 + (-1+i*1)*x*y

r[2,1] =    (1-i*1)*x*y
 + (-1+i*1)*x**2

r[2,2] =    (0+i*1)*y**2
 + (-1+i*1)*x*y
 - 1*x**2

>
```

8.11 Conversion

Pour la conversion en vecteurs numériques, (voir Section 10.12 [Conversion], page 94).

9 Structures et POO

Les structures sont composées de champs nommés ayant une visibilité globale ou privée. Des macros globales ou privées peuvent êtres associées aux structures. Celles-ci accèdent à l'ensemble de ces champs.

9.1 Déclaration de structure

`struct` [Commande]

`struct` <structtype> { <liste_parametres> ; `private` <liste_parametres> ; };

`struct` <structtype> { <identificateur> = <valeur> ; ... `private` <identificateur> = <valeur> ; `private` ... };

Elle déclare un nouveau type de structure dont les champs de la première liste peuvent être accédés de manière globale alors que ceux précédés par le mot-clé `private` ne seront visibles que par les macros associées à cette structure.

Les champs peuvent être initialisés à une valeur par défaut.

Exemple :

```
> // declaration de la structure POINT
> struct POINT {
  x, y;
};
>
> // declaration de la structure MyData
> struct MyData {
  x = 0;
  y = 0;
  z = -1;
  vnumR v([1:3]);
  dim armap[1:5];
  private name = "NONAME";
  private file = "/tmp/myfile";
};
> MyData a;
> afftab(a);
a@x =                        0
a@y =                        0
a@z =                          -1
a@v = a  Vecteur de reels double-precision : nb reels =3
a@armap =
a [1:5 ] nb elements = 5

a@name =  "NONAME"
a@file =  "/tmp/myfile"
```

9.2 Déclaration des identificateurs

`struct` [Commande]

`struct` <structtype> <identificateur> *p1*, <identificateur> *p2*, ... ;

Elle crée les identificateurs dont les noms sont donnés dans la liste. Ces identificateurs seront des structures du type spécifié. Aucun champ de ces structures n'est initialisé.

```
Exemple :
> // declaration de la structure POINT
> struct POINT {
  x, y;
};
> // declaration des variables de type POINT
> struct POINT p1, p2, p3;
> p1;
p1 = structure POINT
> afftab(p1);
p1@x = /* NON INITIALISE */
p1@y = /* NON INITIALISE */
```

9.3 Acces aux attributs

`@` [Operateur]

<structure> `@`<nom>

Elle permet d'accéder au champ de la structure. Si le champ est privé, celui-ci ne peut être accéder que depuis une macro associée à la structure.

```
Exemple :
> struct MyData {
  x,y,z;
  name, file;
};
> struct MyData s;
>
> s@x = 3;
s@x =                          3
> s@y = 1;
s@y =                          1
> s@z = s@x + 2*s@y;
s@z =                          5
> s@name = "temp001";
s@name =  "temp001"
> s@file=file_open(s@name,write);
s@file =  fichier "temp001" ouvert en ecriture
>
```

9.4 Affichage

`afftab` [Commande]

`afftab(`<structure> *x* `);`

Elle affiche la valeur de tous les champs de la structure *x*.

```
Exemple :
> struct MyData {
  x,y,z;
  name, file;
};
> struct MyData s;
>
```

```
> s@x = 3$
> s@y = 1$
> s@z = s@x + 2*s@y$
> s@name = "temp001"$
> s@file=file_open(s@name,write)$
> afftab(s);
s@x =                          3
s@y =                          1
s@z =                          5
s@name =  "temp001"
s@file =  fichier "temp001" ouvert en ecriture
>
```

9.5 Macro constructeur

`macro` [Commande]

`macro` <structtype> @ <structtype> { <corps> };

Elle déclare un constructeur de la structure sans paramètres et un corps de code trip. Cette macro est alors appelée implicitement dès qu'un objet de ce type est instancié.

La macro peut accéder à l'ensemble des champs privés ou non de la structure. L'accès aux champs se fait directement et n'a pas besoin d'être préfixé par `@`.

La variable *self* est automatiquement définie lors de l'exécution pour accéder à la valeur de l'ensemble de la structure qui a généré l'appel.

Exemple :

```
> struct MyData {
  t;
  vnumR v([1:3])$
};
> macro  MyData@MyData {
  private _ALL;
  t = log(2)$
  v[:]=4$
};
>
> MyData a;
> afftab(a);
a@t =         0.6931471805599453
a@v = a  Vecteur de reels double-precision : nb reels =3
>
>
```

9.6 Macros membres

9.6.1 Déclaration

`macro` [Commande]

`macro` <structtype> @ <nom> [<liste_parametres>] { <corps> };

`macro` <structtype> @ <nom> { <corps> };

`private macro` <structure> @ <nom> [<liste_parametres>] { <corps> };

`private macro` <structure> @ <nom> { <corps> };

Elle déclare une macro membre de la structure avec 0 ou plusieurs paramètres et un corps de code trip.

La macro peut accéder à l'ensemble des champs privés ou non de la structure. L'accès aux champs se fait directement et n'a pas besoin d'être préfixé par `@`.

La variable *self* est automatiquement définie lors de l'exécution pour accéder à la valeur de l'ensemble de la structure qui a généré l'appel. Elle est utile pour une affectation ou le passage de paramètres à d'autres macros.

Si la macro est définie comme privée, alors elle ne peut être appelée que par une autre macro associée à la même structure.

```
Exemple :
> struct MyData {
  x,y,z;
  private name, file;
};
>
> // Declare Init comme macro membre de MyData
> macro MyData@Init[vx,vy,vz,vname] {
    x = vx$
    y = vy$
    z = vz$
    name = vname$
    %OpenFile$
    afftab(self);
};
>
> // Declare OpenFile comme une macro privee de MyData
> private macro MyData@OpenFile {
    file = file_open(name,write)$
};
>
>
```

9.6.2 Exécution

% [opérateur]

<structure> % <nom> [<liste_parametres>];

<structure> % <nom> ;

Elle exécute une macro membre de la structure en déclarant l'identificateur *self* comme valeur la structure pendant son exécution.

```
Exemple :
> struct MyData {
  x,y,z;
  private name, file;
};
>
> macro MyData@Init[vx,vy,vz,vname] {
    x = vx$
    y = vy$
    z = vz$
    name = vname$
```

```
    %OpenFile$
    afftab(self);
};
>
> macro MyData@Clear {
    %CloseFile;
};
>
> private macro MyData@OpenFile {
    file = file_open(name,write)$
};
>
> private macro MyData@CloseFile {
    file_close(file);
};
>
> struct MyData s;
> // execute la macro Init
> s%Init[1,2,3,"temp001"];
s@x =                          1
s@y =                          2
s@z =                          3
s@name =  "temp001"
s@file =  fichier "temp001" ouvert en ecriture
>
> // execute la macro Clear
> s%Clear;
>
```

10 Vecteurs numeriques

Les données numériques, stockées dans ces vecteurs, sont toujours des réels ou complexes double-précision, quadruple-précision ou multiprecision suivant le mode numérique courant. Ces vecteurs numériques sont considérés comme des vecteurs colonnes.

Dans le mode numérique NUMRATMP, les nombres rationnels peuvent être stockés dans des vecteurs numériques de type rationnel.

10.1 Declaration

La déclaration explicite de vecteur numérique est nécessaire uniquement avant l'utilisation de la commande `read` , `readbin` (voir Section 10.8 [Entree/SortieTabNum], page 63) et `resize` . Elle est aussi nécessaire pour les tableaux de vecteurs numériques.

10.1.1 vnumR

`vnumR` [Commande]

`vnumR` <nom> `,` ... `;`

`vnumR` <nom> `[` <dimension d'un tableau> `]` `,` ... `;`

Elle déclare un vecteur de réels ou un tableau de vecteur numérique de réels.

La taille des vecteurs de réels est dynamique. Apres cette déclaration, les vecteurs de réels sont vides. Pour spécifier leur taille, il faut utiliser la commande `resize`.

La dimension permet de spécifier le nombre d'éléments du tableau de vecteurs de réels.

`vnumR` [Commande]

`vnumR` <nom> `([` 1: <entier> *n* `])` `,` ... `;`

`vnumR` <nom> `[` <dimension d'un tableau> `]` `([` 1: <entier> *n* `])` `,` ... `;`

Elle déclare un vecteur de *n* réels ou un tableau de vecteur numérique de *n* réels.

Après cette déclaration, les vecteurs de réels sont initialisés avec la valeur 0.

```
Exemple :
> vnumR A, C([1:5]);
> stat(A);
Vecteur numerique A de 0 reels  double-precision.
> stat(C);
Vecteur numerique C de 5 reels  double-precision.
taille en octets du tableau: 40
> bounds=1:2;
bounds = bornes [ 1:2 ]
> vnumR TE[bounds], T0[bounds]([1:6]);
> stat(TE);
Tableau de series
 TE [ 1:2 ]
 liste des elements du tableau :
TE [ 1 ] =
Vecteur numerique TE de 0 reels  double-precision.
TE [ 2 ] =
Vecteur numerique TE de 0 reels  double-precision.
> stat(T0);
Tableau de series
 T0 [ 1:2 ]
 liste des elements du tableau :
```

```
T0 [ 1 ] =
Vecteur numerique T0 de 6 reels  double-precision.
taille en octets du tableau: 48
T0 [ 2 ] =
Vecteur numerique T0 de 6 reels  double-precision.
taille en octets du tableau: 48
>
```

10.1.2 vnumC

vnumC [Commande]

vnumC <nom> , ... ;

vnumC <nom> [<dimension d'un tableau>] , ... ;

Elle déclare un vecteur de complexes ou un tableau de vecteur numérique de complexes.

La taille des vecteurs de complexes est dynamique. Apres cette déclaration, les vecteurs de complexes sont vides. Pour spécifier leur taille, il faut utiliser la commande `resize`.

La dimension permet de spécifier le nombre d'éléments du tableau de vecteurs de complexes.

vnumC [Commande]

vnumC <nom> ([1: <entier> *n*]) , ... ;

vnumC <nom> [<dimension d'un tableau>] ([1: <entier> *n*]) , ... ;

Elle déclare un vecteur de *n* complexes ou un tableau de vecteur numérique de *n* complexes.

Après cette déclaration, les vecteurs de complexes sont initialisés avec la valeur 0+i*0.

```
Exemple :
> vnumC A, C([1:5]);
> stat(A);
Vecteur numerique A de 0 complexes double-precision.
> stat(C);
Vecteur numerique C de 5 complexes double-precision.
taille en octets du tableau: 80
> vnumC TE[1:2], T0[1:2]([1:6]);
> stat(TE);
Tableau de series
 TE [ 1:2 ]
 liste des elements du tableau :
TE [ 1 ] =
Vecteur numerique TE de 0 complexes double-precision.
TE [ 2 ] =
Vecteur numerique TE de 0 complexes double-precision.
> stat(T0);
Tableau de series
 T0 [ 1:2 ]
 liste des elements du tableau :
T0 [ 1 ] =
Vecteur numerique T0 de 6 complexes double-precision.
taille en octets du tableau: 96
T0 [ 2 ] =
Vecteur numerique T0 de 6 complexes double-precision.
taille en octets du tableau: 96
>
```

10.1.3 vnumQ

vnumQ [Commande]

vnumQ <nom> , ... ;

vnumQ <nom> [<dimension d'un tableau>] , ... ;

Elle déclare un vecteur de nombre rationnels ou un tableau de vecteur numérique de rationnels.

La taille des vecteurs de rationnels est dynamique. Apres cette déclaration, les vecteurs de rationnels sont vides. Pour spécifier leur taille, il faut utiliser la commande `resize`.

La dimension permet de spécifier le nombre d'éléments du tableau de vecteurs de rationnels.

vnumQ [Commande]

vnumQ <nom> ([1: <entier> *n*]) , ... ;

vnumQ <nom> [<dimension d'un tableau>] ([1: <entier> *n*]) , ... ;

Elle déclare un vecteur de *n* rationnels ou un tableau de vecteur numérique de *n* rationnels.

Après cette déclaration, les vecteurs de rationnels sont initialisés avec la valeur 0.

```
Exemple :
> _modenum=NUMRATMP;
_modenum   =    NUMRATMP
> vnumQ A, C([1:5]);
> stat(A);
Vecteur numerique A de 0 rationnels.
> stat(C);
Vecteur numerique C de 5 rationnels.
taille en octets du tableau: 160
> vnumQ TE[1:2], T0[1:2]([1:6]);
> stat(TE);
Tableau de series
 TE [ 1:2 ]
 liste des elements du tableau :
TE [ 1 ] =
Vecteur numerique TE de 0 rationnels.
TE [ 2 ] =
Vecteur numerique TE de 0 rationnels.
> stat(T0);
Tableau de series
 T0 [ 1:2 ]
 liste des elements du tableau :
T0 [ 1 ] =
Vecteur numerique T0 de 6 rationnels.
taille en octets du tableau: 192
T0 [ 2 ] =
Vecteur numerique T0 de 6 rationnels.
taille en octets du tableau: 192
>
```

10.2 Initialisation

,,, [Commande]

<nom> = <réel> *binf* , <réel> *bsup* , <réel> *step* ;

Elle déclare un vecteur de réels tel que les éléments soient initialisés par une boucle (de *binf* à *bsup* avec un pas *bstep*):

for (j=1, valeur=*binf*) to valeur<=*bsup* step (j=j+1, valeur=valeur+*bstep*) <nom> [j]=valeur

Le pas *bstep* est optionnel; par défaut, sa valeur est 1.

```
Exemple :
> t=0,10;
t  Vecteur de reels double-precision : nb reels =11
> writes(t);
+0.0000000000000000E+00
+1.0000000000000000E+00
+2.0000000000000000E+00
+3.0000000000000000E+00
+4.0000000000000000E+00
+5.0000000000000000E+00
+6.0000000000000000E+00
+7.0000000000000000E+00
+8.0000000000000000E+00
+9.0000000000000000E+00
+1.0000000000000000E+01
> tab=-pi, 6, pi/100;
tab  Vecteur de reels double-precision : nb reels =291
> writes([::30],tab);
-3.1415926535897931E+00
-2.1991148575128552E+00
-1.2566370614359170E+00
-3.1415926535897887E-01
+6.2831853071795907E-01
+1.5707963267948966E+00
+2.5132741228718354E+00
+3.4557519189487733E+00
+4.3982297150257113E+00
+5.3407075111026483E+00
>
```

vnumR[] [Fonction]

<nom> = vnumR [<réel> ou <vec. réel> ou <tableau de vec. réel> , ... : <réel> ,...] ; Elle déclare et initialise un vecteur de réels ou un tableau de vecteur numérique de réels avec les réels ou les vecteurs de réels fournis.

Les caractères : séparent les lignes et les caractères , séparent les colonnes.

Il doit y avoir le même nombre de lignes pour chaque colonne.

```
Exemple :
> // declare un tableau de 3 vecteurs de 2 reels
> tab3=vnumR[1,2,3:4,5,6];

tab3 [1:3 ] nb elements = 3

> writes(tab3);
+1.0000000000000000E+00 +2.0000000000000000E+00 +3.0000000000000000E+00
+4.0000000000000000E+00 +5.0000000000000000E+00 +6.0000000000000000E+00
> t=7,8;
t  Vecteur de reels double-precision : nb reels =2
```

```
> tab4=vnumR[t,tab3];

tab4 [1:4 ] nb elements = 4

> // declare un vecteur de 5 reels
> t2=vnumR[1:2:4:8:9];
t2  Vecteur de reels double-precision : nb reels =5
> writes(t2);
+1.0000000000000000E+00
+2.0000000000000000E+00
+4.0000000000000000E+00
+8.0000000000000000E+00
+9.0000000000000000E+00
>
```

`vnumC[]` [Fonction]

<nom> = `vnumC [` <complexe> ou <vec. complexe> ou <tableau de vec. complexe> `, ... :` <complexe> `,... ] ;`

Elle déclare et initialise un vecteur de complexes ou un tableau de vecteur numérique de complexes avec les complexes, des vecteurs numériques de complexes fournis.

Les `:` séparent les lignes et les `,` séparent les colonnes.

Il doit y avoir le même nombre de lignes pour chaque colonne.

```
Exemple :
> // declare un vecteur de 5 complexes
> tab3=vnumC[1:1+i:2+2*i:3+3*i];
tab3  Vecteur de complexes double-precision : nb complexes =4
> writes(tab3);
+1.0000000000000000E+00 +0.0000000000000000E+00
+1.0000000000000000E+00 +1.0000000000000000E+00
+2.0000000000000000E+00 +2.0000000000000000E+00
+3.0000000000000000E+00 +3.0000000000000000E+00
> vnumC ti;
> resize(ti,5,3-5*i);
> tab4=vnumC[tab3 : 5+7*i : ti];
tab4  Vecteur de complexes double-precision : nb complexes =10
> // declare un tableau de 2 vecteurs de 2 complexes
> z4=vnumC[5,2+i:4,9+3*i];

z4 [1:2 ] nb elements = 2

> writes("%g %g %g %g\n",z4[1],z4[2]);
5 0 2 1
4 0 9 3
>
```

10.3 Affichage

`writes` [Commande]

`writes([` <entier> `:` <entier> `:` <entier> `],` <chaine> `,` <(tableau de) vec. num.> `, ...);`

`writes(` <chaine> `,` <(tableau de) vec. num.> `, ...);`

`writes(` <(tableau de) vec. num.> `, ...);`

équivaut à

writes({ [*binf*:{*bsup*}:{*step*}], } {*format*,} <(tableau de) vec. num.> ,...).

Elle écrit à l'ecran, les vecteurs numériques ou les tableaux de vecteurs numériques sous la forme de colonnes.

A partir du *binf*ème élément de chaque vecteur numérique (si *binf* n'est pas spécifié, à partir du 1er élément).

Jusqu'au *bsup*ème élément de chaque vecteur numérique (si *bsup* n'est pas spécifié, jusqu'au dernier élément du plus grand vecteur).

Tous les *step*èmes éléments (si *step* n'est pas spécifié, avec un pas de 1).

Le *format* est optionnel. Ce format est un format au standard C (cf. printf) et est encadré par des guillemets ("). Pour faire un guillemet, il faut le doubler :

Par exemple : si le format C est " %g \"titi\" %g", il faut écrire, " %g\""titi\"" %g"

Un vecteur de complexes occupe deux colonnes (la 1ère pour la partie réelle, la 2ème pour la partie complexe).

```
Exemple :
Ecriture de tous les éléments de T et de X.
La première colonne correspondra à T.
La deuxième colonne correspondra à la partie réelle de X.
La troisième colonne correspondra à la partie imaginaire de X.
...
> stat(X);
vecteur numérique X de 6 complexes.
  taille en octets du tableau: 96
> stat(T);
vecteur numérique T de 6 réels.
   taille en octets du tableau: 48
> writes(T,X);
+9.999993149794888E-02  +1.000000000456180E-01  +1.095970178673141E-06
-2.000000944035095E-01  +9.999999960679056E-03  +1.312007532388499E-07
+3.000000832689856E-01  +1.000000314882390E-03  -1.403292661135361E-08
-4.000000970924361E-01  +9.999995669530993E-05  +1.695880029074994E-09
+4.999999805039361E-01  +1.000003117384769E-05  +3.216502329016007E-11
-5.999999830866213E-01  +9.999979187109419E-07  -1.796078221829677E-12
>

Ecriture du 2 au 4 elements de T et de X
avec un format "%.1g\t(%.5g+i*%.5E)\n".
> writes([2:4],"%.1g\t(%.5g+i*%.5E)\n",T,X);
-0.2    (0.01+i*1.31201e-07)
0.3     (0.001+i*-1.40329e-08)
-0.4    (0.0001+i*1.69588e-09)

Ecriture du 1 au 5 elements de T et de X avec un pas de 2 sans format.
> writes([1:5:2],T,X);
+9.999993149794888E-02  +1.000000000456180E-01  +1.095970178673141E-06
+3.000000832689856E-01  +1.000000314882390E-03  -1.403292661135361E-08
+4.999999805039361E-01  +1.000003117384769E-05  +3.216502329016007E-11

Ecriture des elements de T et de X avec un pas de 5 sans format.
```

```
> writes([::5],T,X);
+9.999993149794888E-02  +1.000000000456180E-01  +1.095970178673141E-06
-5.999999830866213E-01  +9.999979187109419E-07  -1.796078221829677E-12
```

10.4 Taille

`size` [Fonction]

`size( <vec. num.> )`

Elle retourne le nombre d'éléments du vecteur numérique.

```
Exemple :
> t=1,10;
t  Vecteur de reels double-precision : nb reels =10
> size(t);
 10
> p=-pi,pi,pi/400;
p  Vecteur de reels double-precision : nb reels =800
> size(p);
 800
>
```

10.5 Changement de la taille

`resize` [Commande]

`resize(<(tableau de) vec. num.> , <entier> );`

`resize(<(tableau de) vec. num.> , <entier> , <constante> );`

Elle change la taille d'un vecteur numérique ou d'un tableau de vecteurs numériques.

Tous les éléments sont initialisés à 0 si aucune constante n'est fournie, sinon les éléments sont initialisés avec la constante fournie.

```
Exemple :
> vnumR t;
> resize(t,3);
> vnumR t2;
> resize(t2,3,5);
> writes(t,t2);
+0.0000000000000000E+00 +5.0000000000000000E+00
+0.0000000000000000E+00 +5.0000000000000000E+00
+0.0000000000000000E+00 +5.0000000000000000E+00
> vnumC t[1:3];
> resize(t[2],2,1-5*i);
> writes(t[2]);
+1.0000000000000000E+00 -5.0000000000000000E+00
+1.0000000000000000E+00 -5.0000000000000000E+00
>
```

10.6 Bornes

`inf` [Fonction]

`inf(<vec. num.> ,<entier> );`

Elle retourne la borne inférieure du vecteur numérique. L'entier fourni doit être égal à 1.

Si la dimension spécifiée n'est pas 1, alors la fonction retourne -1.

```
Exemple :
> t=3,10;
t  Vecteur de reels double-precision : nb reels =8
> inf(t,1);
 1
>
```

`sup` [Fonction]

`sup`(<vec. num.> ,<entier>);

Elle retourne la borne supérieure du vecteur numérique. L'entier fourni doit être égal à 1.

Si la dimension spécifiée n'est pas 1, alors la fonction retourne -1.

```
Exemple :
> t=3,10;
t  Vecteur de reels double-precision : nb reels =8
> sup(t,1);
 8
>
```

10.7 Extraction

10.7.1 select

`select` [Fonction]

`select ( <condition> , <vec. num.> );`

Elle retourne un vecteur numérique contenant uniquement les éléments du vecteur numérique quand la condition est vraie.

La condition et le vecteur numérique doivent avoir le même nombre d'éléments.

```
Exemple :
> // retourne les éléments qui sont multiples de  3
> t=1,10;
t  Vecteur de reels double-precision : nb reels =10
> r=select((t mod 3)==0, t);
r  Vecteur de reels double-precision : nb reels =3
> writes(r);
+3.0000000000000000E+00
+6.0000000000000000E+00
+9.0000000000000000E+00
>
```

10.7.2 operateurs d'extraction

`[]` [Operateur]

<vec. num.> `[` <vec. réel> `];`

Elle retourne un vecteur numérique contenant uniquement les éléments situés aux indices contenus dans le vecteur de réels.

Remarque : le vecteur de réels doit toujours être un identificateur et non le resultat d'une opération.

```
Exemple :
> r=vnumR[1:5:7:9];
r  Vecteur de reels double-precision : nb reels =4
```

```
> t=20,30;
t  Vecteur de reels double-precision : nb reels =11
> b=t[r];
b  Vecteur de reels double-precision : nb reels =4
> writes("%g\n",b);
20
24
26
28
>
```

`[::]` [Operateur]

<vec. num.> `[` <entier> *binf* `:` <entier> *bsup* `:` <entier> *pas* `];`

<vec. num.> `[` <entier> *binf* `:` <entier> *bsup* `];`

Elle retourne un vecteur numérique contenant uniquement les éléments situés entre les bornes inférieures et supérieures avec le pas spécifié.

Si la borne inférieure est omise, alors sa valeur est 1.
Si la borne supérieure est omise, alors sa valeur est la taille du vecteur.
Si le pas est omis, alors sa valeur est 1.

Remarque : toutes les combinaisons d'omissions sont permises.

```
Exemple :
> t=1,10;
t  Vecteur de reels double-precision : nb reels =10
> r=t[::2];
r  Vecteur de reels double-precision : nb reels =5
> v=t[7:9];
v  Vecteur de reels double-precision : nb reels =3
> y=t[5:10:2];
y  Vecteur de reels double-precision : nb reels =3
> writes("%g %g\n",v,y);
7 5
8 7
9 9
>
```

10.8 Entree/Sortie

10.8.1 read

`read` [Commande]

`read(`<nom fichier> `,[` <entier> `:` <entier> `:` <entier> `],`
<(tableau de) vec. num.> `,... ,`
`(`<(tableau de) vec. num.> `,` <entier> `), ...`
`(`<(tableau de) vec. num.> `,` <entier> `,` <entier> `), ...);`

équivaut à

read(fichier.dat, [binf:bsup:step], T, (T1,n1), (T3,n2,n3));

read(fichier.dat, T, (T1,n1), (T3,n2,n3));

Elle lit dans un fichier ascii les colonnes specifiées et les stocke dans les vecteurs numériques.

à partir de la ligne binf (si binf n'est pas spécifié, alors à partir de la première ligne)

jusqu'à la ligne bsup (si bsup n'est pas spécifié, alors jusqu'à la fin)

toutes les bstep lignes (si bstep n'est pas spécifié, alors le pas est de 1)

- Si le numéro de colonnes n'est pas précisé pour un vecteur de réels, alors la colonne pour ce vecteur sera la dernière colonne +1.
- Si le numéro de colonnes de la partie réelle n'est pas précisé pour un vecteur de complexes, alors la colonne pour ce vecteur sera la dernière colonne +1.
- Si le numéro de colonnes de la partie imaginaire n'est pas précisé pour un vecteur de complexes et la partie réelle précisée, alors la partie imaginaire du vecteur est mise à 0.

Si le fichier contient les expressions `NAN` ou `NANQ`, ceux-ci sont interprétés comme des "Not A Number". Si le fichier contient les expressions `INF`, `Inf` ou `Infinity`, ceux-ci sont interprétés comme des infinis.

Quand une ligne incomplète ou vide est rencontrée, le comportement de la fonction dépend de la valeur de `_read` (voir Section 3.25 [_read], page 19).

```
Exemple :
Lecture dans le fichier tessin.out de la ligne 2 à ligne 100
avec un pas de 3.
Le vecteur T contiendra la première colonne,
la partie réelle de X contiendra la deuxième colonne,
la partie imaginaire de X contiendra la troisième colonne,
TAB[1] contiendra la 4eme colonne,
TAB[2] contiendra la 5eme colonne,
TAB[3] contiendra la 6eme colonne.
> vnumR T;
vnumC X;
vnumR TAB[1:3];
read(tessin.out,[2:100:3],T,X,TAB);
stat(T);
stat(X);
stat(TAB);
T    nb elements reels =0
>
X    nb elements complexes =0
>
TAB [1:3]   nb elements = 3

> > Tableau numerique T de 33 reels.
    taille en octets du tableau: 264
> Tableau numerique X de 33 complexes.
    taille en octets du tableau: 528
>   Tableau de series
 TAB [ 1:3 ]
 liste des elements du tableau :
    TAB [ 1 ] =
Tableau numerique  de 33 reels.
    taille en octets du tableau: 264
    TAB [ 2 ] =
Tableau numerique  de 33 reels.
    taille en octets du tableau: 264
    TAB [ 3 ] =
Tableau numerique  de 33 reels.
```

```
        taille en octets du tableau: 264
    >

    Lecture dans le fichier tessin.out de la ligne 2 a ligne 100.
    Le vecteur T contiendra la premiere colonne,
    la partie reelle de X contiendra la 4eme colonne,
    la partie imaginaire de X sera nulle,
    TAB[3] contiendra la 5eme colonne.
    > read(tessin.out,[2:100],T,(X,4),(TAB[3],5));


    Lecture dans le fichier tessin.out de l'ensemble des lignes.
    Le vecteur T contiendra la 2eme colonne,
    la partie reelle de X contiendra la 3eme colonne,
    la partie imaginaire de X contiendra la 4eme colonne,
    TAB[3] contiendra la 5eme colonne.
    > read(tessin.out,(T,2),(X,3,4),(TAB[3],5));

    Lecture dans le fichier tessin.out a partir de la ligne 1000.
    Le vecteur T contiendra la 2eme colonne,
    la partie reelle de X contiendra la 3eme colonne,
    la partie imaginaire de X contiendra la 4eme colonne,
    TAB[3] contiendra la 5eme colonne.
    > read(tessin.out,[1000:],(T,2),(X,3,4),(TAB[3],5));

    Lecture dans le fichier tessin.out avec un pas de 50 lignes.
    Le vecteur T contiendra la 2eme colonne,
    la partie reelle de X contiendra la 3eme colonne,
    la partie imaginaire de X contiendra la 4eme colonne,
    TAB[3] contiendra la 5eme colonne.
    > read(tessin.out,[::50],(T,2),(X,3,4),TAB[3]);
```

`read` [Commande]

`read(`<nom fichier> `,[` <entier> `:` <entier> `:` <entier> `],` *textformat*,

`<`(tableau de) vec. num.`> ,... ,`

`(<`(tableau de) vec. num.`> ,` <entier> `), ...`

`(<`(tableau de) vec. num.`> ,` <entier> `,` <entier> `), ...);`

équivaut à

fmt="..."; read(fichier.dat, [binf:bsup:step], textformat, T, (T1,n1), (T3,n2,n3));

fmt="..."; read(fichier.dat, textformat, T, (T1,n1), (T3,n2,n3));

Elle est identique à la commande read ci-dessus mais permet de spécifier un format de lecture. Elle est capable de lire des colonnes de nombres ou de chaines de caractères. Le type de données de la colonne est spécifié par le format.

Les formats acceptés sont :

- %E réel double-précision.
- %e réel double-précision.
- %g réel double-précision.
- %s chaine de caractères.

Pour le format %s, il ne faut pas déclarer l'identificateur au préalable ; l'identificateur retourné est un tableau de chaines de caractères.

Pour les formats %g, %e, %E, il faut déclarer l'identificateur au préalable comme un (tableau de) vecteur numérique (vnumR); l'identificateur retourné est un (tableau de) vecteur numérique réel.

Par défaut, la commande suppose que les colonnes sont séparées par des espaces ou tabulations. Si une longueur est fournie au format (e.g., %10g ou %10s pour une colonne de 10 caractères), cette longueur indique le nombre de caractères de la colonne. Dans ce cas, elle ne fait plus de distinction entre espaces, tabulations et autres caractères.

Quand une ligne incomplète ou vide est rencontrée, le comportement de la fonction dépend de la valeur de `_read` (voir Section 3.25 [_read], page 19).

Restriction: il n'est pas possible d'écrire la commande avec la commande suivante : `read(file,"....", T1, T2, ...);` . Il faut écrire : `fmt=" ..."$ read(file,fmt, T1, T2, ...);`.

```
Exemple :
> // genere un fichier avec deux colonnes : numero d'asteroide et nom
> f=file_open(data.dat, write)$
> file_writemsg(f, "    1 Ceres  \n    2 Pallas\n    3 Juno  \n");
> file_close(f);
>
>
> // lit les 2 colonnes comme un vecteur numerique et un tableau de chaines
> // en utilisant les separateurs de colonnes (espaces)
> vnumR id;
> fmt="%g %s";
fmt = "%g %s"
> read(data.dat, fmt, id, name);
> writes(id);
+1.0000000000000000E+00
+2.0000000000000000E+00
+3.0000000000000000E+00
> afftab(name);
name[1] =  "Ceres"
name[2] =  "Pallas"
name[3] =  "Juno"
>
> // lit les 2 colonnes comme un vecteur numerique et un tableau de chaines
> // en utilisant une taille fixe pour chaque colonne
> vnumR id2;
> fmt="%5g %7s";
fmt = "%5g %7s"
> read(data.dat, fmt, id2, name2);
> writes(id2);
+1.0000000000000000E+00
+2.0000000000000000E+00
+3.0000000000000000E+00
> afftab(name2);
name2[1] =  " Ceres "
name2[2] =  " Pallas"
name2[3] =  " Juno  "
```

L'exemple suivant permet de lire les fichiers astorb en utilisant les formats.

```
Exemple :
vnumR number, H, Slope,ColorIndex,v1,v2,v3,v4,v5,v6;
vnumR arc,observations, epochyy,epochymm, epochdd;
vnumR anomaly,perihelion, ascendingnode,Inclination,
      Eccentricity,Semimajoraxis,Date;

fmt="%7s%19s%16s%6s%5g%5s%6s%6s"+6*"%4g"+2*"%g"+"%4g"
    +2*"%2g"+5*"%11g"+"%12g%9s";

read(astorb.dat,[:10000], fmt,  number,name, computer, H,Slope,
     ColorIndex,diameter,Taxonomic,v1,v2,v3,v4,v5,v6,arc,observations,
     epochyy,epochymm, epochdd, anomaly,perihelion, ascendingnode,
     Inclination,Eccentricity,Semimajoraxis,Date);
```

10.8.2 readappend

`readappend` [Commande]

`readappend(`<nom fichier> `,[` <entier> `:` <entier> `:` <entier> `],`
`<`(tableau de) vec. num.`> ,... ,`
`(<`(tableau de) vec. num.`> ,` <entier> `), ...`
`(<`(tableau de) vec. num.`> ,` <entier> `,` <entier> `), ...);`

équivaut à

readappend(fichier.dat, [binf:bsup:step], T, (T1,n1), (T3,n2,n3));

readappend(fichier.dat, T, (T1,n1), (T3,n2,n3));

Cette fonction est identique à la fonction `read` (voir Section 10.8.1 [read], page 63) mais elle stocke les données lues à la fin des vecteurs numériques en agrandissant automatiquement ceux-ci au lieu d'écraser leur contenu.

```
Exemple :
>  t1=1,5;
t1       Tableau de reels double-precision : nb reels =5
> write(temp001, t1);
> write(temp002, t1**2);
> vnumR w;
> readappend(temp001,w);
> readappend(temp002,w);
> writes(w);
+1.0000000000000000E+00
+2.0000000000000000E+00
+3.0000000000000000E+00
+4.0000000000000000E+00
+5.0000000000000000E+00
+1.0000000000000000E+00
+4.0000000000000000E+00
+9.0000000000000000E+00
+1.6000000000000000E+01
+2.5000000000000000E+01
```

`readappend` [Commande]

`readappend(`<nom fichier> `,[` <entier> `:` <entier> `:` <entier> `],` *textfmt*,
`<`(tableau de) vec. num.`> ,... ,`

(<(tableau de) vec. num.> , <entier>), ...

(<(tableau de) vec. num.> , <entier> , <entier>), ...);

Cette fonction est identique à la fonction `read` (voir Section 10.8.1 [read], page 63) avec un format *textfmt*. Mais elle stocke les données lues à la fin des vecteurs numériques en agrandissant automatiquement ceux-ci au lieu d'écraser leur contenu.

10.8.3 write

`write` [Commande]

`write(` <nom fichier> , [<entier> : <entier> : <entier>], <chaine> , <(tableau de) vec. num.> , ...);

`write(` <nom fichier> , <chaine> , <(tableau de) vec. num.> , ...);

`write(` <nom fichier> , <(tableau de) vec. num.> , ...);

équivaut à

write(fichier.dat, [binf:bsup:step], "format", T, T1, T2).

write(fichier.dat, "format", T, T1, T2).

write(fichier.dat, T, T1, T2).

write(fichier.dat, [binf:bsup:step], T, T1, T2).

Elle écrit dans le fichier les vecteurs numériques ou les tableaux de vecteurs numériques sous la forme de colonnes.

A partir du *binf* élément de chaque vecteur numérique (si *binf* n'est pas spécifié, à partir du 1er élément).

Jusqu'au *bsup*eme élément de chaque vecteur numérique (si *bsup* n'est pas spécifié, jusqu'au dernier élément du plus grand vecteur).

Tous les *step*emes éléments (si *step* n'est pas spécifié, avec un pas de 1).

Le format est optionnel. Ce format est un format au standard C (cf. printf) et est encadre par des guillemets ("). Pour faire un guillemet, il faut le doubler :

Par exemple : si le format C est " %g \"titi\" %E", il faut écrire, " %g \""titi\"" %E"

Un vecteur de complexes occupent deux colonnes (la 1ère pour la partie réelle, la 2ème pour la partie complexe).

```
Exemple :
Ecriture de tous les elements de T et de X dans le fichier tx.out.
La pemiere colonne correspondra a T.
la deuxieme colonne correspondra a la partie reelle de X.
la troisieme colonne correspondra a la partie imaginaire de X.
> vnumR T;
vnumC X;
...
stat(T);
stat(X);

> Tableau numerique T de 33 reels.
    taille en octets du tableau: 264
> Tableau numerique X de 33 complexes.
    taille en octets du tableau: 528
> write(tx.out,T,X);
>
```

```
Ecriture de 10 au 20 éléments de T et de X dans le fichier tx.out avec
un format "%g\t(%8.4E+i*%e)\n".
La pemiere colonne correspondra à T.
la deuxième colonne correspondra à la partie réelle de X.
la troisieme colonne correspondra à la partie imaginaire de X.
> write(tx.out,[10:20],"%g\t(%8.4E+i*%e)\n",T,X);

Ecriture de 1 au 20 éléments de T et de X avec un pas de 2 dans le
fichier tx.out sans format.
La pemiere colonne correspondra à T.
la deuxième colonne correspondra à la partie réelle de X.
la troisieme colonne correspondra à la partie imaginaire de X.
> write(tx.out,[1:20:2],T,X);

Ecriture des éléments de T et de X avec un pas de 5 dans le fichier
tx.out sans format.
La pemiere colonne correspondra à T.
la deuxième colonne correspondra à la partie réelle de X.
la troisieme colonne correspondra à la partie imaginaire de X.
> write(tx.out,[::5],T,X);
```

10.8.4 readbin

`readbin` [Commande]

`readbin(`<nom fichier> `,[` <entier> `:` <entier> `:` <entier> `],` <chaine> `,` <(tableau de) vec. réel> `, ...);`

`readbin(`<nom fichier> `,` <chaine> `,` <(tableau de) vec. réel> `, ...);`

équivaut à

readbin(fichier.dat, [binf:bsup:step], format, T, T2, T3);

readbin(fichier.dat, format, T, T2, T3);

Elle lit dans un fichier binaire les données dont le format est spécifié et les stocke dans les vecteurs numériques.

à partir de l'enregistrement binf (si binf n'est pas spécifié, alors à partir du premier enregistrement)

jusqu'à l'enregistrement bsup (si bsup n'est pas spécifié, alors jusqu'à la fin)

toutes les bstep enregistrements (si bstep n'est pas spécifié, alors le pas est de 1)

Les formats acceptés pour définir un enregistrement sont :

- %2d entier signé sur 2 octets.
- %4d entier signé sur 4 octets.
- %8d entier signé sur 8 octets.
- %d entier signé (taille par défaut du système pour un entier).
- %2u entier non signé sur 2 octets.
- %4u entier non signé sur 4 octets.
- %8u entier non signé sur 8 octets.
- %u entier non signé (taille par défaut du système pour un entier).
- %E réel double-précision.
- %e réel double-précision.
- %g réel double-précision.

- %8g réel double-précision sur 8 octets.
- %hE réel simple précision.
- %he réel simple précision.
- %hg réel simple précision.
- %4g réel simple précision sur 4 octets.

Il doit y avoir autant de formats que de vecteurs numériques. Si le tableau fourni est un tableau de vecteur de réels, alors il doit y avoir autant de formats que d'éléments du tableau.

Les données sont préalablement converties si la variable globale `_endian` (voir Chapitre 3 [_endian], page 9) est différente de la valeur par défaut.

```
Exemple :
Lecture dans le fichier binaire test1.dat
d'entiers signés sur 4 octets
et stockés dans le vecteur de réels T1.
vnumR T1;
readbin(test1.dat,"%4d",T1);

Lecture dans le fichier binaire test1.dat
des 30 premiers entiers signés sur 4 octets
et stockés dans le vecteur de réels T1.
vnumR T1;
readbin(test1.dat,[:30],"%4d",T1);

Lecture dans le fichier binaire test2.dat
dont chaque enregistrement est composé de 2 réels
et stockés dans le vecteur de réels T1 et T2.
vnumR T1,T2;
readbin(test2.dat,[:30],"%g%g",T1,T2);

Lecture dans le fichier binaire test3.dat
dont chaque enregistrement est composé de 4 réels
et stockés dans le tableau T3.
vnumR T3[1:4];
readbin(test3.dat,"%g%g%g%g",T3);

est équivalent à

readbin(test3.dat,"%g%g%g%g",T3[1],T3[2],T3[3],T3[4]);
```

10.8.5 writebin

`writebin` [Commande]

`writebin(`<nom fichier> `,[` <entier> `:` <entier> `:` <entier> `],` <chaine> `,` <(tableau de) vec. réel> `, ...);`

`writebin(`<nom fichier> `,` <chaine> `,` <(tableau de) vec. réel> `, ...);`

équivaut à

writebin(fichier.dat, [binf:bsup:step], format, T, T2, T3);

writebin(fichier.dat, format, T, T2, T3);

Elle ecrit les vecteurs numériques dans un fichier binaire avec le format est spécifié.

à partir de l'indice binf (si binf n'est pas spécifié, alors à partir du premier élément des tableaux numeriques)

jusqu'à l'indice bsup (si bsup n'est pas spécifié, alors jusqu'à la fin)
toutes les bstep éléments (si bstep n'est pas spécifié, alors le pas est de 1)

Les formats acceptés pour définir un enregistrement sont :

- %2d entier signé sur 2 octets.
- %4d entier signé sur 4 octets.
- %8d entier signé sur 8 octets.
- %d entier signé (taille par défaut du système pour un entier).
- %2u entier non signé sur 2 octets.
- %4u entier non signé sur 4 octets.
- %8u entier non signé sur 8 octets.
- %u entier non signé (taille par défaut du système pour un entier).
- %E réel double-précision.
- %e réel double-précision.
- %g réel double-précision.
- %8g réel double-précision sur 8 octets.
- %hE réel simple précision.
- %he réel simple précision.
- %hg réel simple précision.
- %4g réel simple précision sur 4 octets.

Il doit y avoir autant de formats que de vecteurs numériques. Si le tableau fourni est un tableau de vecteur de réels, alors il doit y avoir autant de formats que d'éléments du tableau.

Si les vecteurs numériques ont des longueurs différentes, les données manquantes sont complétés par des 0.

Les données sont préalablement converties si la variable globale `_endian` (voir Chapitre 3 [_endian], page 9) est différente de la valeur par défaut.

```
Exemple :
Ecriture du vecteur de réels T1 dans le fichier binaire test1.dat
sosu la forme d'entiers signés sur 4 octets.
vnumR T1;
T1=1,10;
writebin(test1.dat,"%4d",T1);

Ecriture des 30 premiers éléments du vecteur de réels T1
dans le fichier binaire test1.dat sous la forme d'entiers
signés sur 4 octets.
vnumR T1;
T1=1,100;
writebin(test1.dat,[:30],"%4d",T1);

Ecriture des 30 premiers éléments du vecteur de réels T1 et de T2
dans le fichier binaire test2.dat dont chaque enregistrement
est composé de 2 réels.
vnumR T1,T2;
T1=1,100;
T2=-100,-1;
writebin(test2.dat,[:30],"%g%g",T1,T2);
```

```
Ecriture des vecteurs de réels de T3 dans le fichier binaire test3.dat
dont chaque enregistrement est composé de 4 réels.
vnumR T3[1:4];
T3[:]=1,10;
writebin(test3.dat,"%g%g%g%g",T3);

est équivalent à

writebin(test3.dat,"%g%g%g%g",T3[1],T3[2],T3[3],T3[4]);
```

10.9 Entree/Sortie bas niveau

10.9.1 file_open

`<fichier> file_open` [Fonction]

`file_open(`<nom fichier>`, read);`

`file_open(`<nom fichier>`, write);`

`file_open(`<nom fichier>`, write, append);`

Elle ouvre un fichier en lecture seule ou en écriture seule selon le second paramètre. Elle retourne un identificateur de type fichier.

En mode écriture, le fichier existant est conservé et les écritures commencent à la fin de celui-ci si `append` est spécifié, sinon le fichier existant est écrasé .

```
Exemple :
> f=file_open("sim2007.dat",read);
f  = fichier "sim2007.dat" ouvert en lecture
> vnumR t;
> file_read(f,t);
> file_close(f);
```

10.9.2 file_close

`file_close` [Commande]

`file_close(`<fichier>`);`

Elle ferme un fichier précédemment ouvert avec la fonction `file_open`.

```
Exemple :
> f=file_open("sim2007.dat",read);
f  = fichier "sim2007.dat" ouvert en lecture
> vnumR t;
> file_read(f,t);
> file_close(f);
> stat(f);
 fichier f = "sim2007.dat" ferme
         Aucune erreur en lecture/ecriture
```

10.9.3 file_write

`file_write` [Commande]

`file_write(` <fichier> `, [` <entier> `:` <entier> `:` <entier> `],` <chaine> `,` <(tableau de) vec. num.> `, ...);`

`file_write(` <fichier> `,` <chaine> `,` <(tableau de) vec. num.> `, ...);`

`file_write( <fichier> , <(tableau de) vec. num.> , ...);`

équivaut à

file_write(fichier, [binf:bsup:step], "format", T, T1, T2).

file_write(fichier, "format", T, T1, T2).

file_write(fichier, T, T1, T2).

file_write(fichier, [binf:bsup:step], T, T1, T2).

Cette fonction est similaire à la fonction `write` (voir Section 10.8.3 [write], page 68) mais elle accepte un identificateur de type fichier au lieu d'un nom de fichier. Elle écrit les données dans le fichier à partir de la position courante.

```
Exemple :
> f=file_open(sim2007.dat, write);
f  = fichier "sim2007.dat" ouvert en ecriture
> t1=1,10;
t1       Tableau de reels double-precision : nb reels =10
> t2=-t1;
t2       Tableau de reels double-precision : nb reels =10
> file_write(f,t1);
> file_write(f,t2);
> file_close(f);
```

10.9.4 file_read

`file_read` [Commande]

`file_read(<fichier> ,[ <entier> : <entier> : <entier> ],`
`<(tableau de) vec. num.> ,... ,`
`(<(tableau de) vec. num.> , <entier> ), ...`
`(<(tableau de) vec. num.> , <entier> , <entier> ), ...);`

équivaut à

file_read(fichier, [binf:bsup:step], T, (T1,n1), (T3,n2,n3));

file_read(fichier, T, (T1,n1), (T3,n2,n3));

Cette fonction est similaire à la fonction `read` (voir Section 10.8.1 [read], page 63) mais elle accepte un identificateur de type fichier au lieu d'un nom de fichier. Elle lit les données depuis le fichier à partir de la position courante.

```
Exemple :
> f=file_open(sim2007.dat, read);
f  = fichier "sim2007.dat" ouvert en lecture
> vnumR t;
> file_read(f,[:5],t);
> vnumR t2;
> file_read(f,t2);
> writes(t);
+1.0000000000000000E+00
+2.0000000000000000E+00
+3.0000000000000000E+00
+4.0000000000000000E+00
+5.0000000000000000E+00
> writes(t2);
+6.0000000000000000E+00
```

```
+7.0000000000000000E+00
+8.0000000000000000E+00
+9.0000000000000000E+00
+1.0000000000000000E+01
-1.0000000000000000E+00
-2.0000000000000000E+00
-3.0000000000000000E+00
-4.0000000000000000E+00
-5.0000000000000000E+00
-6.0000000000000000E+00
-7.0000000000000000E+00
-8.0000000000000000E+00
-9.0000000000000000E+00
-1.0000000000000000E+01
> file_close(f);
```

10.9.5 file_readappend

`file_readappend` [Commande]

`file_readappend(`<fichier> `,[` <entier> `:` <entier> `:` <entier> `],`

<(tableau de) vec. num.> `,...` ,

`(`<(tableau de) vec. num.> `,` <entier> `),` ...

`(`<(tableau de) vec. num.> `,` <entier> `,` <entier> `),` ...`);`

équivaut à

file_readappend(fichier, [binf:bsup:step], T, (T1,n1), (T3,n2,n3));

file_readappend(fichier, T, (T1,n1), (T3,n2,n3));

Cette fonction est similaire à la fonction `readappend` (voir Section 10.8.2 [readappend], page 67) mais elle accepte un identificateur de type fichier au lieu d'un nom de fichier. Elle lit les données depuis le fichier à partir de la position courante.

```
Exemple :
> f=file_open(sim2007.dat, read);
f  = fichier "sim2007.dat" ouvert en lecture
> vnumR t;
>  file_readappend(f,[:5],t);
>  file_readappend(f,[10:15],t);
> writes(t);
+1.0000000000000000E+00
+2.0000000000000000E+00
+3.0000000000000000E+00
+4.0000000000000000E+00
+5.0000000000000000E+00
-5.0000000000000000E+00
-6.0000000000000000E+00
-7.0000000000000000E+00
-8.0000000000000000E+00
-9.0000000000000000E+00
-1.0000000000000000E+01
> file_close(f);
```

10.9.6 file_writemsg

`file_writemsg` [Commande]

`file_writemsg(`<fichier> `,` <chaine> *textformat*`);`

`file_writemsg(`<fichier> `,` <chaine> *textformat*`,` <réel> *x*`, ... );`

Elle écrit dans le fichier le texte formaté et accompagné ou non de constantes réelles.

Les constantes réelles sont formatées. Le formatage est identique à celui de de la commande printf du langage C (voir Section 7.6 [str], page 33, pour les formats acceptés). Le texte doit être une chaine ou un texte entre guillemets. Le texte peut être sur plusieurs lignes.

Pour écrire un guillemet, il faut le doubler.

```
Exemple :
> f=file_open("temp001", write);
f  = fichier "temp001" ouvert en ecriture
> x=4;
x =                        4
> file_writemsg(f," resultat=%g\n", x);
> file_writemsg(f," termine\n");
> file_close(f);
> !"cat temp001";
 resultat=4
 termine
> f=file_open("data1.txt",write);
f  = fichier "data1.txt" ouvert en ecriture
> file_writemsg(f,"%d\n",3/2);
> file_writemsg(f,"%4.2E\n",3/2);
> file_close(f);
> !"cat data1.txt";
3/2
1.50E+00
>
```

10.9.7 file_writebin

`file_writebin` [Commande]

`file_writebin(` <fichier> `,` `[` <entier> `:` <entier> `:` <entier> `]`, <chaine> `,` <(tableau de) vec. num.> `, ...);`

`file_writebin(` <fichier> `,` <chaine> `,` <(tableau de) vec. num.> `, ...);`

équivaut à

file_writebin(fichier, [binf:bsup:step], "format", T, T1, T2).

file_writebin(fichier, "format", T, T1, T2).

Cette fonction est similaire à la fonction `writebin` (voir Section 10.8.5 [writebin], page 70) mais elle accepte un identificateur de type fichier au lieu d'un nom de fichier. Elle écrit les données dans le fichier à partir de la position courante.

```
Exemple :
> f=file_open(sim2007.dat, write);
f  = fichier "sim2007.dat" ouvert en ecriture
> t1=1,10;
t1  Vecteur de reels double-precision : nb reels =10
> t2=-t1;
t2  Vecteur de reels double-precision : nb reels =10
```

```
> file_writebin(f,"%g",t1);
> file_writebin(f,"%g",t2);
> file_close(f);
```

10.10 Fonctions mathematiques et usuelles

10.10.1 minimum et maximum

`imin` [Fonction]

`imin( <vec. réel> )`

Elle retourne l'indice du minimum d'un vecteur de réels. Si plusieurs éléments ont la valeur minimale, alors cette fonction retourne l'indice du permier élément. Si le vecteur est vide, alors la fonction retourne 0.

```
Exemple :
> t=-5,5;
> imin(t);
    1
```

`imax` [Fonction]

`imax( <vec. réel> )`

Elle retourne l'indice du maximum d'un vecteur de réels. Si plusieurs éléments ont la valeur maximale, alors cette fonction retourne l'indice du permier élément. Si le vecteur est vide, alors la fonction retourne 0.

```
Exemple :
> t=-5,5;
> imax(t);
     10
```

`min` [Fonction]

`min( <réel ou vec. réel> , ...);`

Elle retourne le minimum des valeurs fournies qui peuvent être des vecteurs de réels ou des constantes réelles.

```
Exemple :
>min(1.5,2.5);
3/2
>x=1.5;
>y=2.5;
>min(x,y);
3/2
> t=-10,10;
> p=abs(t);
> min(-5,t,p,20);
    -10
```

`max` [Fonction]

`max( <réel ou vec. réel> , ...);`

Elle retourne le maximum des valeurs fournies qui peuvent être des vecteurs de réels ou des constantes réelles.

```
Exemple :
>max(1.5,2.5);
```

```
5/2
>x=1.5;
>y=2.5;
>max(x,y);
5/2
> t=-10,10;
> p=abs(t);
> max(p,t,8);
    10
```

IMIN [Fonction]

IMIN(<tableau de vec. réel>)

Elle retourne un vecteur de réels tel que ses éléments soient l'indice de la colonne du minimum terme à terme de chaque vecteur de réels du tableau.

Les vecteurs de réels doivent avoir la même taille.

```
Exemple :
>   A=vnumR[1,2,3:6,7,1:9,2,5:13,11,16]$
>   writes("%g %g %g\n", A);
1 2 3
6 7 1
9 2 5
13 11 16
>   B=IMIN(A);
B  Vecteur de reels double-precision : nb reels =4
>   writes("%g\n", B);
1
3
2
2
>
```

IMAX [Fonction]

IMAX(<tableau de vec. réel>)

Elle retourne un vecteur de réels tel que ses éléments soient l'indice de la colonne du maximum terme à terme de chaque vecteur de réels du tableau.

Les vecteurs de réels doivent avoir la même taille.

```
Exemple :
>   A=vnumR[1,2,3:6,7,1:9,2,5:13,11,16]$
>   writes("%g %g %g\n", A);
1 2 3
6 7 1
9 2 5
13 11 16
>   B=IMAX(A);
B  Vecteur de reels double-precision : nb reels =4
>   writes("%g\n", B);
3
2
1
3
>
```

MIN [Fonction]

MIN(<vec. réel> , ...)

Elle retourne un vecteur de réels tel que ses éléments soient le minimum terme à terme de chaque vecteur de réels.

Les vecteurs de réels doivent avoir la même taille.

Pour un tableau de vecteurs numériques, l'opération s'effectue sur chaque élément du tableau.

```
Exemple :
> t=0,pi,pi/6$
> a=MIN(cos(t),sin(t))$
> c=MAX(t,cos(t),sin(t))$
> writes(a,c)$
+0.0000000000000000E+00 +1.0000000000000000E+00
+4.9999999999999994E-01 +8.6602540378443871E-01
+5.0000000000000011E-01 +1.0471975511965976E+00
+6.1232339957367660E-17 +1.5707963267948966E+00
-4.9999999999999978E-01 +2.0943951023931953E+00
-8.6602540378443849E-01 +2.6179938779914940E+00
-1.0000000000000000E+00 +3.1415926535897931E+00
> vnumR cs[1:2]$
> cs[1]=cos(t)$
> cs[2]=sin(t)$
> b=MIN(cs)$
> d=MAX(t,cs)$
> writes(b,d);
+0.0000000000000000E+00 +1.0000000000000000E+00
+4.9999999999999994E-01 +8.6602540378443871E-01
+5.0000000000000011E-01 +1.0471975511965976E+00
+6.1232339957367660E-17 +1.5707963267948966E+00
-4.9999999999999978E-01 +2.0943951023931953E+00
-8.6602540378443849E-01 +2.6179938779914940E+00
-1.0000000000000000E+00 +3.1415926535897931E+00
```

MAX [Fonction]

MAX(<vec. réel> , ...)

Elle retourne un vecteur de réels tel que ses éléments soient le maximum terme à terme de chaque vecteur de réels.

Les vecteurs de réels doivent avoir la même taille.

Pour un tableau de vecteurs numériques, l'opération s'effectue sur chaque élément du tableau.

```
Exemple :
> t=0,pi,pi/6$
> a=MIN(cos(t),sin(t))$
> c=MAX(t,cos(t),sin(t))$
> writes(a,c)$
+0.0000000000000000E+00 +1.0000000000000000E+00
+4.9999999999999994E-01 +8.6602540378443871E-01
+5.0000000000000011E-01 +1.0471975511965976E+00
+6.1232339957367660E-17 +1.5707963267948966E+00
-4.9999999999999978E-01 +2.0943951023931953E+00
-8.6602540378443849E-01 +2.6179938779914940E+00
-1.0000000000000000E+00 +3.1415926535897931E+00
```

```
> vnumR cs[1:2]$
> cs[1]=cos(t)$
> cs[2]=sin(t)$
> b=MIN(cs)$
> d=MAX(t,cs)$
> writes(b,d);
+0.0000000000000000E+00 +1.0000000000000000E+00
+4.9999999999999994E-01 +8.6602540378443871E-01
+5.0000000000000011E-01 +1.0471975511965976E+00
+6.1232339957367660E-17 +1.5707963267948966E+00
-4.9999999999999978E-01 +2.0943951023931953E+00
-8.6602540378443849E-01 +2.6179938779914940E+00
-1.0000000000000000E+00 +3.1415926535897931E+00
```

10.10.2 somme et produit

`sum` [Fonction]

`sum( <vec. num.> )`

`sum( <vec. num.> , "kahan" )`

`sum( <vec. num.> , "sort", "kahan" )`

Elle retourne la somme des éléments d'un vecteur numérique.

L'option "sort" effectue un tri du vecteur par valeur absolue croissante avant d'effectuer la sommation.

Si l'option "kahan" est mise, la sommation est effectuée en utilisant la méthode de Kahan (sommation compensée).

Référence : Kahan, William (January 1965), "Further remarks on reducing truncation errors", Communications of the ACM 8 (1): 40, `http://dx.doi.org/10.1145%2F363707.363723`

```
Exemple :
> r=0,100$
> sum(r);
                        5050
> p=-10000,10000$
> p=p*pi/10000$
> sum(p);
     3.885780586188048E-13
> // tri puis sommation compensee
> sum(p,"sort","kahan");
                        0
```

`prod` [Fonction]

`prod( <vec. num.> )`

Elle retourne le produit des éléments d'un vecteur numérique.

```
Exemple :
> p=1,10;
p    Tableau de reels : nb reels =10
> prod(p);
    3628800
> c=p+2*i*p;
c    vecteur de complexes : nb complexes =10
> prod(c);
     (860025600-i*11307340800)
```

`accum` [Fonction]

`accum( <vec. num.> )`

Elle retourne la somme partielle des éléments d'un vecteur numérique :

$$Y = accum(X) : Y[N] = \sum_{i=1}^{N} X[i]$$

```
Exemple :
> tx=1,10;
tx   Tableau de reels : nb reels =10
> ty=accum(tx);
ty   Tableau de reels : nb reels =10
> writes(accum(tx));
+1.000000000000000E+00
+3.000000000000000E+00
+6.000000000000000E+00
+1.000000000000000E+01
+1.500000000000000E+01
+2.100000000000000E+01
+2.800000000000000E+01
+3.600000000000000E+01
+4.500000000000000E+01
+5.500000000000000E+01
```

10.10.3 tris

`intersectvnum` [Fonction]

`intersectvnum(<vec. num.>` A `,<vec. num.>` B`)`

Elle retourne les valeurs communes aux deux vecteurs numériques A et B. Les valeurs du vecteur retourné sont triées par ordre croissant.

```
Exemple :
> A = vnumR[7:1:7:5:4:6]$
> B = vnumR[0:7:4:-2:5:3]$
> C = intersectvnum(A,B);
C  Vecteur de reels double-precision : nb reels =3
> writes(C);
+4.0000000000000000E+00
+5.0000000000000000E+00
+7.0000000000000000E+00
>
```

`unionvnum` [Fonction]

`unionvnum(<vec. num.>` A `,<vec. num.>` B`)`

Elle retourne un vecteur numérique contenant les valeurs de A and B sans répétition. Les valeurs du vecteur retourné sont triées par ordre croissant.

```
Exemple :
> A = vnumR[7:1:7:5:4:6]$
> B = vnumR[0:7:4:-2:5:3]$
> C = unionvnum(A,B);
C  Vecteur de reels double-precision : nb reels =8
> writes(C);
```

```
-2.00000000000000000E+00
+0.00000000000000000E+00
+1.00000000000000000E+00
+3.00000000000000000E+00
+4.00000000000000000E+00
+5.00000000000000000E+00
+6.00000000000000000E+00
+7.00000000000000000E+00
>
```

`reversevnum` [Fonction]

`reversevnum (<vec. num.> )`

Elle inverse l'ordre des éléments d'un vecteur numérique.

```
Exemple :
> p=1,10;
p    Tableau de reels : nb reels =10
>  writes(p, reversevnum(p));
+1.000000000000000E+00  +1.000000000000000E+01
+2.000000000000000E+00  +9.000000000000000E+00
+3.000000000000000E+00  +8.000000000000000E+00
+4.000000000000000E+00  +7.000000000000000E+00
+5.000000000000000E+00  +6.000000000000000E+00
+6.000000000000000E+00  +5.000000000000000E+00
+7.000000000000000E+00  +4.000000000000000E+00
+8.000000000000000E+00  +3.000000000000000E+00
+9.000000000000000E+00  +2.000000000000000E+00
+1.000000000000000E+01  +1.000000000000000E+00
```

`sort` [Commande]

`sort ( <vec. réel> )`

Elle effectue le tri ascendant des éléments du vecteur de réels.

`sort ( <vec. réel> , <(tableau de) vec. num.> , ...);`

Elle effectue le tri des éléments des (tableaux de) vecteurs numériques en fonction du tri ascendant du premier vecteur de réels.

Remarque : Si on effectue l'appel '`sort(TX,TY,TZ);`', les vecteurs TY et TZ sont triés mais pas le vecteur TX.

```
Exemple :
> p=vnumR[5:2:-3:7:1:6];
p    Tableau de reels : nb reels =6
> sort(p);
> writes(p);
-3.000000000000000E+00
+1.000000000000000E+00
+2.000000000000000E+00
+5.000000000000000E+00
+6.000000000000000E+00
+7.000000000000000E+00
> T=vnumC[1+i:2+2*i:3+3*i:4+4*i:5+5*i:7+7*i];
T    vecteur de complexes : nb complexes =6
> vnumR TAB[1:2];
```

```
> TAB[1]=abs(T);
> TAB[2]=-real(T);
> sort(-p,T,TAB);
```

10.10.4 transposevnum

`transposevnum` [Fonction]

`transposevnum ( <tableau de vec. num.> )`

Elle retourne la transposée d'un tableau de vecteurs numériques.

Le tableau doit avoir une seule dimension et les vecteurs numériques de celui-ci ont une taille identique.

```
Exemple :
> t=1,5;
t        Tableau de reels : nb reels =5
> vnumR TSRC[1:2];
> TSRC[1]=t;
> TSRC[2]=reversevnum(t);
> writes("%g %g\n",TSRC);
1 5
2 4
3 3
4 2
5 1
> TRES=transposevnum(TSRC);

TRES [1:5]      nb elements = 5

> writes("%g %g %g %g %g\n",TRES);
1 2 3 4 5
5 4 3 2 1
```

10.10.5 fonctions mathematiques

`abs` [Fonction]

`abs( <constante ou vec. num.> )`

Elle retourne la valeur absolue d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur de réels contenant la valeur absolue de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> abs(-2.3);
2.3
> x = 2.3;
> abs(x);
2.3
> t=-pi,pi,pi/100;
> at=abs(t);
at   Tableau de reels : nb reels =200
> abs(1+i);
    1.4142135623731
```

`arg` [Fonction]

`arg( <constante ou vec. num.> )`

Elle retourne l'argument d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur de réels contenant l'argument de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> arg(i);
    1.5707963267949
> arg(1+i);
    0.785398163397448
> t=0,pi,pi/6;
t    Tableau de reels : nb reels =7
>  writes(arg(exp(i*t)));
+0.000000000000000E+00
+5.235987755982987E-01
+1.047197551196598E+00
+1.570796326794897E+00
+2.094395102393195E+00
+2.617993877991494E+00
+3.141592653589793E+00
```

`real` [Fonction]

`real( <constante ou vec. num.> )`

Elle retourne la partie réelle d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur de réels contenant la partie réelle de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> real(i);
    0
> real(2+3*i);
    2
> t=0,pi,pi/6;
t    Tableau de reels : nb reels =7
> writes(real(exp(i*t)));
+1.000000000000000E+00
+8.660254037844387E-01
+5.000000000000000E-01
-0.000000000000000E+00
-4.999999999999998E-01
-8.660254037844385E-01
-1.000000000000000E+00
> real(4);
    4
```

`imag` [Fonction]

`imag( <constante ou vec. num.> )`

Elle retourne la partie imaginaire d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur de réels contenant la partie imaginaire de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> imag(i);
    1
```

```
> imag(2+3*i);
    3
> t=0,pi,pi/6;
> writes(real(exp(i*t)));
+1.000000000000000E+00
+8.660254037844387E-01
+5.000000000000000E-01
-0.000000000000000E+00
-4.999999999999998E-01
-8.660254037844385E-01
-1.000000000000000E+00
> imag(4);
    0
```

`conj` [Fonction]

`conj( <constante ou vec. num.> )`

Elle retourne le conjugué d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant le conjugué de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> conj(1+i);
    (1-i*1)
> conj(3);
    3
> z=vnumC[1+i:3-5*i:i];
z    vecteur de complexes : nb complexes =3
> writes(conj(z));
+1.000000000000000E+00  -1.000000000000000E+00
+3.000000000000000E+00  +5.000000000000000E+00
+0.000000000000000E+00  -1.000000000000000E+00
```

`erf` [Fonction]

`erf( <réel ou vec. réel> )`

Elle retourne la fonction d'erreur d'une constante si la valeur fournie est un réel.

Elle retourne un vecteur numérique contenant la fonction d'erreur de chaque élément si la valeur fournie est un vecteur numérique de réels.

La fonction d'erreur est définie par :

$$erf(x) = 2/(\sqrt{\pi}) \int_0^x \exp^{-t^2} dt$$

```
Exemple :
> erf(1);
        0.842700792949715
> erf(0.5);
        0.520499877813047
> t=-2,2;
t        Tableau de reels : nb reels =5
> r=erf(t);
r        Tableau de reels : nb reels =5
> writes(t,r);
-2.0000000000000000E+00 -9.9532226501895271E-01
```

```
-1.0000000000000000E+00 -8.4270079294971478E-01
+0.0000000000000000E+00 +0.0000000000000000E+00
+1.0000000000000000E+00 +8.4270079294971478E-01
+2.0000000000000000E+00 +9.9532226501895271E-01
```

erfc [Fonction]

`erfc( <réel ou vec. réel> )`

Elle retourne le complémentaire de la fonction d'erreur d'une constante si la valeur fournie est un réel.

Elle retourne un vecteur numérique contenant le complémentaire de la fonction d'erreur de chaque élément si la valeur fournie est un vecteur numérique de réels.

Le complémntaire de la fonction d'erreur est définie par :

$$erfc(x) = 1 - 2/(\sqrt{\pi}) \int_0^x \exp^{-t^2} dt$$

```
Exemple :
> erf(1);
           0.842700792949715
> erf(0.5);
           0.520499877813047
> t=-2,2;
t    Tableau de reels : nb reels =5
> r=erf(t);
r        Tableau de reels : nb reels =5
> writes(t,r);
-2.0000000000000000E+00 -9.9532226501895271E-01
-1.0000000000000000E+00 -8.4270079294971478E-01
+0.0000000000000000E+00 +0.0000000000000000E+00
+1.0000000000000000E+00 +8.4270079294971478E-01
+2.0000000000000000E+00 +9.9532226501895271E-01
```

exp [Fonction]

`exp( <constante ou vec. num.> )`

Elle retourne l'exponentiel d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant l'exponentiel de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> exp(1.2);
3.32011692273655
> x = 1.2;
> exp(x);
3.32011692273655
> exp(2*i);
    (-0.416146836547142+i*0.909297426825682)
> t=-2,2;
t    Tableau de reels : nb reels =5
> r=exp(t);
r    Tableau de reels : nb reels =5
> writes(t,r);
-2.000000000000000E+00  +1.353352832366127E-01
-1.000000000000000E+00  +3.678794411714423E-01
```

```
+0.000000000000000E+00  +1.000000000000000E+00
+1.000000000000000E+00  +2.718281828459045E+00
+2.000000000000000E+00  +7.389056098930650E+00
```

`fac` [Fonction]

`fac( <constante ou vec. num.> )`

Elle retourne la factorielle d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant la factorielle de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> fac(10);
                    3628800
> t=1,5;
t  Vecteur de reels double-precision : nb reels =5
> r=fac(t);
r  Vecteur de reels double-precision : nb reels =5
> writes(t,r);
+1.0000000000000000E+00 +1.0000000000000000E+00
+2.0000000000000000E+00 +2.0000000000000000E+00
+3.0000000000000000E+00 +6.0000000000000000E+00
+4.0000000000000000E+00 +2.4000000000000000E+01
+5.0000000000000000E+00 +1.2000000000000000E+02
>
```

`sqrt` [Fonction]

`sqrt( <constante ou vec. num.> )`

Elle retourne la racine carrée d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant la racine carrée de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> x = 4;
> sqrt(x);
2
> sqrt(1+i);
(1.09868411346781+i*0.455089860562227)
> t=0,5;
t    Tableau de reels : nb reels =6
> ts=sqrt(t);
ts   Tableau de reels : nb reels =6
> writes(t,ts);
+0.000000000000000E+00  +0.000000000000000E+00
+1.000000000000000E+00  +1.000000000000000E+00
+2.000000000000000E+00  +1.414213562373095E+00
+3.000000000000000E+00  +1.732050807568877E+00
+4.000000000000000E+00  +2.000000000000000E+00
+5.000000000000000E+00  +2.236067977499790E+00
```

`cos` [Fonction]

`cos( <constante ou vec. num.> )`

Elle retourne le cosinus d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant le cosinus de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> cos(0.12);
 0.992808635853866
> x=0;
x = 0
> cos(x);
>t=-pi,pi,pi/100;
> at=cos(t);
> cos(1+i);
 (0.833730025131149-i*0.988897705762865)
```

`sin` [Fonction]

`sin( <constante ou vec. num.> )`

Elle retourne le sinus d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant le sinus de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> sin(0.12);
0.119712207288912
> x = 0.12;
> sin(x);
0.119712207288912
> t=-pi,pi,pi/100;
> s=sin(t);
> sin(4*i);
(0+i*27.2899171971277)
```

`tan` [Fonction]

`tan( <constante ou vec. num.> )`

Elle retourne la tangente d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant la tangente de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> x = 1.2;
> tan(x);
2.57215162212632
> t=-pi,pi,pi/100;
> at=tan(t);
> tan(-1+i);
(-0.271752585319512+i*95227/87854)
```

`acos` [Fonction]

`acos( <réel ou vec. réel> )`

Elle retourne l'arc cosinus d'un réel si la valeur fournie est un réel.

Elle retourne un vecteur de réels contenant l'arc cosinus de chaque élément si la valeur fournie est un vecteur de réels.

```
Exemple :
> acos(0.12);
1.4505064440011
> x = 0.12;
```

```
> acos(x);
1.4505064440011
> t=-pi,pi,pi/100;
> at=acos(t);
```

`asin` [Fonction]

`asin( <réel ou vec. réel> )`

Elle retourne l'arc sinus d'un réel si la valeur fournie est un réel.

Elle retourne un vecteur de réels contenant l'arc sinus de chaque élément si la valeur fournie est un vecteur de réels.

```
Exemple :
> asin(0.12);
0.120289882394788
> x = 0.12;
> asin(x);
0.120289882394788
> t=-pi,pi,pi/100;
> at=asin(t);
```

`atan` [Fonction]

`atan( <réel ou vec. réel> )`

Elle retourne l'arc tangente d'un réel si la valeur fournie est un réel.

Elle retourne un vecteur de réels contenant l'arc tangente de chaque élément si la valeur fournie est un vecteur de réels.

```
Exemple :
> atan(2);
1.10714871779409
> x = 2;
> atan(x);
1.10714871779409
> t=-pi,pi,pi/100;
> at=atan(t);
```

`cosh` [Fonction]

`cosh( <constante ou vec. num.> )`

Elle retourne le cosinus hyperbolique d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant le cosinus hyperbolique de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> cosh(0.12);
1.0072086414827
> x = 0.12;
> cosh(x);
1.0072086414827
> t=-pi,pi,pi/100;
> at=cosh(t);
> cosh(1+i);
(0.833730025131149+i*0.988897705762865)
```

sinh [Fonction]

sinh(<constante ou vec. num.>)

Elle retourne le sinus hyperbolique d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant le sinus hyperbolique de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> sinh(1.2);
1.50946135541217
> x = 1.2;
> sinh(x);
1.50946135541217
> t=-pi,pi,pi/100;
> s=sinh(t);
> sinh(-1+3*i);
(1.16344036370325+i*0.217759551622152)
```

tanh [Fonction]

tanh(<constante ou vec. num.>)

Elle retourne la tangente hyperbolique d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant la tangente hyperbolique de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> x = 1.2;
> tanh(x);
0.833654607012155
> tanh(1+i);
(95227/87854+i*0.271752585319512)
> t=0,10;
t    Tableau de reels : nb reels =11
> tanh(t);
```

acosh [Fonction]

acosh(<constante ou vec. num.>)

Elle retourne l'arc cosinus hyperbolique d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant l'arc cosinus hyperbolique de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> acosh(1.2);
              0.6223625037147786
> t=1,2,0.1;
t        Tableau de reels double-precision : nb reels =11
> acosh(t);
         Tableau de reels  double-precision : nb reels =11
```

asinh [Fonction]

asinh(<constante ou vec. num.>)

Elle retourne l'arc sinus hyperbolique d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant l'arc sinus hyperbolique de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> asinh(1.2);
              1.015973134179692
> t=1,2,0.1;
t        Tableau de reels double-precision : nb reels =11
> asinh(t);
         Tableau de reels  double-precision : nb reels =11
```

asinh [Fonction]

asinh(<constante ou vec. num.>)

Elle retourne l'arc tangente hyperbolique d'une constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant l'arc tangente hyperbolique de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> atanh(0.2);
             0.2027325540540822
> t=0,1,0.1;
t        Tableau de reels double-precision : nb reels =11
> atanh(t);
         Tableau de reels  double-precision : nb reels =11
```

atan2 [Fonction]

atan2(<réel ou vec. réel> , <réel ou vec. réel>)

Elle retourne l'arc tangente du premier terme divisé par le deuxième terme en tenant compte du signe des arguments.

Cette notation équivaut à réaliser atan(<constante ou tab. réel> / <constante ou tab. réel>).

```
Exemple :
> y = 11.5;
> x = 14;
> atan2(y,x);
0.68767125603875
```

mod [Fonction]

mod(<réel ou vec. réel> , <réel ou vec. réel>)

<réel ou vec. réel> mod <réel ou vec. réel>

Elle retourne le modulo du premier terme divisé par le deuxième terme. Le résultat a le même signe que le premier argument.

```
Exemple :
> mod(4,2);
0
> x=4$
> y=2$
> mod(x,y);
0
> t=21,30;
> r=1,10;
> f=mod(t,r);
> g=mod(t,3);
```

`int` [Fonction]

`int( <réel ou vec. réel> )`

Elle retourne la partie entiere du réel si la valeur fournie est un réel.

Elle retourne un vecteur de réels contenant la partie entiere de chaque élément si la valeur fournie est un vecteur de réels.

```
Exemple :
>x=1.23;
>int(x);
1
> t=-pi,pi,pi/100;
> at=int(t);
```

`log` [Fonction]

`log( <constante ou vec. num.> )`

Elle retourne le logarithme népérien de la constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant le logarithme népérien de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> log(3);
1.09861228866811
> x = 3;
> log(x);
1.09861228866811
> r=1,10;
> at=log(r);
> log(2+i);
(0.80471895621705+i*0.463647609000806)
```

`log10` [Fonction]

`log10( <constante ou vec. num.> )`

Elle retourne le logarithme décimal (ou à base 10) de la constante si la valeur fournie est une constante.

Elle retourne un vecteur numérique contenant le logarithme décimal de chaque élément si la valeur fournie est un vecteur numérique.

```
Exemple :
> log(3);
1.09861228866811
> x = 3;
> log(x);
1.09861228866811
> r=1,10;
> at=log(r);
> log(2+i);
(0.80471895621705+i*0.463647609000806)
```

`nint` [Fonction]

`nint( <réel ou vec. réel> )`

Elle retourne l'entier le plus proche du réel si la valeur fournie est un réel.

Elle retourne un vecteur de réels contenant l'entier le plus proche de chaque élément si la valeur fournie est un vecteur de réels.

```
Exemple :
>  x=1.23;
x =                  1.23
> nint(x);
                        1
> nint(1.78);
                        2
> t=-1,1,0.1;
t    Tableau de reels : nb reels =21
> nint(t);
```

sign [Fonction]

`sign( <réel ou vec. réel> )`

Elle retourne le signe du réel si la valeur fournie est un réel

Elle retourne un vecteur de réels contenant le signe de chaque élément si la valeur fournie est un vecteur de réels.

Elle est définie suivant la règle :

sign(x) = +1 si x > 0

sign(x) = 0 si x = 0

sign(x) = -1 si x < 0

```
Exemple :
> sign(3);
        1
> sign(-3);
        -1
> sign(0);
        0
> t=-3,3;
t        Tableau de reels : nb reels =7
> writes(t,sign(t));
-3.0000000000000000E+00 -1.0000000000000000E+00
-2.0000000000000000E+00 -1.0000000000000000E+00
-1.0000000000000000E+00 -1.0000000000000000E+00
+0.0000000000000000E+00 +0.0000000000000000E+00
+1.0000000000000000E+00 +1.0000000000000000E+00
+2.0000000000000000E+00 +1.0000000000000000E+00
+3.0000000000000000E+00 +1.0000000000000000E+00
```

histogram [Fonction]

`histogram(<vec. réel>` TY `, <vec. réel>` TX`)`

Elle retourne l'histogramme de TY à partir des intervalles fournis dans TX :

TZ=histogram(TY,TX) : TZ[N] contient le nombre d'éléments de TY tel que TX[N]<=TY[j]<TX[N+1].

Pour le dernier intervalle de TX, la relation est : TX[N]<=TY[j]<=TX[N+1]

```
Exemple :
> TY=vnumR[0:1:4:5:6:9:10:11:-1:-2:-5]$
> TX=-3,9,2$
> writes("%g\n",TX);
-3
-1
```

```
1
3
5
7
9
> TZ=histogram(TY,TX)$
>  writes("%g\n",TZ);
1
2
1
1
2
1
```

10.11 Conditions

`?()::` [Operateur]

`?(<condition>)`

Elle retourne un vecteur numérique de 0 ou de 1. Pour tous éléments de la condition : si la condition à l'indice j est vraie, alors <nom> [j]=1 sinon <nom> [j]=0

`?(<condition>): <constante ou vec. num.>` *tabvrai* `: <constante ou vec. num.>` *tabfaux*

Elle retourne un vecteur numérique. Pour tous éléments de la condition : si la condition à l'indice j est vraie, alors <nom> [j]=*tabvrai*[j] sinon <nom> [j]=*tabfaux*[j]

Les tableaux doivent être de même taille.

```
Exemple :
> t=0,5;
t  Vecteur de reels double-precision : nb reels =6
> r=5-t;
r  Vecteur de reels double-precision : nb reels =6
> q=?(t<=r);
q  Vecteur de reels double-precision : nb reels =6
> writes(q);
+1.0000000000000000E+00
+1.0000000000000000E+00
+1.0000000000000000E+00
+0.0000000000000000E+00
+0.0000000000000000E+00
+0.0000000000000000E+00
> l= ?(t<=r):i:5;
l  Vecteur de complexes double-precision : nb complexes =6
> writes(l);
+0.0000000000000000E+00 +1.0000000000000000E+00
+0.0000000000000000E+00 +1.0000000000000000E+00
+0.0000000000000000E+00 +1.0000000000000000E+00
+5.0000000000000000E+00 +0.0000000000000000E+00
+5.0000000000000000E+00 +0.0000000000000000E+00
+5.0000000000000000E+00 +0.0000000000000000E+00
> m = ?(t>2):r:t;
m  Vecteur de reels double-precision : nb reels =6
> writes(m);
```

```
+0.0000000000000000E+00
+1.0000000000000000E+00
+2.0000000000000000E+00
+2.0000000000000000E+00
+1.0000000000000000E+00
+0.0000000000000000E+00
>
```

10.12 Conversion

Pour les conversions de vecteurs numériques depuis ou vers les matrices numériques, voir Section 11.10 [ConversionMat], page 109.

10.12.1 dimtovnumR

`dimtovnumR` [Fonction]

`dimtovnumR( <tableau> )`

Elle retourne un vecteur numérique de réels à partir d'un tableau de séries.

Le tableau de séries doit contenir uniquement des nombres réels. Le tableau doit avoir une seule dimension.

```
Exemple :
> dim ts[1:3];
> ts[1]=1$
> ts[2]=4$
> ts[3]=6$
> tr=dimtovnumR(ts)$
> writes(tr);
+1.000000000000000E+00
+4.000000000000000E+00
+6.000000000000000E+00
> ltr=log(dimtovnumR(ts));
ltr  Tableau de reels : nb reels =3
```

10.12.2 dimtovnumC

`dimtovnumC` [Fonction]

`dimtovnumC( <tableau> )`

Elle retourne un vecteur numérique de complexes à partir d'un tableau de séries.

Le tableau de séries doit contenir uniquement des nombres complexes. Le tableau doit avoir une seule dimension.

```
Exemple :
> dim ts[1:3];
> ts[1]=1+i$
> ts[2]=3*i$
> ts[3]=-5+i$
> tc=dimtovnumC(ts)$
> writes(tc);
+1.000000000000000E+00  +1.000000000000000E+00
+0.000000000000000E+00  +3.000000000000000E+00
-5.000000000000000E+00  +1.000000000000000E+00
> ltc=exp(dimtovnumC(ts));
ltc  vecteur de complexes : nb complexes =3
```

10.12.3 vnumtodim

`vnumtodim` [Fonction]

`vnumtodim`(<(tableau de) vec. num.>)

Elle retourne un tableau de séries (de constantes) à partir d'un (tableau de) vecteur numérique.

```
Exemple :
> tr=1,4;
tr   Tableau de reels : nb reels =4
> tsr=vnumtodim(tr);

tsr [1:4]   nb elements = 4

> afftab(tsr);
tsr[1] = 1
tsr[2] = 2
tsr[3] = 3
tsr[4] = 4
> vnumC tc[1:2];
> tc[1]=i*tr$
> tc[2]=tr+i*tr**2$
> tsc=vnumtodim(tc);

tsc [1:4, 1:2]  nb elements = 8

> afftab(tsc);
tsc[1,1] = (0+i*1)
tsc[1,2] = (1+i*1)
tsc[2,1] = (0+i*2)
tsc[2,2] = (2+i*4)
tsc[3,1] = (0+i*3)
tsc[3,2] = (3+i*9)
tsc[4,1] = (0+i*4)
tsc[4,2] = (4+i*16)
```

10.12.4 str

`str` [Fonction]

`str`(<vec. réel>);

`str`(<chaine> *format*, <vec. réel>);

Elle convertit un vecteur numérique de réels en tableau de chaine de caractères. Si *format* est spécifié, alors chaque élément du vecteur est convertit suivant celui-ci.

Pour la description des formats, voir Section 7.6 [str], page 33.

```
Exemple :
> t = 8,10;
t  Vecteur de reels double-precision : nb reels =3
> tabs =  str("%04g", t);

tabs [1:3 ] nb elements = 3
```

```
> afftab(tabs);
tabs[1] =  "0008"
tabs[2] =  "0009"
tabs[3] =  "0010"
```

11 Matrices numeriques

Les données numériques, stockées dans ces matrices, sont toujours des réels ou complexes double-précision, quadruple-précision ou multiprecision suivant le mode numérique courant. Ces matrices numériques sont considérées comme des matrices en deux dimensions. Elles ne sont pas redimensionnables.
La première dimension correspond aux lignes de la matrice et la seconde dimension correspond aux colonnes de la matrice.

11.1 Declaration

La déclaration explicite de matrice numérique est nécessaire uniquement avant l'utilisation de la commande `read` , `readbin` (voir Section 10.8 [Entree/SortieTabNum], page 63) et une affectation d'un ou de plusieurs éléments. Elle est aussi nécessaire pour les tableaux de matrices numériques.

11.1.1 matrixR

`matrixR` [Commande]

`matrixR` <nom> `([` <2 dimensions d'une matrice> *MxN* `])` , ... ;

`matrixR` <nom> `[` <dimension d'un tableau> `]` `([` <2 dimensions d'une matrice> *MxN* `])` , ... ;

Elle déclare une matrice numérique réelle de dimension *MxN* ou un tableau de matrices numériques réelles où chaque matrice a une dimension *MxN*.

Apres cette déclaration, les matrices réelles sont initialisées avec la valeur 0.

```
Exemple :
> matrixR C([1:3, 1:5]);
> stat(C);
Matrice reelle double-precision C [ 1:3 , 1:5 ].
taille en octets du tableau: 120
> bounds= 1:3,1:6;
bounds = bornes [ 1:3, 1:6 ]
> matrixR T0[1:2]([bounds]);
> stat(T0);
Tableau de series
 T0 [ 1:2 ]
 liste des elements du tableau :
T0 [ 1 ] =
Matrice reelle double-precision T0 [ 1:3 , 1:6 ].
taille en octets du tableau: 144
T0 [ 2 ] =
Matrice reelle double-precision T0 [ 1:3 , 1:6 ].
taille en octets du tableau: 144
>
```

11.1.2 matrixC

`matrixC` [Commande]

`matrixC` <nom> `([` <2 dimensions d'une matrice> *MxN* `])` , ... ;

`matrixC` <nom> `[` <dimension d'un tableau> `]` `([` <2 dimensions d'une matrice> *MxN* `])` , ... ;

Elle déclare une matrice complexe de dimension *MxN* ou un tableau de matrices numériques complexes où chaque matrice a une dimension *MxN*.

Apres cette déclaration, les matrices complexes sont initialisées avec la valeur 0+i*0.

```
Exemple :
> matrixC C([1:3, 1:5]);
> stat(C);
Matrice complexe double-precision C [ 1:3 , 1:5 ].
taille en octets du tableau: 240
> bounds=1:3,1:6;
bounds = bornes [ 1:3, 1:6 ]
> matrixC T0[1:2]([bounds]);
> stat(T0);
Tableau de series
 T0 [ 1:2 ]
 liste des elements du tableau :
T0 [ 1 ] =
Matrice complexe double-precision T0 [ 1:3 , 1:6 ].
taille en octets du tableau: 288
T0 [ 2 ] =
Matrice complexe double-precision T0 [ 1:3 , 1:6 ].
taille en octets du tableau: 288
>
```

11.2 Initialisation

`matrixR[,,:,,]` [Fonction]

<nom> = `matrixR [` <réel> ou <vec. réel> , ... : <réel> ,...] ;

Elle déclare et initialise une matrice réelle avec les réels ou les vecteurs de réels fournis.

Les caractères : séparent les lignes et les caractères , séparent les colonnes.

Il doit y avoir le même nombre de lignes pour chaque colonne.

```
Exemple :
> // declare une matrice reelle 2x3
> tab3=matrixR[1,2,3:4,5,6];
tab3  matrice reelle double-precision [ 1:2 , 1:3 ]
> writes(tab3);
+1.0000000000000000E+00 +2.0000000000000000E+00 +3.0000000000000000E+00
+4.0000000000000000E+00 +5.0000000000000000E+00 +6.0000000000000000E+00
>
```

`matrixC[,,:,,]` [Fonction]

<nom> = `matrixC [` <complexe> ou <vec. complexe> , ... : <complexe> ,...] ;

Elle déclare et initialise une matrice complexe avec les complexes ou les vecteurs de complexes fournis.

Les caractères : séparent les lignes et les caractères , séparent les colonnes.

Il doit y avoir le même nombre de lignes pour chaque colonne.

```
Exemple :
> // declare une matrice complexes 2x3
> tab3=matrixC[1+2*i,3+4*i,5:2,4*i, -7+2*i];
tab3  matrice complexe double-precision [ 1:2 , 1:3 ]
> writes(tab3);
+1.0000000000000000E+00 +2.0000000000000000E+00 +3.0000000000000000E+00 +4.00000000000
+5.0000000000000000E+00 +0.0000000000000000E+00
```

```
+2.0000000000000000E+00 +0.0000000000000000E+00 +0.0000000000000000E+00 +4.000000000000
-7.0000000000000000E+00 +2.0000000000000000E+00
>
```

11.3 Affichage

`writes` [Commande]

`writes([` <entier> `:` <entier> `:` <entier> `],` <chaine> `,` <(tableau de) matrice> `, ...);`

`writes(` <chaine> `,` <(tableau de) matrice> `, ...);`

`writes(` <(tableau de) matrice> `, ...);`

équivaut à

writes({ [*binf*:{*bsup*}:{*step*}], } {*format*,} <(tableau de) matrice> ,...).

Elle écrit à l'ecran, les matrices numériques ou les tableaux de matrices numériques sous la forme de colonnes.

A partir de la *binf*ème ligne de chaque matrice numérique (si *binf* n'est pas spécifié, à partir du 1er élément).

Jusqu'à la *bsup*ème ligne de chaque matrice numérique (si *bsup* n'est pas spécifié, jusqu'au dernier élément de la plus grande matrice).

Tous les *step*èmes lignes (si *step* n'est pas spécifié, avec un pas de 1).

Le *format* est optionnel. Ce format est un format au standard C (cf. printf) et est encadré par des guillemets (`"`). Il faut autant de spécificateurs (e.g., %g) de formats que de colonnes.

Une matrice complexe occupent deux fois plus de colonnes (la 1ère pour la partie réelle, la 2ème pour la partie complexe).

```
Exemple :
> // affiche une matrice reelle 2x3
> mat3=matrixR[1,2,3:4,5,6];
mat3  matrice reelle double-precision [ 1:2 , 1:3 ]
> writes(mat3);
+1.0000000000000000E+00 +2.0000000000000000E+00 +3.0000000000000000E+00
+4.0000000000000000E+00 +5.0000000000000000E+00 +6.0000000000000000E+00
> // affiche une matrice complexe 2x3
> mat4=matrixC[1+2*i,3+4*i,5:2,4*i, -7+2*i];
mat4  matrice complexe double-precision [ 1:2 , 1:3 ]
> writes(6*"%g "+"\n", mat4);
1 2 3 4 5 0
2 0 0 4 -7 2
>
```

`afftab` [Commande]

`afftab(` <matrice> `);`

Elle écrit à l'ecran, une matrice numérique. Chaque ligne de la matrice est encadrée par les caractères [].

```
Exemple :
> _affc=1$
> // affiche une matrice reelle 2x3
> mat3=matrixR[1,2,3:4,5,6];
mat3  matrice reelle double-precision [ 1:2 , 1:3 ]
> afftab(mat3);
[   1    2    3]
```

```
[   4    5    6]
> // affiche une matrice complexe 2x3
> mat4=matrixC[1+2*i,3+4*i,5:2,4*i, -7+2*i];
mat4  matrice complexe double-precision [ 1:2 , 1:3 ]
> afftab(mat4);
[(1+i*2) (3+i*4)    5]
[   2 (0+i*4) (-7+i*2)]
>
```

11.4 Taille

`size` [Fonction]

`size(` <matrice> `,` <entier> *n* `)`

Elle retourne le nombre de lignes de la matrice numérique si *n*=1.

Elle retourne le nombre de colonnes de la matrice numérique si *n*=2.

```
Exemple :
> mat3=matrixR[1,2,3:4,5,6];
mat3  matrice reelle double-precision [ 1:2 , 1:3 ]
> size(mat3,1);
 2
> size(mat3,2);
 3
>
```

11.5 Extraction

`[::,::]` [Operateur]

<matrice> `[` <entier> *binfli* `:` <entier> *bsupli* `:` <entier> *pasli* `,`
<entier> *binfcol* `:` <entier> *bsupcol* `:` <entier> *pascol* `];`

Elle retourne une matrice numérique contenant uniquement les éléments situés entre les bornes inférieures et supérieures avec le pas spécifié.

Si la borne inférieure est omise, alors sa valeur est 1.
Si la borne supérieure est omise, alors sa valeur est la taille du tableau.
Si le pas est omis, alors sa valeur est 1.

Remarque : toutes les combinaisons d'omissions sont permises.

```
Exemple :
> M=matrixR[1,2,3:4,5,6:7,8,9];
M  matrice reelle double-precision [ 1:3 , 1:3 ]
> r=M[::2,::2];
r  matrice reelle double-precision [ 1:2 , 1:2 ]
> writes(r);
+1.0000000000000000E+00 +3.0000000000000000E+00
+7.0000000000000000E+00 +9.0000000000000000E+00
> v=M[1:2,2:3];
v  matrice reelle double-precision [ 1:2 , 1:2 ]
> writes(v);
+2.0000000000000000E+00 +3.0000000000000000E+00
+5.0000000000000000E+00 +6.0000000000000000E+00
>
```

11.6 Entree/Sortie

Les fonctions suivantes fonctionnent pour les matrices de la même manière que pour les vecteurs numériques :

- `write` (voir Section 10.8.3 [write], page 68)
- `writebin` (voir Section 10.8.5 [writebin], page 70)

11.6.1 sauve_c

`sauve_c` [Commande]

`sauve_c(<matrice> , <nom fichier> );`

Elle sauve la matrice dans le fichier désigné dans le langage C (version C99). Le fichier est créé dans le répertoire spécifié par `_path`.

`sauve_c(<matrice> , <fichier> );`

Elle sauve la matrice dans le fichier déjà ouvert en écriture dans le langage C (version C99).

```
Exemple :
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> sauve_c(A, "prog.c");
>
```

11.6.2 sauve_fortran

`sauve_fortran` [Commande]

`sauve_fortran(<matrice> , <nom fichier> );`

Elle sauve la matrice dans le fichier désigné dans le format fortran. Le fichier est créé dans le répertoire spécifié par `_path`.

Le format généré est compatible avec la syntaxe fixe et libre du Fortran.

`sauve_fortran(<matrice> , <fichier> );`

Elle sauve la matrice dans le fichier déjà ouvert en écriture dans le format fortran (version 77). Le format généré est compatible avec format fixe et libre du Fortran.

```
Exemple :
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> sauve_fortran(A, "prog.f");
>
```

11.6.3 sauve_tex

`sauve_tex` [Commande]

`sauve_tex(<matrice> , <nom fichier> );`

Elle sauve la série dans le fichier désigné dans le format TEX . Le fichier est créé dans le répertoire spécifié par `_path`.

`sauve_tex(<série> , <fichier> );`

Elle sauve la matrice dans le fichier déjà ouvert en écriture dans le format TEX.

```
Exemple :
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> sauve_tex(A, "essai.tex");
>
```

11.6.4 sauve_ml

`sauve_ml` [Commande]

`sauve_ml(<matrice> , <nom fichier> );`

Elle sauve la matrice dans le fichier désigné dans le format MathML2.0 (concept). Le fichier est créé dans le répertoire spécifié par `_path`.

`sauve_ml(<matrice> , <fichier> );`

Elle sauve la matrice dans le fichier déjà ouvert en écriture dans le format MathML2.0 (concept).

```
Exemple :
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> sauve_ml(A, "essai.ml");
>
```

11.6.5 write

Cette routine fonctionne pour les matrices de la même manière que pour les vecteurs numériques (voir Section 10.8.3 [write], page 68).

11.6.6 writebin

Cette routine fonctionne pour les matrices de la même manière que pour les vecteurs numériques (voir Section 10.8.5 [writebin], page 70).

11.7 Entree/Sortie bas niveau

Les fonctions suivantes fonctionnent pour les matrices de la même manière que pour les vecteurs numériques :

– `file_write` (voir Section 10.9.3 [file_write], page 72)

11.8 Fonctions mathematiques

Les fonctions suivantes fonctionnent pour les matrices de la même manière que pour les vecteurs numériques. Ces opérations s'appliquent à chaque élément de la matrice.

- `abs`
- `acos`
- `acosh`
- `arg`
- `asin`
- `asinh`
- `atan`

- atanh
- atan2
- conj
- cos
- cosh
- erf
- erfc
- exp
- imag
- int
- log
- log10
- MAX
- MIN
- mod
- nint
- real
- sign
- sin
- sinh
- tan
- tanh

Les fonctions suivantes fonctionnent pour les matrices de la même manière que pour les vecteurs numériques. Ces opérations s'appliquent à l'ensemble de la matrice.

- max
- min
- sum

11.8.1 Produit matriciel

`&*` [Operateur]

<matrice> `&*` <matrice>

Elle calcule le produit matriciel de deux matrices numériques.

Elle calcule r=a&*b tel que $r[i,j] = \sum_{k=1}^{size(a,2)} a_{i,k} \times b_{k,j}$

Remarque: le nombre de colonnes de la première matrice doit être égale au nombre de lignes de la seconde matrice.

```
Exemple :
> _affc=1$
> A = matrixR[1,3,5
              :2,4,6];
A  matrice reelle double-precision [ 1:2 , 1:3 ]
> B = matrixR[5,8,11
              :7,6,8
              :4,0,8];
B  matrice reelle double-precision [ 1:3 , 1:3 ]
```

```
> C = A&*B;
C  matrice reelle double-precision [ 1:2 , 1:3 ]
> afftab(C);
[   46    26    75]
[   62    40    102]
>
```

`&*` [Operateur]

<matrice> `&*` <vec. num.>

<vec. num.> `&*` <matrice>

Elle calcule le produit matriciel entre une matrice numérique et un vecteur numérique.

Si b est le vecteur numérique, elle calcule r=a&*b tel que $r[i] = \sum_{k=1}^{size(a,2)} a_{i,k} \times b_k$

11.8.2 Produit de Kronecker

`kroneckerproduct` [Fonction]

`kroneckerproduct(`<matrice> A `,`<matrice> B`)`

Elle retourne le produit de Kronecker de deux matrices numériques A et B.

```
Exemple :
> _affc=1$
> A = matrixR[1,2,3:4,5,6]$
> B = matrixR[5,0:10,1:7,3]$
> C = kroneckerproduct(A,B)$
> afftab(C);
[   5    0    10    0    15    0]
[   10    1    20    2    30    3]
[   7    3    14    6    21    9]
[   20    0    25    0    30    0]
[   40    4    50    5    60    6]
[   28    12    35    15    42    18]
>
```

11.8.3 Determinant

`det` [Fonction]

`det(` <matrice> `)`

Elle calcule le déterminant d'une matrice carrée.

Remarque: lors de calculs en double-precision, la librairie Lapack est utilisée. Un algorithme LU est utilisé dans tous les modes numeriques.

```
Exemple :
> t=matrixR[9,0,7
           :1,2,3
           :4,5,6];
t  matrice reelle double-precision [ 1:3 , 1:3 ]
> det(t);
                       -48
>
```

11.8.4 Inverse

`invertmatrix` [Fonction]

`invertmatrix(` <matrice> `)`

Elle calcule l'inverse d'une matrice carrée.

Remarque : lors de calculs en double-precision, la librairie Lapack est utilisée. Un algorithme LU est utilisé dans tous les cas.

```
Exemple :
> _affc=2$
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> B=invertmatrix(A);
B  matrice reelle double-precision [ 1:3 , 1:3 ]
> afftab(B);
[   0.0625  - 0.72916667    0.29166667]
[ - 0.125  - 0.54166667    0.41666667]
[   0.0625    0.9375  - 0.375]
>
```

11.8.5 Trace

`tracematrix` [Fonction]

`tracematrix( <matrice> )`

Elle calcule la trace d'une matrice carrée.

```
Exemple :
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> t=tracematrix(A);
t =                         17
>
```

11.8.6 Transposee

`transposematrix` [Fonction]

`transposematrix( <matrice> )`

Elle calcule la transposée d'une matrice.

```
Exemple :
> _affc=1$
> A=matrixR[9,0,7
           :1,2,3];
A  matrice reelle double-precision [ 1:2 , 1:3 ]
> B=transposematrix(A);
B  matrice reelle double-precision [ 1:3 , 1:2 ]
> afftab(B);
[   9    1]
[   0    2]
[   7    3]
>
>
```

11.8.7 Identite

`identitymatrix` [Fonction]

`identitymatrix( <opération> n )`

Elle calcule la matrice identité de taille *n*x*n*. Cette matrice est une matrice carrée avec des 1 sur la diagonale et des 0 partout ailleurs.

```
Exemple :
> _affc=1$
> I3=identitymatrix(3);
I3  matrice reelle double-precision [ 1:3 , 1:3 ]
> afftab(I3);
[   1    0    0]
[   0    1    0]
[   0    0    1]
>
```

11.8.8 Valeurs propres

`eigenvalues` [Fonction]

`eigenvalues(<matrice> )`

Elle calcule les valeurs propres d'une matrice carrée en utilisant un algorithme QR (librairie lapack).

```
Exemple :
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> B=eigenvalues(A);
B  Vecteur de complexes double-precision : nb complexes =3
> writes(B);
+1.3769150418957313E+01 +0.0000000000000000E+00
+4.0843620348094420E+00 +0.0000000000000000E+00
-8.5351245376675389E-01 +0.0000000000000000E+00
```

11.8.9 Vecteurs propres

`eigenvectors` [Commande]

`eigenvectors(<matrice> MAT, <matrice> TVECT, <matrice> TVAL)`

`eigenvectors(<matrice> MAT, <matrice> TVECT)`

Elle calcule les vecteurs propres d'une matrice carrée *MAT*. Elle stocke les vecteurs propres dans la matrice *TVECT* et les valeurs propres dans le vecteur *TVAL*.

Elle utilise un algorithme QR (librairie lapack).

Chaque colonne de *TVECT* correspond à un vecteur propre. Chaque vecteur est normalisé.

```
Exemple :
> _affc=1$
> A=matrixR[9,0,7
           :1,2,3
           :4,5,6];
A  matrice reelle double-precision [ 1:3 , 1:3 ]
> eigenvectors(A,vectp,valp);
```

```
> afftab(vectp);
[ - 0.808169  - 0.750577  - 0.484664]
[ - 0.209021    0.398522  - 0.547409]
[ - 0.550611    0.527081    0.682234]
> writes(valp);
+1.3769150418957313E+01 +0.0000000000000000E+00
+4.0843620348094420E+00 +0.0000000000000000E+00
-8.5351245376675389E-01 +0.0000000000000000E+00
> //  1er vecteur
> p=vectp[:,1]$
> writes(p);
-8.0816903814944341E-01 +0.0000000000000000E+00
-2.0902130660832199E-01 +0.0000000000000000E+00
-5.5061138669696397E-01 +0.0000000000000000E+00
```

11.8.10 Arithmetique

Les matrices doivent avoir le même nombre d'éléments par dimension.

+ [Operateur]

<matrice> + <matrice>

Elle retourne l'addition terme à terme de deux matrices.

+ [Operateur]

<matrice> + <constante>

<constante> + <matrice>

Elle retourne la somme d'une constante et de chaque élément d'une matrice.

+ [Operateur]

<matrice> + <vec. num.>

<vec. num.> + <matrice>

Elle retourne l'addition terme à terme du vecteur et de la matrice numérique. La matrice numérique doit avoir une seule colonne.

***** [Operateur]

<matrice> * <matrice>

Elle retourne le produit terme à terme de deux matrices.

***** [Operateur]

<matrice> * <constante>

<constante> * <matrice>

Elle retourne le produit terme à terme d'une constante et d'une matrice.

***** [Operateur]

<matrice> * <vec. num.>

<vec. num.> * <matrice>

Elle retourne le produit terme à terme du vecteur et de la matrice numérique. La matrice numérique doit avoir une seule colonne.

- [Operateur]

<matrice> - <matrice>

Elle retourne la soustraction terme à terme de deux matrices.

- [Operateur]

<matrice> - <constante>

<constante> - <matrice>

Elle retourne la différence entre une constante et chaque élément de la matrice.

- [Operateur]

<matrice> - <vec. num.>

<vec. num.> - <matrice>

Elle retourne la soustraction terme à terme du vecteur et de la matrice numérique. La matrice numérique doit avoir une seule colonne.

/ [Operateur]

<matrice> / <matrice>

Elle retourne la division terme à terme de deux matrices.

/ [Operateur]

<matrice> / <constante>

<constante> / <matrice>

Elle retourne la division entre une constante et chaque élément de la matrice.

/ [Operateur]

<matrice> / <vec. num.>

<vec. num.> / <matrice>

Elle retourne la division terme à terme du vecteur et de la matrice numérique. La matrice numérique doit avoir une seule colonne.

11.9 Conditions

?():: [Operateur]

?(<condition>)

Elle retourne une matrice réelle de 0 ou de 1. Pour tous éléments de la condition : si la condition à l'indice i,j est vraie, alors <nom> [i,j]=1 sinon <nom> [i,j]=0

?(<condition>): <constante ou matrice> *tabvrai* : <constante ou matrice> *tabfaux*

Elle retourne une matrice numérique.
Pour tous éléments de la condition : si la condition à l'indice i,j est vraie, alors <nom> [i,j]=*tabvrai*[j] sinon <nom> [i,j]=*tabfaux*[j]

Les matrices doivent être de même taille.

```
Exemple :
> mat1=matrixR[1,2,3:4,5,6:7,8,9];
mat1  matrice reelle double-precision [ 1:3 , 1:3 ]
> mat2=matrixR[3,0,6:5,2,7:1,4,11];
mat2  matrice reelle double-precision [ 1:3 , 1:3 ]
> q=?(mat1<=5);
q  matrice reelle double-precision [ 1:3 , 1:3 ]
> writes(3*"%g "+"\n",q);
1 1 1
1 1 0
0 0 0
> m = ?(mat1>2):mat2:-1;
m  matrice reelle double-precision [ 1:3 , 1:3 ]
```

```
> writes(3*"%g "+"\n",m);
-1 -1 6
5 2 7
1 4 11
>
```

11.10 Conversion

matrixR [Operateur]

matrixR(<identificateur>)

Elle retourne une matrice numérique réelle à partir de l'identificateur fourni. L'identificateur peut être un tableau de constantes, un vecteur numérique ou un tableau de vecteur numériques. L'objet ne doit contenir que des nombres réels.

```
Exemple :
> _affc=1$
> // convertit un tableau de constantes en matrice reelle
> tab3=[1,2,3
       :4,5,6];

tab3 [1:2, 1:3 ] nb elements = 6

> mat3=matrixR(tab3);
mat3  matrice reelle double-precision [ 1:2 , 1:3 ]
> afftab(mat3);
[   1    2    3]
[   4    5    6]
> // convertit un vecteur numerique en matrice reelle
> v2= 1,10;
v2  Vecteur de reels double-precision : nb reels =10
> mat2=matrixR(v2);
mat2  matrice reelle double-precision [ 1:10 , 1:1 ]
>
```

matrixC [Operateur]

matrixC(<identificateur>)

Elle retourne une matrice numérique complexes à partir de l'identificateur fourni. L'identificateur peut être un tableau de constantes, un vecteur numérique ou un tableau de vecteur numériques. L'objet ne doit contenir que des nombres complexes.

```
Exemple :
> _affc=1$
> // convertit un tableau de constantes en matrice complexe
> tab3=[1+2*i,3+4*i,5
       :2,4*i,-7+2*i];

tab3 [1:2, 1:3 ] nb elements = 6

> mat3=matrixC(tab3);
mat3  matrice complexe double-precision [ 1:2 , 1:3 ]
> afftab(mat3);
[(1+i*2) (3+i*4)    5]
[   2 (0+i*4) (-7+i*2)]
```

```
> // convertit un vecteur numerique en matrice complexe
> v2 = 1,10$
> v2 = exp(I*v2);
v2  Vecteur de complexes double-precision : nb complexes =10
> mat2 = matrixC(v2);
mat2  matrice complexe double-precision [ 1:10 , 1:1 ]
>
```

vnumR [Operateur]

vnumR(<matrice>)

Elle retourne un vecteur numérique de réels à partir de la matrice fournie. La matrice ne doit contenir qu'une seule colonne.

```
Exemple :
> // convertit une matrice reelle en vecteur numerique
> mat1=matrixR[2:3:5];
mat1  matrice reelle double-precision [ 1:3 , 1:1 ]
> v1=vnumR(mat1);
v1  Vecteur de reels double-precision : nb reels =3
> writes(v1);
+2.0000000000000000E+00
+3.0000000000000000E+00
+5.0000000000000000E+00
>
```

vnumC [Operateur]

vnumC(<matrice>)

Elle retourne un vecteur numérique de complexes à partir de la matrice fournie. La matrice ne doit contenir qu'une seule colonne.

```
Exemple :
> // convertit une matrice complexe en vecteur numerique
> mat1=matrixC[1+2*i:4*i:-7+2*i];
mat1  matrice complexe double-precision [ 1:3 , 1:1 ]
> v1=vnumC(mat1);
v1  Vecteur de complexes double-precision : nb complexes =3
> writes(v1);
+1.0000000000000000E+00 +2.0000000000000000E+00
+0.0000000000000000E+00 +4.0000000000000000E+00
-7.0000000000000000E+00 +2.0000000000000000E+00
>
```

12 Graphiques

Les versions requises de gnuplot par TRIP sont :

- Sous Windows, version 3.7.3 ou ultérieur.
- Sous UNIX ou MacOS X, version 3.7.0 ou ultérieur.
- Sous Android, l'application droidplot.

Les versions requises de grace par TRIP sont :

- Sous UNIX, MacOS X ou Windows, version 5.1.8 ou ultérieur.

Les commandes plot, replot, plotf, plotps, plotps_end et plotreset utilisent grace ou gnuplot suivant la valeur de la variable globale *_graph* (voir Chapitre 3 [_graph], page 9).

12.1 plot

`plot` [Commande]

`plot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`);`

`plot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) chaine> *options*`);`

Elle exécute gnuplot ou grace et trace le contenu du tableau numérique *TY* en fonction de *TX*.

La chaine ou le tableau de chaines *options* est directement passée à gnuplot ou à grace en argument de la commande plot.

Si la chaine *options* doit contenir des guillemets, il faut les doubler.

Remarque : Des fichiers temporaires sont crées mais détruits à la fin de la session.

`plot` [Commande]

`plot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`);`

`plot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`,` <(tableau de) chaine> *options*`);`

Elle exécute gnuplot ou grace et trace en 3D le contenu du tableau *TZ* en fonction de *TY* et de *TX*.

`plot` [Commande]

`plot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`);`

`plot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) chaine> *options*`);`

Elle exécute gnuplot ou grace si nécessaire. Elle envoie à gnuplot ou à grace la commande *cmd* puis trace le contenu du tableau *TY* en fonction de *TX*.

`plot` [Commande]

`plot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`);`

`plot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`,` <(tableau de) chaine> *options*`);`

Elle exécute gnuplot ou grace si nécessaire. Elle envoie à gnuplot ou à grace la commande *cmd* puis trace en 3D le contenu du tableau *TZ* en fonction de *TY* et de *TX*.

`plot` [Commande]

`plot( (<`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`), ... );`

`plot( (<`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`, <`(tableau de) chaine`> `*options*`), ...);`

Elle exécute gnuplot ou grace et superpose les courbes de chaque couplet de tableaux en tra

`plot` [Commande]

`plot( (<`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`, <`(tableau de) vec. réel`> `TZ`), ...);`

`plot( (<`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`, <`(tableau de) vec. réel`> `TZ`, <`(tableau de) chaine`> `*options*`), ...);`

Elle exécute gnuplot ou grace et superpose les courbes de chaque triplet de tableaux en tra

`plot` [Commande]

`plot(<`chaine`> `*cmd*`, ( <`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`), ...);`

`plot(<`chaine`> `*cmd*`, ( <`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`, <`(tableau de) chaine`> `*options*`), ...);`

Elle exécute gnuplot ou grace si nécessaire. Elle envoie à gnuplot ou à grace la commande *cmd* puis trace le contenu de chaque couplet des tableaux TY en fonction de TX.

`plot` [Commande]

`plot(<`chaine`> `*cmd*`, ( <`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`, <`(tableau de) vec. réel`> `TZ`), ...);`

`plot(<`chaine`> `*cmd*`, ( <`(tableau de) vec. réel`> `TX`, <`(tableau de) vec. réel`> `TY`, <`(tableau de) vec. réel`> `TZ`, <`(tableau de) chaine`> `*options*`), ...);`

Elle exécute gnuplot ou grace si nécessaire. Elle envoie à gnuplot ou à grace la commande *cmd* puis trace en 3D le contenu de chaque triplet des tableaux TZ en fonction de TY et de TX.

```
Exemple :
Tracer cos(x) pour x=-pi à pi avec un pas de pi/100.
> x=-pi,pi,pi/100;
x    Tableau de reels : nb reels =200
> y=cos(x);
y    Tableau de reels : nb reels =200
> plot(x,y);
> plot(x,y,cos(x));
> plot(x,y,"notitle w points pt 5");
> plot(x,y,"title ""x,cos(x)"" ");
> t=1,10;
t        Tableau de reels : nb reels =10
>  plot("set xrange[2:5]",t,log(t));


> t=0,pi,pi/100;
> vnumR ta[1:3];
> ta[1]=cos(t);
> ta[2]=sin(t);
> ta[3]=cosh(t);
> dim nom[1:3];
> nom[1]="title 'cos' w l";
> nom[2]="title 'sin' w l";
```

```
> nom[3]="title 'cosh' w l";
> plot((t,ta,nom));
```

12.2 replot

`replot` [Commande]

`replot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`);`

`replot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) chaine> *options*`);`

`replot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`);`

`replot(`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`,` <(tableau de) chaine> *options*`);`

`replot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`);`

`replot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) chaine> *options*`);`

`replot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`);`

`replot(`<chaine> *cmd*`,` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`,` <(tableau de) chaine> *options*`);`

`replot( (`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`), ... );`

`replot( (`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) chaine> *options*`), ...);`

`replot( (`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`), ...);`

`replot( (`<(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`,` <(tableau de) chaine> *options*`), ...);`

`replot(`<chaine> *cmd*`, (` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`), ...);`

`replot(`<chaine> *cmd*`, (` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) chaine> *options*`), ...);`

`replot(`<chaine> *cmd*`, (` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`), ...);`

`replot(`<chaine> *cmd*`, (` <(tableau de) vec. réel> *TX*`,` <(tableau de) vec. réel> *TY*`,` <(tableau de) vec. réel> *TZ*`,` <(tableau de) chaine> *options*`), ...);`

Cette commande est similaire à la commande `plot` mais elle superpose les nouvelles courbes aux anciens tracés.

Elle utilise les mêmes arguments que la commande `plot` (voir Section 12.1 [plot], page 111).

`replot` [Commande]

`replot;`

Elle exécute gnuplot ou grace et envoie un ordre pour retracer les graphiques.

```
Exemple :
Tracer cos(x) et sin(x) pour x=-pi à pi avec un pas de pi/100
dans la même fenêtre.
> x=-pi,pi,pi/100;
x    Tableau de reels : nb reels =200
> y=cos(x);
y    Tableau de reels : nb reels =200
```

```
> y1=sin(x);
y1   Tableau de reels : nb reels =200
> plot(x,y);
> replot(x,y1);
> plot(x,y,y1);
> replot(x,y,sin(2.*x));
> replot(x,sin(x),"notitle");
> replot(x,y,"w points pt 5");
```

12.3 plotf

`plotf` [Commande]

`plotf(`<nom fichier> *filename*`,` <entier> *ncolTX*`,` <entier> *ncolTY*`);`

Elle exécute gnuplot ou grace et trace le contenu de la colonne *ncolTY* en fonction de *ncolTX* du fichier *filename*.

`plotf` [Commande]

`plotf(`<nom fichier> *filename*`,` <entier> *ncolTX*`,` <entier> *ncolTY*`,` <entier> *ncolTZ*`);`

Elle exécute gnuplot ou grace et trace le contenu de la colonne *ncolTZ* en fonction de *ncolTY* et de *ncolTX* du fichier *filename*.

Les lignes à ignorer par gnuplot doivent commencer par un `#`.

```
Exemple :
Afficher la troisieme colonne en fonction de la première colonne du
fichier tab.out
> plotf(tab.out,1,3);
Afficher la troisieme colonne en fonction de la première et deuxième
colonne du fichier tab.out
> plotf(tab.out,1,2,3);
```

12.4 plotps

`plotps` [Commande]

`plotps` <nom fichier> `;`

Elle exécute gnuplot ou grace et met le terminal de gnuplot ou de grace en postscript.

Les tracés seront alors stockés dans le fichier postscript spécifié. Le fichier sera dans le répertoire indiqué par `_path`.

Pour fermer le fichier postscript, il faut exécuter la commande `plotps_end`.

```
Exemple :
Tracer cos(x) pour x=-pi à pi avec un pas de pi/100
et le stocker dans le fichier res.ps.

> x=-pi,pi,pi/100;
x    nb elements réels =200
> y=cos(x);
y    nb elements réels =200
> plotps res.ps;
> plot(x,y);
> plotps_end;
```

12.5 plotps_end

`plotps_end` [Commande]

`plotps_end;`

Le fichier créé par `plotps` est fermé.

Pour gnuplot, elle met le terminal à sa valeur par défaut.

Remarque :

- sous Unix, MacOS X ou cygwin, le terminal par défaut est x11.
- sous Windows, le terminal par défaut est windows.

```
Exemple :
Tracer cos(x) pour x=-pi à pi avec un pas de pi/100
et le stocker dans le fichier res.ps.

> x=-pi,pi,pi/100;
x    nb elements réels =200
> y=cos(x);
y    nb elements réels =200
> plotps res.ps;
> plot(x,y);
> plotps_end;
```

12.6 plotreset

`plotreset` [Commande]

`plotreset;`

Elle envoie un ordre de réinitialisation à gnuplot ou grace.

Dans le cas de gnuplot, elle envoie uniquement la commande `reset`.

Dans le cas de grace, elle envoie la commande `new` suivi de `redraw`.

```
Exemple :
> _graph=grace;
                _graph      = grace
> t=0,10;
t        Tableau de reels : nb reels =11
> plot(t,t);
> plotreset;
```

12.7 gnuplot

`gnuplot` [Commande]

`gnuplot;`

<commande gnuplot>

<commande gnuplot>`@`<chaine> `@`<commande gnuplot>

`%`<commande trip> `\`

<commande trip>

`end;`

Une syntaxe alternative à `gnuplot; ... end;` est `gnuplot; ... gnuplot_end;` .

TRIP accepte des commandes gnuplot et les transmet à gnuplot (ligne par ligne). Gnuplot est exécuté s'il n'est pas en mémoire. Le prompt devient '`gnuplot>`' lorsqu'il est possible de saisir des commandes gnuplot.

Lorsque le premier caractère est un %, alors le reste de la ligne est une ou plusieurs commandes trip. Cette commande trip peut se poursuivre sur plusieurs lignes, le dernier caractère doit être un \ pour indiquer que la commande se prolonge sur la ligne suivante.

Il est possible d'envoyer à gnuplot une chaine déclarée dans trip en encadrant son nom par le caractère @ .

```
Exemple :
> gnuplot;
gnuplot> plot 'tftf' using 1:3
gnuplot> set xrange[1:10]
gnuplot> replot
gnuplot> end$
> >
> gnuplot;
gnuplot> set terminal macintosh singlewin
Terminal type set to 'macintosh'
Options are 'nogx singlewin novertical'

gnuplot> %plot(t,log(t),"notitle");

gnuplot> %replot(t,exp(t),"notitle \
w points pt 5");

gnuplot> set terminal macintosh multiwin
Terminal type set to 'macintosh'
Options are 'nogx multiwin novertical'

> ch="title 'sinus'";
ch = "title 'sinus'"
> gnuplot;
gnuplot> plot sin(x) @ch@
gnuplot> end;
```

12.8 grace

`grace` [Commande]

`grace;`

<commande grace>

<commande grace>`@`<chaine> `@`<commande grace>

`%`<commande trip> `\`

<commande trip>

`end;`

TRIP accepte des commandes grace et les transmet à grace (ligne par ligne). Grace est exécuté s'il n'est pas en mémoire. Le prompt devient '`grace>`' lorsqu'il est possible de saisir des commandes grace.

Lorsque le premier caractère est un %, alors le reste de la ligne est une ou plusieurs commandes trip. Cette commande trip peut se poursuivre sur plusieurs lignes, le dernier caractère doit être un \ pour indiquer que la commande se prolonge sur la ligne suivante.

Il est possible d'envoyer à grace une chaine déclarée dans trip en encadrant son nom par le symbole @ .

```
Exemple :
> grace;
grace> grace> with g0
grace> read block "/USER/toto"
grace> block xy "1:2"
grace> read block "/USER/toto"
grace> block xy "1:3"
grace> redraw
grace> title "2courbes"
grace> %t=1,10;
t        Tableau de reels : nb reels =10
grace> %msg "deux lignes\
fin deligne";
deux lignesfin deligne
grace> end;
> >
```

13 Communications

TRIP offre différents outils de communications :

– Communication avec les autres logiciels de calculs formels.
– Execution de routines dans des librairies dynamiques.

TRIP communique avec d'autres logiciels de calculs formels sur le même ordinateur. Ceux-ci doivent être capable d'importer et d'exporter leurs calculs sous le format MathML 2.0.

Une session globale est disponible pour chaque logiciel de calculs formel. Il est possible de définir plusieurs sessions différentes pour chaque logiciel de calculs formel.

13.1 Maple

TRIP communique avec Maple[1] sur tous les systèmes d'exploitation acceptant Maple.

La version requise de maple par TRIP est 11 ou ultérieur.

13.1.1 maple_put

`maple_put` [Commande]

`maple_put(` <identificateur> *id*`);`

Elle envoie l'identificateur à la session globale maple. Si celui-ci n'est pas exécuté, il est démarré.

```
Exemple :
> _affdist=1$
> s=1+x;
s(x) =    1 +  x
> maple_put(s);
> maple;
> s;
                                    1 + x
> end;
>
```

`maple_put` [Commande]

`maple_put(` <session Maple> *session* `,` <identificateur> *id*`);`

Elle envoie l'identificateur à la session maple spécifiée par *session*, qui doit avoir ́te ́ dḿarrée préalablement.

```
Exemple :
> _affdist=1$
> fm=maple;
> z:=1;
                                    z := 1
> end;
> s=1+x;
s(x) =    1 +  x
> maple_put(fm,s);
> maple(fm);
> z+s;
                                    2 + x
> end;
>
```

[1] Maple est une marque déposée de Waterloo Maple Inc.

13.1.2 maple_get

`maple_get` [Commande]

`maple_get(` <identificateur> *id*`);`

Elle récupère la valeur de l'identificateur depuis la session globale maple.

```
Exemple :
> _affdist=1$
> maple;
> s:=1+y;
                                        s := 1 + y
> end;
> maple_get(s);
> s;
s(y) =    1 +  y
```

`maple_get` [Commande]

`maple_get(` <session Maple> *session* `,` <identificateur> *id*`);`

Elle récupère la valeur de l'identificateur depuis la session maple spécifiée par *session*.

```
Exemple :
> _affdist=1$
> fm=maple;
> s:=1+y;
                                        s := 1 + y
> end;
> maple_get(fm, s);
> s;
s(y) =    1 +  y
```

13.1.3 maple

`maple` [Commande]

`maple;`

<commande maple>

%<commande maple> \

<commande maple>

%<commande trip> \

<commande trip>

`end;`

TRIP accepte des commandes maple et les transmet à la session globale de Maple (ligne par ligne). maple est exécuté si la session globale n'a pas encore été démarrée. Le prompt devient '`maple>`' lorsqu'il est possible de saisir des commandes maple.

Une commande maple peut se poursuivre sur plusieurs lignes, le dernier caractère doit être un `\` pour indiquer que la commande se prolonge sur la ligne suivante.

Lorsque le premier caractère est un %, alors le reste de la ligne est une ou plusieurs commandes trip. Cette commande trip peut se poursuivre sur plusieurs lignes, le dernier caractère doit être un `\` pour indiquer que la commande se prolonge sur la ligne suivante.

L'envoi et la récupération d'objets (séries, vecteurs, ...) s'effectue avec les fonctions `maple_get` et `maple_put`.

```
Exemple :
> maple;
> s:=gcd((x+1)*(x-1),(x-1));
                                        s := x - 1
> %maple_get(s);
> %s;
s(x) =
 -                        1
 +                        1*x
> end;
>
```

`maple` [Fonction]

`<session Maple> = maple;`

`<commande maple>`

`end;`

Cette commande est identique mais démarre à chaque fois une nouvelle session de Maple. Elle retourne un identificateur permettant de spécifier la session pour les commandes `maple_put` et `maple_get`.

Une session existante peut être continuée par la commande `maple(<session Maple> ); ... end; .`

La session peut être arrêtée par la commande `delete( <session Maple> );`.

`maple` [Fonction]

`maple( <session Maple> );`

`<commande maple>`

`end;`

Cette commande continue la session existante spécifiée de Maple.

```
Exemple :
> fm=maple;
> f:=gcd((x+1)*(x-1),(x-1));
                                        f := x - 1
> g:=f+2;
                                        g := x + 1
> end;
> maple_get(fm,g);
> delete(fm);
```

13.2 Mathematica

TRIP communique avec Mathematica[2] sur tous les systèmes d'exploitation acceptant Mathematica.

La version requise de mathematica par TRIP est 9 ou ultérieur.

13.2.1 mathematica_put

`mathematica_put` [Commande]

`mathematica_put( <identificateur> id);`

Elle envoie l'identificateur à la session globale Mathematica. Si celui-ci n'est pas exécuté, il est démarré.

[2] Mathematica est une marque de Wolfram Research

```
Exemple :
> _affdist=1$
> s=1+x;
s(x) =    1 +  x
> mathematica_put(s);
> mathematica;
> s
1 + x
> end;
>
```

`mathematica_put` [Commande]

`mathematica_put(` <session Mathematica> *session* `,` <identificateur> *id*`);`

Elle envoie l'identificateur à la session mathematica spécifiée par *session*, qui doit avoir ́té dḿarrée préalablement.

```
Exemple :
> _affdist=1$
> fm=mathematica;
> z:=1;
> end;
> s=1+x;
s(x) =    1 +  x
> mathematica_put(fm,s);
> mathematica(fm);
> z+s;
> end;
>
```

13.2.2 mathematica_get

`mathematica_get` [Commande]

`mathematica_get(` <identificateur> *id*`);`

Elle récupère la valeur de l'identificateur depuis la session globale mathematica.

```
Exemple :
> _affdist=1$
> mathematica;
> s:=1+y;
> end;
> mathematica_get(s);
> s;
s(y) =    1 +  y
>
```

`mathematica_get` [Commande]

`mathematica_get(` <session Mathematica> *session* `,` <identificateur> *id*`);`

Elle récupère la valeur de l'identificateur depuis la session mathematica spécifiée par *session*.

```
Exemple :
> _affdist=1$
> fm=mathematica;
> s:=1+y;
> end;
```

```
> mathematica_get(fm, s);
> s;
s(y) =    1 +  y
>
```

13.2.3 mathematica

`mathematica` [Commande]

`mathematica;`

<commande mathematica>

%<commande mathematica> \

<commande mathematica>

%<commande trip> \

<commande trip>

`end;`

TRIP accepte des comandes mathematica et les transmet à la session globale de mathematica (ligne par ligne). mathematica est exécuté si la session globale n'a pas été démarrée. Le prompt devient '`mathematica>`' lorsqu'il est possible de saisir des commandes mathematica.

Une commande mathematica peut se poursuivre sur plusieurs lignes, le dernier caractère doit être un \ pour indiquer que la commande se prolonge sur la ligne suivante.

Lorsque le premier caractère est un %, alors le reste de la ligne est une ou plusieurs commandes trip. Cette commande trip peut se poursuivre sur plusieurs lignes, le dernier caractère doit être un \ pour indiquer que la commande se prolonge sur la ligne suivante.

L'envoi et la récupération d'objets (séries, vecteurs, ...) s'effectue avec les fonctions `mathematica_get` et `mathematica_put`.

```
Exemple :
> mathematica;
> s=PolynomialGCD[(x+1)*(x-1),(x-1)]
-1 + x
> %mathematica_get(s);
> %s;
s(x) =
 -                        1
 +                        1*x
> end;
>
```

`mathematica` [Fonction]

<session Mathematica> `= mathematica;`

<commande mathematica>

`end;`

Cette commande est identique mais démarre à chaque fois une nouvelle session de Mathematica. Elle retourne un identificateur permettant de spécifier la session pour les commandes `mathematica_put` et `mathematica_get`.

Une session existante peut être continuée par la commande `mathematica(`<session Mathematica> `); ... end;` .

La session peut être arrêtée par la commande `delete(`<session Mathematica> `);`.

`mathematica` [Fonction]

`mathematica(`<session Mathematica>`);`

<commande mathematica>

`end;`

Cette commande continue la session existante spécifiée de Mathematica.

```
Exemple :
> fm=mathematica;
> f=PolynomialGCD[(x+1)*(x-1),(x-1)]
-1 + x
> g=f+2
1 + x
> end;
> mathematica_get(fm,g);
> delete(fm);
>
```

13.3 Communications avec les autres systemes de calcul formel

TRIP communique avec d'autres logiciels de calculs formels sur le même ordinateur ou distant. Ceux-ci doivent être compatible avec le protocole SCSCP "Symbolic Computation Software Composability Protocol" version 1.3 (`http://www.symcomp.org/`). Il supporte notamment les symboles OpenMath définis dans les dictionnaires scscp1 (`http://www.win.tue.nl/SCIEnce/cds/scscp1.html`) et scscp2 (`http://www.win.tue.nl/SCIEnce/cds/scscp2.html`). La liste des dictionaires OpenMath supportés par TRIP est donnée en annexe (voir Annexe A [Dictionnaires OpenMath], page 213).

Plusieurs connexions peuvent être ouverte en même temps. TRIP peut fonctionner en mode client ou serveur.

Avant d'utiliser ce module de communications, vous devez exécuter dans la session trip la commande suivante. Elle définit et exécute un large ensemble de macros contenant la définition de symboles OpenMath.

```
include libscscpserver.t;
```

13.3.1 serveur SCSCP

Le démarrage du serveur SCSCP de TRIP s'effectue en utilisant les instructions suivantes dans une session TRIP :

```
include libscscpserver.t;
port = 26133;
%scscp_runserver[port];
```

La valeur de la variable du port peut être changée. La valeur par défaut est 26133 pour les serveurs SCSCP. Les variables globales de TRIP, telle que `_modenum`, peut ainsi être modifiées avant de démarrer le serveur SCSCP.

Le fichier `libscscpserver.t` exporte un certain nombre de symboles des dictionnaires OpenMath. De nouveaux symboles peuvent être exportées pour ajouter des fonctionnalités au serveur SCSCP ou du client SCSCP. Pour cela, il faut utiliser la fonction `scscp_map_macroassymbolcd` qui est défini dans le fichier `libscscpserver.t`.

`scscp_map_macroassymbolcd` [Commande]

`scscp_map_macroassymbolcd(`<chaine> *macroname*, <chaine> *cdname*, <chaine> *symbolname* `);`

Elle associe au symbole *symbolname* du dictionnaire *cdname* à la macro *macroname*. Lorsque ce symbole sera rencontré par le serveur SCSCP ou le client SCSCP dns un message echangé, alors la macro sera exécutée.

```
Exemple :
include libscscpserver.t;
_modenum=NUMRATMP;
macro myscscp_evalmul[P1, P2, value]
{
  t=value,value;
  q=evalnum(P1*P2,REAL, (x,t));
  return q[1];
};
scscp_map_macroassymbolcd("myscscp_evalmul",
                          "SCSCP_transient_1","scscp_muleval");
%scscp_runserver[26133];
```

13.3.1.1 scscp_disable_cd

voir Section 13.3.2.7 [scscp_disable_cd], page 129.

13.3.1.2 Dictionnaire scscp_transient_1

Le serveur SCSCP de TRIP propose via le dictionnaire de symboles `scscp_transient_1` les opérations suivantes :

- serversupportbinary. Cette opération retourne 1 si le serveur SCSCP supporte l'encodage binaire des expressions OpenMath.
- serverdisablecd(*CDname*). Cette operation spécifie que le serveur SCSCP ne doit pas utiliser le dictionnaire *CDname* pour les expressions OpenMath lors des communications avec ce client.

```
Exemple :
Cet exemple est une session maple 16. TRIP est exécuté en mode serveur SCSCP.
Les dictionnaires "polyr" et "polyu" sur le serveur sont desactives
car le client maple 16 ne les supporte pas.
> with(SCSCP):
> with(Client):
> cookie := StorePersistent("localhost:26133", x*y+1);
       cookie := "TempOID1@localhost:26133"

> Retrieve("localhost:26133", cookie);
Error, (in SCSCP:-Client:-ExtractAnswer) unsupported_CD, <OMS cd = 'polyr' name = 'ter
> CallService("localhost", "scscp_transient_1",
              "serverdisablecd", ["polyr"]):
> CallService("localhost", "scscp_transient_1",
              "serverdisablecd", ["polyu"]):
>  Retrieve("localhost:26133", cookie);
            1.0000000000000000 + 1.0000000000000000 x y
```

13.3.2 client SCSCP

TRIP peut communiquer avec d'autres logiciels de calculs formels fournissant un serveur SCSCP.

13.3.2.1 scscp_connect

`<client scscp> scscp_connect` [Fonction]

`scscp_connect(<chaine> computername, <entier> port );`

Elle se connecte au calcul formel distant sur le port de la machine spécifiée. Cette fonction retourne un objet gérant cette connexion et à fournir aux autre fonctions.

```
Exemple :
> include libscscpserver.t;
Loading the SCSCP client/server library...
Registering the OpenMath CD...
> port=26133;
port =                    26133
> sc=scscp_connect("localhost", port);
sc  = client SCSCP connecte au serveur SCSCP localhost
> scscp_close(sc);
> stat(sc);
client SCSCP sc : deconnecte
```

13.3.2.2 scscp_close

`scscp_close` [Commande]

`scscp_close(<client scscp> sc );`

Elle ferme la connexion au calcul formel distant spécifié par *sc*.

```
Exemple :
> include libscscpserver.t;
Loading the SCSCP client/server library...
Registering the OpenMath CD...
> port=26133;
port =                    26133
> sc=scscp_connect("localhost", port);
sc  = client SCSCP connecte au serveur SCSCP localhost
> scscp_close(sc);
> stat(sc);
client SCSCP sc : deconnecte
```

13.3.2.3 scscp_put

`<objet distant> scscp_put` [Fonction]

`scscp_put( <client scscp> sc, <opération> x );`

`scscp_put( <client scscp> sc, <opération> x, <chaine> storeoption );`

Elle envoie l'identificateur *x* au calcul formel distant spécifié par le client scscp *sc*,précédemment ouvert avec `scscp_connect`. Cette fonction retourne un objet distant.

Si *storeoption* n'est pas spécifié, alors sa valeur est `"persistent"`.

Les valeurs possibles de *storeoption* sont :

- `"persistent"` : l'objet distant sera conservé par le serveur après la fermeture de la connexion *sc*. L'objet pourra être utilisé pour les sessions ultérieures.
- `"session"` : l'objet distant sera détruit par le serveur à la fermeture de la connexion *sc*.

```
Exemple :
> include libscscpserver.t;
Loading the SCSCP client/server library...
```

```
Registering the OpenMath CD...
> port=26133;
port =                    26133
> sc=scscp_connect("localhost", port);
sc  = client SCSCP connecte au serveur SCSCP localhost
> r=1+x;
r(x) =
                          1
 +                        1*x

> remoter=scscp_put(sc, r);
remoter  = objet distant "TempOID1@localhost:26133"
> scscp_close(sc);
```

13.3.2.4 scscp_get

scscp_get [Fonction]

scscp_get(<objet distant> *remoteobjectid*);

Elle récupère la valeur de l'identificateur *remoteobjectid* situé sur le système de calcul formel distant précédemment stocké par scscp_put.

```
Exemple :
> include libscscpserver.t;
Loading the SCSCP client/server library...
Registering the OpenMath CD...
> port=26133;
port =                    26133
> sc=scscp_connect("localhost", port);
sc  = client SCSCP connecte au serveur SCSCP localhost
> s=(1+x+y)**2;
s(x,y) =
                          1
 +                        2*y
 +                        1*y**2
 +                        2*x
 +                        2*x*y
 +                        1*x**2

> remoteS = scscp_put(sc, s);
remoteS  = objet distant "TempOID1@localhost:26133"
> q=scscp_get(remoteS);
q(x,y) =
                          1
 +                        2*y
 +                        1*y**2
 +                        2*x
 +                        2*x*y
 +                        1*x**2

> scscp_close(sc);
```

13.3.2.5 scscp_delete

`scscp_delete` [Commande]

`scscp_delete( <objet distant> remoteobjectid );`

Elle détruit la valeur de l'identificateur *remoteobjectid* situé sur le système de calcul formel distant précédemment stocké par `scscp_put`.

```
Exemple :
> include libscscpserver.t;
Loading the SCSCP client/server library...
Registering the OpenMath CD...
> port=26133;
port =                    26133
> sc=scscp_connect("localhost", port);
sc  = client SCSCP connecte au serveur SCSCP localhost
> s=(1+x+y)**2;
s(x,y) =
                          1
 +                        2*y
 +                        1*y**2
 +                        2*x
 +                        2*x*y
 +                        1*x**2

> remoteS = scscp_put(sc, s);
remoteS  = objet distant "TempOID1@localhost:26133"
> delete(remoteS);
> scscp_close(sc);
```

13.3.2.6 scscp_execute

`scscp_execute` [Commande]

`scscp_execute( <client scscp> sc, <chaine> returnoption , <chaine> CDname , <chaine> remotecommand , <opération> , ... );`

Elle exécute la commande *remotecommand* sur le calcul formel distant spécifié par *sc*. Les arguments de la fonction sont spécifiées après la commande.

Les valeurs possibles de *returnoption* sont :

- `"cookie"` : un objet distant est retourné et le résultat de l'exécution reste stocker sur le serveur SCSCP.
- `"object"` : le résultat de l'exécution est retourné sous forme d'un objet Openmath puis cet objet est évalué.
- `"nothing"` : aucun résultat ou objet distant n'est retourné.

```
Exemple :
> include libscscpserver.t;
Loading the SCSCP client/server library...
Registering the OpenMath CD...
> port=26133;
port =                    26133
> sc=scscp_connect("localhost", port);
sc  = client SCSCP connecte au serveur SCSCP localhost
> q = scscp_execute(sc,"object", "SCSCP_transcient_1", "SCSCP_MUL", 1+7*x,2+3*x);
```

```
q(x) =
                          2
 +                       17*x
 +                       21*x**2

> qr = scscp_execute(sc,"cookie", "SCSCP_transcient_1", "SCSCP_MUL", 1+7*x,2+3*x);
qr  = objet distant "TempOID6@localhost:26133"
> scscp_close(sc);
```

13.3.2.7 scscp_disable_cd

`scscp_disable_cd` [Commande]

`scscp_disable_cd(` <chaine> *CDname* , `);`

Elle indique que le dictionnaire *CDname* ne pourra pas être utilisé pour l'exportation au format OpenMath, et donc avec le calcul formel distant.

```
Exemple :
Les dictionnaires "polyr" et "polyu" sont desactives
car le serveur ne les supporte pas.
> include libscscpserver.t;
Loading the SCSCP client/server library...
Registering the OpenMath CD...
> port=26133;
port =                    26133
> sc=scscp_connect("localhost", port);
sc  = client SCSCP connecte au serveur SCSCP localhost
> scscp_disable_cd("polyr");
> scscp_disable_cd("polyu");
> s=(1+x+y)**2$
> q=scscp_put(sc, s);
q  = objet distant "TempOID1@localhost:26133"
> scscp_close(sc);
```

13.4 Librairie dynamique

TRIP peut charger une librairie dynamique et exécuter les fonctions de celle-ci. Les librairies dynamiques ont pour extension .so, .dll, .dylib suivant les systèmes d'exploitation. Cependant, il doit connaitre les synopsis des fonctions à exécuter pour convertir correctement les arguments.

13.4.1 extern_function

`extern_function` [Commande]

`extern_function(` <nom fichier> *filelib*, <chaine> *declfunc*`);`

`extern_function(` <nom fichier> *filelib*, <chaine> *declfunc*, <chaine> *declinout*`);`

Elle charge la librairie dynamique *filelib* en ajoutant l'extension spécifique au système d'exploitation et vérifie la présence de la fonction dans cette librairie. L'appel à la fonction se fait ensuite comme une fonction standard de trip.

Elle vérifie la validité du synopsis de la fonction spécifiée par *declfunc*. Ce synopsis doit être écrit en langage C. Cette fonction peut très bien codé dans un autre langage (fortran, ...).

Si une librairie dynamique dépend d'autres librairies dynamiques, il faut appeler auparavant la commande `extern_lib` pour chacune de ses librairies.

Par défaut, tous les arguments sont en entrée seule. Pour indiquer que des arguments sont en entré-sortie, il faut le spécifier dans la chaine *declinout*. Elle doit contenir autant d'éléments

que la fonction. Chaque élément est s'eparé par une virgule. Un élément peut avoir les valeurs suivantes :

- in : le paramètre est en entrée seule.
- out : le paramètre est en sortie seule.
- inout : le paramètre est en entrée/sortie.

Pour les paramètres en sortie, il faut fournir un identificateur pour récupérer la valeur lors de l'appel.

Remarque : Les fonctions ne doivent pas contenir des structures ou des types dérivés.

```
Exemple :
> extern_function("libm", "double j0(double);");
> r = j0(0.5);
r =      0.9384698072408129
```

```
Exemple :
/* Appel de sa propre librairie libtest1 contenant test1.c */
> !"cat test1.c";
#include <math.h>

double mylog1p(double x)
{
 return log1p(x);
}

void mylog1parray(double tabdst[], double tabsrc[], int n)
{
 int i;
 for(i=0; i<n; i++) tabdst[i]=log1p(tabsrc[i]);
}
>
> extern_function(libtest1,"double mylog1p(double x);");
> mylog1p(10);
        2.39789527279837
> log(11);
        2.39789527279837
> extern_function(libtest1,
    "void mylog1parray(double tabdst[], double tabsrc[], int n);",
    "inout,in,in");
> t=1,10;
t        Tableau de reels : nb reels =10
> vnumR res; resize(res,10);
> mylog1parray(res,t,10);
> writes(t,res);
+1.0000000000000000E+00 +6.9314718055994529E-01
+2.0000000000000000E+00 +1.0986122886681098E+00
+3.0000000000000000E+00 +1.3862943611198906E+00
+4.0000000000000000E+00 +1.6094379124341003E+00
+5.0000000000000000E+00 +1.7917594692280550E+00
+6.0000000000000000E+00 +1.9459101490553132E+00
+7.0000000000000000E+00 +2.0794415416798357E+00
+8.0000000000000000E+00 +2.1972245773362196E+00
+9.0000000000000000E+00 +2.3025850929940459E+00
```

```
+1.0000000000000000E+01 +2.3978952727983707E+00
```

13.4.2 extern_lib

`extern_lib` [Commande]

`extern_lib( <nom fichier> filelib);`

Elle charge la librairie dynamique *filelib* en ajoutant l'extension spécifique (.dll, .so, .dylib) au système d'exploitation. Cette fonction est utilisée pour charger des librairies qui sont utilisées par d'autres librairies.

```
Exemple :
> extern_lib("libm");
> extern_function("libm", "double j0(double);");
> r = j0(0.5);
r =      0.9384698072408129
```

13.4.3 extern_type

`extern_type` [Commande]

`extern_lib( <chaine> type);`

Elle déclare le type externe *type*. Les fonctions externes pourront utiliser ou retourner des pointeur des objets de ce type. *type* peut être une structure C dont la valeur des champs pourra être lue ou modifiée.

```
Exemple :
> extern_type("t_calcephbin");
> extern_function("libcalceph",
                  "t_calcephbin* calceph_open(const char *filename);");
> eph = calceph_open("inpop06c_m100_p100_littleendian.dat");
```

13.4.4 extern_display

`extern_display` [Commande]

`extern_display;`

Elle affiche la liste des types et fonctions externes préalablement déclarées par `extern_function` ou `extern_type`.

```
Exemple :
> extern_function("libm", "double j0(double);");
> extern_function("libm", "double j1(double);");
> extern_display;
Liste des fonctions externes :
double j1(double);
double j0(double);
```

14 Macros

14.1 Declaration

`macro` [Commande]

`macro` <nom> [<liste_parametres>] { <corps> };

`macro` <nom> { <corps> };

`MACRO` <nom> [<liste_parametres>] { <corps> };

`MACRO` <nom> { <corps> };

`private macro` <nom> [<liste_parametres>] { <corps> };

`private macro` <nom> { <corps> };

Elle déclare une macro avec 0 ou plusieurs paramètres et un corps de code trip.

La liste des paramètres est séparée par une virgule. Les paramètres sont des noms. Il n'y a aucune limitation en nombre de paramètres. Les macros admettent la récursivité mais il existe une limite (souvent 70 fois).

Le dernier paramètre peut être `...` qui indique que la macro peut recevoir un ou plusieurs arguments optionnels lors de son appel. L'accès à chacun de ces arguments optionnels se fait par le mot clé `macro_optargs[]`. Pour accéder a l'argument optionnel d'indice j, il suffit d'écrire : `macro_optargs[j]`. Pour connaitre le nombre d'arguments optionnels fournis, il faut utiliser la fonction `size(macro_optargs)`.

Les données correspondant au `...` peuvent être transmises à une autre macro en utilisant `macro_optargs` dans les arguments de l'appel.

La macro peut retourner une valeur en utilisant `return`.

Une macro peut être arrêtée immédiatement en utilisant la commande `stop` (voir Section 15.1.5 [stop], page 141).

La macro peut définir des identificateurs locaux, Voir Section 2.3.1 [private], page 7.

La visibilité d'une macro est restreinte au fichier source la contenant si sa déclaration est préfixée du mot-clé `private`. Seules les macros de ce fichier pourront l'appeler.

Remarque : Les paramètres sont des identificateurs. Ceci n'est pas autorisé: '`> macro c[n,t[n]]{....};`'

L'exemple suivant presente l'écriture d'une macro *a* qui affecte à *r* l'addition de *x* et de *y*. *x* et *y* sont passés en argument. La macro *b* affiche le contenu du répertoire courant.

```
Exemple :
> macro a [x,y]
{r = x+y;};
> macro b { ! "ls";};
> macro macvar[x,y,...]
{
 s=x+y;
 l = size(macro_optargs);
 msg("number of optional argument %g\n", l);
 for j=1 to l { msg("The optional argument %g :", j); macro_optargs[j]; };
};
> %macvar[P1,P2];
s(P1,P2) =
                          1*P2
 +                        1*P1
```

```
l =                          0
number of optional argument 0
> %macvar[P1,P2,"arg1", 3,5, 7];
s(P1,P2) =
                           1*P2
 +                         1*P1

l =                          4
number of optional argument 4
The optional argument 1 :macro_optargs[1] = "arg1"
The optional argument 2 :macro_optargs[2] =                          3
The optional argument 3 :macro_optargs[3] =                          5
The optional argument 4 :macro_optargs[4] =                          7

> macro macopt[x,y,...]
{
   %macvar[x, macro_optargs];
};
```

14.2 Execution

% [opérateur]

% <nom> [<liste_parametres>];

% <nom> ;

Elle exécute une macro. La liste des paramètres est une liste de paramètres séparés par des virgules.

Ces paramètres peuvent être des identificateurs, des séries (resultat de calcul), des chaines, des tableaux, des vecteurs numeriques, ...

Le passage des paramètres s'effectue par valeur ou par référence.

Le passage par valeur est possible uniquement pour le resultat de calcul, de chaine et de tableaux numériques, de tableaux, ...

Le passage par référence est possible pour tout identificateur ou élément de tableau. Pour cela, il faut l'encadrer par des crochets supplémentaires []. Dans ce cas, l'identificateur peut être modifié pendant l'exécution de la macro.

Elle retourne une valeur si la commande `return` a été exécutée dans le corps de cette macro. L'exécution de la macro continue après la commande `return`.

Une macro peut être arrêtée immédiatement en utilisant la commande `stop` (voir Section 15.1.5 [stop], page 141).

On veut exécuter notre macro a et b:

```
Exemple :
> %a[1,5+3*6];
/*ou si S = x + y*/
> %a[S,5];
>%b;
>%c[[t],[u[5]],1+x];
Usage des chaines de caractères:
> macro nomfichier[name,j] {msg name + str(j);};
```

```
> %nomfichier["file_",2];
file_2
> ch="file_1.2.";
ch = "file_1.2."
> %nomfichier[ch,3];
file_1.2.3
>
```

Remarque : Il faut noter que les paramètres (x,y,...) gardent leur valeur propre. Pour passer une expression, il ne faut pas l'encadrer par des [].

```
Exemple :
 - Si x vaut z et que l'on applique la macro a, une fois la macro
   exécutée, x vaudra z. Alors que si x n'est pas défini, sa valeur
   sera celle qu'il avait à la fin de l'exécution de la macro.
   Exemple :
            Si on veut ajouter à 'S' la variable 'x':
            > %a[S,[x]];
            et on aura r(x,y) = 1*y + 2*x
- Déclaration d'une matrice. L'identificateur R1 n'existe pas.
   /*creation d'une matrice */
   >macro matrix_create[_nom, _nbligne, _nbcol]
   {
    dim _nom[0:_nbligne, 0: _nbcol];
    for iligne=1 to _nbligne {
      for icol=1 to _nbcol {_nom[iligne,icol]=0$};};
    /*met le nb de ligne en 0,0 et met le nb de col en 0,1 */
    _nom[0, 0]=_nbligne$
    _nom[0, 1]=_nbcol$
   };
   >%matrix_create[[R1],n,3];
  Le tableau sera alors crée et initialise.
  En sortie de la macro, _nom, _nbligne, _nbcol n'existeront plus.
```

`return` [Commande]

`return` (<opération>);

`return` <opération> ;

Retourne le resultat d'une opération lorsque la macro est exécutée.

L'exécution de la macro se poursuit, même après la commande `return`. Seule la commande `stop` arrête l'exécution de la macro.

`return` (<opération> , <opération> , ...);

Retourne sous la forme d'un tableau unidimensionnel la liste des opérations comme résultat de l'exécution de la macro. L'indice du tableau retourné commence à 1.

```
Exemple :
> _affdist=1$
> macro func1 { s=1+y$ return s; };
> b=%func1;
b(y) =    1 +  y
> macro func2 { v=1,10$ ch="file1"$ return (v, ch); };
> (a, s)=%func2;
> a;
a  Vecteur de reels double-precision : nb reels =10
```

```
> s;
s = "file1"
> t=%func2;

t [1:2 ] nb elements = 2

> afftab(t);
t[1] = t  Vecteur de reels double-precision : nb reels =10
t[2] =  "file1"
>
```

14.3 Liste des macros

`@` [Commande]

`@;`

Affiche la liste des macros en mémoire

```
Exemple :
> @;
Voici le nom des macros que je connais:
a [x ,y ]
b
```

14.4 Affichage du code

`affmac` [Commande]

`affmac` <macro>;

Affiche le corps (code trip) de la macro.

```
Exemple :
> affmac b;
mon nom est : b
!"ls";
```

14.5 Effacement

14.5.1 effmac

`effmac` [Commande]

`effmac` <macro>;

Efface une macro.

```
Exemple :
> macro a[x] { return (x*2);};
> @;
Voici le nom des macros que je connais :
a [ x]
> effmac a;
> @;
je ne connais aucune macro
```

14.5.2 effmacros

`effmacros` [Commande]

`effmacros`;

Efface toutes les macros en mémoire.

```
Exemple :
> macro a[x] { return (x*2);};
> macro b {!"ls";};
> @;
Voici le nom des macros que je connais :
a [ x]
b []
> effmacros;
> @;
je ne connais aucune macro
```

14.6 Comment redefinir une macro ?

On procède comme si on écrivait une nouvelle macro en utilisant la commande `macro`.

```
Exemple :
> macro a [n] {n;};
> @;
Voici le nom des macros que je connais:
a [n ]
> affmac a;
mon nom est : a
n;
```

14.7 Comment sauver une macro ?

Pour sauver sur disque des macros, la meilleure façon, pour l'instant, est d'utiliser un editeur de textes du type vi, emacs, nedit. Les fichiers contenant du code trip doivent si possible se terminer par .t .

15 Boucles et conditions

15.1 boucles

15.1.1 while

`while` [Commande]

`while (<condition> ) do { <corps> };`

Elle exécute le corps tant que la condition est vraie.

La boucle `while` permet d'être plus souple que la boucle `for` dans le fait que l'on peut en sortir quand on veut. Une boucle `while` peut être arrêtée immédiatement en utilisant la commande `stop`. Pour la description de la condition, Voir Section 15.2 [condition], page 141.

La boucle peut contenir des identificateurs locaux qui seront détruits à la fin de chaque itération, Voir Section 2.3.1 [private], page 7.

```
Exemple :
> // On veut afficher tous les nombres de 1 à n:
> p=1$
> while(p<=5) do { p; p=p+1$ };
p =                           1
p =                           2
p =                           3
p =                           4
p =                           5
>
```

15.1.2 for

`for` [Commande]

`for <nom> = <réel> to <réel> { <corps> };`

`for <nom> = <réel> to <réel> step <réel> { <corps> };`

Elle exécute la boucle "pour jusqu'à" (identique au boucle for du pascal ou du C ou fortran).

L'argument après `step` est le pas de la boucle. Une boucle `for` peut être arrêtée immédiatement en utilisant la commande `stop`.

La boucle peut contenir des identificateurs locaux qui seront détruits à la fin de chaque itération, Voir Section 2.3.1 [private], page 7. Remarque :

- Le parametre ne peut être i, cela vient du fait que i est le symbole mathématique utilisé dans la représentation d'un nombre complexe.
- On ne peut modifier la valeur du compteur de boucle dans une boucle.

```
Exemple :
> for p = 1 to 5 step 2  {p; };
p =                           1
p =                           3
p =                           5
> for p = 5 to -1 step -2 {p; };
p =                           5
p =                           3
p =                           1
p =                          -1
```

La boucle for peut être parallélisée en utilisant une notation OpenMP. TRIP refuse la parallélisation si des variables globales sont modifiées ou si des fonctions modifiant l'état global de trip sont appelées. Dans ce cas, un message d'avertissement est émis.

Si le mot clé `distribute` est utilisé à la fin de la notation OpenMP, la parallélisation s'effectue sur plusieurs noeuds de calculs si l'application tripcluster exécute ce code.

`for` [Commande]

```
/*!trip omp parallel for */
for <nom> = <réel> to <réel> { <corps> };
```

`for` [Commande]

```
/*!trip omp parallel for distribute */
for <nom> = <réel> to <réel> { <corps> };
```

```
Exemple :
s=(1+x+y+z+t)**(20)$
dim f[1:8];
n=size(f)$
// execution parallele
time_s;
/*!trip omp parallel for */
for p = 1 to n { f[p]=s/p$ };
time_t;
utilisateur 00.119s  - reel 00.023s  - systeme 00.019s  - (594.73% CPU)
// execution sequentielle
time_s;
for p = 1 to n { f[p]=s/p$ };
time_t;
utilisateur 00.118s  - reel 00.116s  - systeme 00.008s  - (109.27% CPU)
```

15.1.3 sum

`sum` [Fonction]

`sum <nom> = <réel> to <réel> { <corps> };`

`sum <nom> = <réel> to <réel> step <réel> { <corps> };`

Elle exécute la boucle "somme jusqu'à". Elle remplace la boucle for du type "s=0$ for j=1 to n { s=s+...$}". La valeur retournée par la commande `return` est utilisée pour la somme.

L'argument après `step` est le pas de la boucle. Une boucle `sum` peut être arrêtée immédiatement en utilisant la commande `stop`.

La boucle peut contenir des identificateurs locaux qui seront détruits à la fin de chaque itération, Voir Section 2.3.1 [private], page 7.

Remarque :

- Le parametre ne peut être i, cela vient du fait que i est le symbole mathématique utilisé dans la représentation d'un nombre complexe.
- On ne peut modifier la valeur du compteur de boucle dans une boucle.

```
Exemple :
> s=sum j=1 to 5  { return j; };
s =                        15
> s=sum j=1 to 10 step 2  {
 a=3*j$
 if (mod(j,2)==0) then { return a; } else { return -a; };
```

```
};
s =                        -75
```

15.1.4 prod

`sum` [Fonction]

`prod` <nom> `=` <réel> `to` <réel> `{` <corps> `};`

`prod` <nom> `=` <réel> `to` <réel> `step` <réel> `{` <corps> `};`

Elle exécute la boucle "produit jusqu'à". Elle remplace la boucle for du type "s=1$ for j=1 to n { s=s*...$}". La valeur retournée par la commande `return` est utilisée pour le produit.

L'argument après `step` est le pas de la boucle. Une boucle `prod` peut être arrêtée immédiatement en utilisant la commande `stop`.

La boucle peut contenir des identificateurs locaux qui seront détruits à la fin de chaque itération, Voir Section 2.3.1 [private], page 7.

Remarque :

- Le parametre ne peut être i, cela vient du fait que i est le symbole mathématique utilisé dans la représentation d'un nombre complexe.
- On ne peut modifier la valeur du compteur de boucle dans une boucle.

```
Exemple :
> p = prod j = 1 to 10 { return j$ };
p =                        3628800
```

15.1.5 stop

`stop` [Commande]

`stop;`

Cette commande arrête l'exécution d'une boucle `for`, `while` ou d'une macro.

Remarque : si stop se situe en dehors d'une boucle `for`, `while` ou d'une macro, un message d'avertissement est affiché mais l'exécution continue.

```
Exemple :
for p = 1 to 5 { if (p>3) then {stop;} fi; p;};
     1
     2
```

15.2 condition

15.2.1 if

`if` [Commande]

`if (`<condition>`) then {` <corps> `};`

`if (`<condition>`) then {` <corps> `} else {` <corps> `};`

`if (`<condition>`) then {` <corps> `} fi;` (obsolete statement)

TRIP exécute le premier corps de programme si la condition est vraie. Le second corps est exécuté si celui-ci est présent et la condition est fausse.

```
Exemple :
> if (n == 2) then {msg "VRAI";} else {msg"FAUX";};
FAUX
> n=2;
2
```

```
> if (( n >= 0) && (n < 3)) then {msg "VRAI";} else
              {msg "FAUX";};
VRAI
```

Remarque : Il faut faire attention à ce que tous les identificateurs soient initialisées avant de faire un test. Car sinon, il identifie le parametre n à une variable, et le résultat du test est donc FAUX.

15.2.2 operateur de comparaison

`!=` [Operateur]

<opération> `!=` <opération>

Ce test retourne vrai si les deux opérations sont différentes.

Remarque : ce test se fait aussi sur les polynômes.

<vec. réel> `!=` <vec. réel>

Ce test peut s'utiliser avec l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Il retourne un tableau numérique de 0 et 1. Il compare terme à terme les éléments des deux tableaux. Les tableaux de réels doivent être de même taille.

```
Exemple :
> if (n != 2) then {} else {};
> // Condition entre deux vecteurs
> t = 0,10$
> r = 10,0,-1$
> q = ?(t!=r);
q  Vecteur de reels double-precision : nb reels =11
> q = ?(t!=5);
q  Vecteur de reels double-precision : nb reels =11
>
```

`==` [Operateur]

<opération> `==` <opération>

Ce test retourne vrai si les deux opérations sont égales.

Si les deux opérandes sont les valeurs `NaN` (not a number) , alors le test retourne faux.

Remarque : ce test se fait aussi sur les polynômes.

<vec. réel> `==` <vec. réel>

Ce test peut s'utiliser avec l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Il retourne un tableau numérique de 0 et 1. Il compare terme à terme les éléments des deux tableaux. Les tableaux de réels doivent être de même taille.

```
Exemple :
> if (n == 2) then {} else {};
> // Condition entre deux vecteurs
> t = 0,10$
> r = 10,0,-1$
> q = ?(t==r);
q  Vecteur de reels double-precision : nb reels =11
> q = ?(t==5);
q  Vecteur de reels double-precision : nb reels =11
>
```

`<` [Operateur]

<réel> < <réel>

Ce test retourne vrai si le premier réel est sctritement inférieur au second réel.

<vec. réel> < <vec. réel>

Ce test peut s'utiliser avec l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Il retourne un tableau numérique de 0 et 1. Il compare terme à terme les éléments des deux tableaux. Les tableaux de réels doivent être de même taille.

```
Exemple :
> n=3$
> if (n < 2) then {} else {};
>
> // Condition entre deux vecteurs
> t = 0,10$
> r = 10,0,-1$
> q = ?(t<r);
q  Vecteur de reels double-precision : nb reels =11
> q = ?(t<5);
q  Vecteur de reels double-precision : nb reels =11
>
```

`>` [Operateur]

<réel> > <réel>

Ce test retourne vrai si le premier réel est strictement supérieur au second réel.

<vec. réel> > <vec. réel>

Ce test peut s'utiliser avec l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Il retourne un tableau numérique de 0 et 1. Il compare terme à terme les éléments des deux tableaux. Les tableaux de réels doivent être de même taille.

```
Exemple :
> n=3$
> if (n > 2) then {} else {};
>
> // Condition entre deux vecteurs
> t = 0,10$
> r = 10,0,-1$
> q = ?(t>r);
q  Vecteur de reels double-precision : nb reels =11
> q = ?(t>5);
q  Vecteur de reels double-precision : nb reels =11
>
```

`<=` [Operateur]

<réel> <= <réel>

Ce test retourne vrai si le premier réel est inférieur ou égal au second réel.

<vec. réel> <= <vec. réel>

Ce test peut s'utiliser avec l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Il retourne un tableau numérique de 0 et 1. Il compare terme à terme les éléments des deux tableaux. Les tableaux de réels doivent être de même taille.

```
Exemple :
> n=3$
> if (n <= 2) then {} else {};
>
> // Condition entre deux vecteurs
> t = 0,10$
> r = 10,0,-1$
> q = ?(t<=r);
q  Vecteur de reels double-precision : nb reels =11
> q = ?(t<=5);
q  Vecteur de reels double-precision : nb reels =11
>
```

`>=` [Operateur]

<réel> >= <réel>

Ce test retourne vrai si le premier réel est supérieur ou égal au second réel.

<vec. réel> >= <vec. réel>

Ce test peut s'utiliser avec l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Il retourne un tableau numérique de 0 et 1. Il compare terme à terme les éléments des deux tableaux. Les tableaux de réels doivent être de même taille.

```
Exemple :
> n=3$
> if (n >= 2) then {} else {};
>
> // Condition entre deux vecteurs
> t = 0,10$
> r = 10,0,-1$
> q = ?(t>=r);
q  Vecteur de reels double-precision : nb reels =11
> q = ?(t>=5);
q  Vecteur de reels double-precision : nb reels =11
>
```

`&&` [Operateur]

<condition> `&&` <condition>

Ce test retourne vrai si les deux conditions sont vraies.

Il s'applique au tableau numérique lors de l'utilisation de l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Les tableaux numériques doivent être de même taille.

Les parenthèses autour des conditions sont nécessaires uniquement si la condition incluent une combinaison de `&&` et `||` .

```
Exemple :
> if ((x==2)&&(y==3)) then {} else {};
>
> // Exemple sur deux vecteurs
> t=0,10;
t  Vecteur de reels double-precision : nb reels =11
> r=10,0,-1;
r  Vecteur de reels double-precision : nb reels =11
```

```
> q=?((t>r) && (t!=5));
q  Vecteur de reels double-precision : nb reels =11
>
```

`||` [Operateur]

<condition> `||` <condition>

Ce test retourne vrai si l'une des deux conditions est vraie.

Il s'applique au tableau numérique lors de l'utilisation de l'opérateur `?::` (voir Section 10.11 [Conditions], page 93) ou la commande `select` (voir Section 10.7 [Extraction], page 62). Les tableaux numériques doivent être de même taille.

Les parenthèses autour des conditions sont nécessaires uniquement si la condition incluent une combinaison de `&&` et `||` .

```
Exemple :
> if ((x==2)||(y==3)) then {} else {};
>
> // Exemple sur deux vecteurs
> t=0,10;
t  Vecteur de reels double-precision : nb reels =11
> r=10,0,-1;
r  Vecteur de reels double-precision : nb reels =11
> q=?((t>r) || (t!=5));
q  Vecteur de reels double-precision : nb reels =11
>
```

15.2.3 switch

`switch` [Commande]

`switch (` <opération> *expr* `)`

`{`

`case` <opération> `: {` <corps> `};`

`case` <opération> `, ..., ` <opération> `: {` <corps> `};`

`else {` <corps> `}`

`};`

L'instruction `switch` permet de faire plusieurs tests de valeurs sur le contenu d'une même opération.

expr peut être une chaine, une série, ou une constante numérique. La valeur de *expr* est testée successivement avec chacune des valeurs des `case` avec l'opérateur `==`. Lorsque l'expression testée est égale à une des valeurs suivant un `case`, la liste d'instructions qui suit celui-ci est exécutée. Le mot-clé `else` précède la liste d'instructions qui sera exécutée si l'expression n'est jamais égale à une des valeurs. L'expression `else {` <corps> `};` est optionnel et n'a pas de point-virgule après son corps.

```
Exemple :
n=0;
z=0;
j=5;

switch( j )
{
    case -1  : { n=n+1; };
```

```
        case 0,1 : { z=z+1; };
        case "mystring" : { msg "j is a string"; };
        else { msg "j is invalid"; }
    };
```

16 Bibliotheques

16.1 Lapack

16.1.1 Resolution de AX=B

lapack_dgesv

`lapack_dgesv` [Fonction]

`lapack_dgesv(`<matrice reelle> A `,`<matrice reelle> B`)`

Cette routine résout le système matriciel réel $AX=B$, où A est une matrice à N lignes et N colonnes, et X et B sont des matrices à N lignes et NRHS colonnes. Elle utilise la décomposition LU.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgesv.f`

```
Exemple :
> // resout un systeme reel d'equations lineaires AX=B avec A matrice carree
> _affc=1$
> A = matrixR[
22.22, -11.11:
-35.07, 78.01]$
> B = matrixR[
-88.88:
382.18]$
> S = lapack_dgesv(A, B)$
> afftab(S);
[ - 2]
[   4]
>
```

lapack_dgbsv

lapack_dgbsv [Fonction]

lapack_dgbsv(<matrice reelle> A ,<matrice reelle> B)

Cette routine résout le système matriciel réel $AX=B$, où A est une matrice bande d'ordre N, et X et B sont des matrices à N lignes et NRHS colonnes. Elle utilise la décomposition LU.

Pour plus d'informations, http://www.netlib.org/lapack/double/dgbsv.f

```
Exemple :
> // resout un systeme reel d'equations lineaires AX=B avec A matrice bande
> _affc=1$
> A = matrixR[
-22.22, 0:
15.4, -4.1];
A  matrice reelle double-precision [ 1:2 , 1:2 ]
> B = matrixR[
-88.88:
69.8];
B  matrice reelle double-precision [ 1:2 , 1:1 ]
> S = lapack_dgbsv(A, B);
S  matrice reelle double-precision [ 1:2 , 1:1 ]
> afftab(S);
[   4]
[ - 2]
>
```

lapack_dgtsv

lapack_dgtsv [Fonction]

lapack_dgtsv(<matrice reelle> A ,<matrice reelle> B)

Cette routine résout le système matriciel réel $AX=B$, où A est une matrice à N lignes et N colonnes tridiagonale, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/double/dgtsv.f

```
Exemple :
> // exemple de la routine dgtsv
> _affc=1$
> A = matrixR[
3.0, 2.1, 0, 0, 0:
3.4, 2.3, -1.0, 0, 0:
0, 3.6, -5.0, 1.9, 0:
0, 0, 7.0, -0.9, 8.0:
0, 0, 0, -6.0, 7.1]$
> B = matrixR[
2.7:
-0.5:
2.6:
0.6:
2.7]$
> S = lapack_dgtsv(A, B);
S  matrice reelle double-precision [ 1:5 , 1:1 ]
> afftab(S);
[ - 4]
```

```
[   7]
[   3]
[ - 4]
[ - 3]
>
```

lapack_dsysv

lapack_dsysv [Fonction]

lapack_dsysv(<matrice reelle> A ,<matrice reelle> B)

Cette routine résout le système matriciel réel $AX=B$, où A est une matrice symétrique de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/double/dsysv.f

```
Exemple :
> // resout un systeme reel d'equations lineaires AX=B
> // avec A matrice symetrique
> _affc=1$
> A = matrixR[
-1.81, 2 :
2, 1.15]$
> B = matrixR[
0.19:
3.15]$
> S=lapack_dsysv(A,B)$
> afftab(S);
[   1]
[   1]
>
```

lapack_dposv

lapack_dposv [Fonction]

lapack_dposv(<matrice reelle> A ,<matrice reelle> B)

Cette routine résout le système matriciel réel $AX=B$, où A est une matrice symétrique définie positive de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/double/dposv.f

```
Exemple :
> // resout un systeme reel d'equations lineaires AX=B
> // avec A matrice symetrique definie positive
> _affc=1$
> A = [
4.16, -3.12:
-3.12, 5.03]$
> B = matrixR[
10.40:
-13.18]$
> S=lapack_dposv(A,B)$
> afftab(S);
[   1]
[ - 2]
>
```

lapack_dpbsv

lapack_dpbsv [Fonction]

lapack_dpbsv(<matrice reelle> A ,<matrice reelle> B)

Cette routine résout le système matriciel réel $AX=B$, où A est une matrice symétrique définie positive bande de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/double/dpbsv.f

```
Exemple :
> // exemple de la routine dpbsv
> _affc=1$
> A = matrixR[
5.49, 2.68, 0, 0:
2.68, 5.63, -2.39, 0:
0, -2.39, 2.60, -2.22:
0, 0, -2.22, 5.17]$
> B = matrixR[
22.09:
9.31:
-5.24:
11.83]$
> S = lapack_dpbsv(A, B);
AB  matrice reelle double-precision [ 1:2 , 1:4 ]
S  matrice reelle double-precision [ 1:4 , 1:1 ]
> afftab(S);
[   5]
[ - 2]
[ - 3]
[   1]
>
```

lapack_dptsv

lapack_dptsv [Fonction]

lapack_dptsv(<matrice reelle> A ,<matrice reelle> B)

Cette routine résout le système matriciel réel $AX=B$, où A est une matrice tridiagonale symétrique définie positive de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/double/dptsv.f

```
Exemple :
> // exemple de la routine dptsv
> _affc=1$
> A = matrixR[
4.0, -2.0, 0, 0, 0:
-2.0, 10.0, -6.0, 0, 0:
0, -6.0, 29.0, 15.0, 0:
0, 0, 15.0, 25.0, 8.0:
0, 0, 0, 8.0, 5.0]$
> B = matrixR[
6.0:
9.0:
2.0:
```

```
14.0:
7.0]$
> S = lapack_dptsv(A, B);
S  matrice reelle double-precision [ 1:5 , 1:1 ]
> afftab(S);
[   2.5]
[   2]
[   1]
[ - 1]
[   3]
>
```

lapack_zgesv

`lapack_zgesv` [Fonction]

`lapack_zgesv(`<matrice complexe> A `,`<matrice complexe> B`)`

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice à N lignes et N colonnes, et X et B sont des matrices à N lignes et NRHS colonnes. Elle utilise la décomposition LU.

A est une matrice carrée. B est une matrice.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgesv.f`

```
Exemple :
> // resout un systeme complexe d'equations lineaires AX=B
> // avec A matrice carree
> _affc=1$
> A = matrixC[
1*i, 0:
0, 2*i]$
> B = matrixC[
7.37*i:
14.74]$
> S=lapack_zgesv(A,B)$
> afftab(S);
[   7.37]
[(0-i*7.37)]
>
```

lapack_zgbsv

`lapack_zgbsv` [Fonction]

`lapack_zgbsv(`<matrice complexe> A `,`<matrice complexe> B`)`

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice bande d'ordre N avec KL diagonales inférieures et KU supérieures, et X et B sont des matrices à N lignes et NRHS colonnes. Elle utilise la décomposition LU.

A est une matrice bande. B est une matrice.

Dans le cas où A est diagonale, l'appel est équivalent à "lapack_zgesv(A,B);"

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgbsv.f`

```
Exemple :
> // resout un systeme complexe d'equations lineaires AX=B
> // avec A matrice bande
```

```
> _affc=1$
> A = matrixC[
-1.65+2.26*i, -2.05-0.85*i, 0.97-2.84*i, 0:
6.30*i, -1.48-1.75*i, -3.99+4.01*i, 0.59-0.48*i:
0, -0.77+2.83*i, -1.06+1.94*i, 3.33-1.04*i:
0, 0, 4.48-1.09*i, -0.46-1.72*i]$
> B = matrixC[
-1.06+21.50*i:
-22.72-53.90*i:
28.24-38.60*i:
-34.56+16.73*i]$
> S = lapack_zgbsv(A, B);
S  matrice complexe double-precision [ 1:4 , 1:1 ]
> afftab(S);
[(-3+i*2)]
[(1-i*7)]
[(-5+i*4)]
[(6-i*8)]
>
```

lapack_zgtsv

`lapack_zgtsv` [Fonction]

`lapack_zgtsv(`<matrice complexe> A `,`<matrice complexe> B`)`

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice à N lignes et N colonnes tridiagonale, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgtsv.f`

```
Exemple :
> // exemple de la routine zgtsv
> _affc=1$
> A = matrixC[
-1.3+1.3*i, 2-1*i, 0, 0, 0:
1-2*i, -1.3+1.3*i, 2+1*i, 0, 0:
0, 1+i, -1.3+3.3*i, -1+i, 0:
0, 0, 2-3*i, -0.3+4.3*i, 1-i:
0, 0, 0, 1+i, -3.3+1.3*i]$
> B = matrixC[
2.4-5*i:
3.4+18.2*i:
-14.7+9.7*i:
31.9-7.7*i:
-1.0+1.6*i]$
> S = lapack_zgtsv(A, B);
S  matrice complexe double-precision [ 1:5 , 1:1 ]
> afftab(S);
[(1+i*1)]
[(3-i*1)]
[(4+i*5)]
[(-1-i*2)]
[(1-i*1)]
>
```

lapack_zsysv

lapack_zsysv [Fonction]

lapack_zsysv(<matrice complexe> A ,<matrice complexe> B)

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice symétrique de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/complex16/zsysv.f

```
Exemple :
> // exemple de la routine zsysv
> _affc=1$
> A = matrixC[
-0.56+0.12*i, -1.54-2.86*i, 5.32-1.59*i, 3.80+0.92*i:
-1.54-2.86*i, -2.83-0.03*i, -3.52+0.58*i, -7.86-2.96*i:
5.32-1.59*i, -3.52+0.58*i, 8.86+1.81*i, 5.14-0.64*i:
3.80+0.92*i, -7.86-2.96*i, 5.14-0.64*i, -0.39-0.71*i]$
> B = matrixC[
-6.43+19.24*i:
-0.49-1.47*i:
-48.18+66*i:
-55.64+41.22*i]$
> S = lapack_zsysv(A, B);
S  matrice complexe double-precision [ 1:4 , 1:1 ]
> afftab(S);
[(-4+i*3)]
[(3-i*2)]
[(-2+i*5)]
[(1-i*1)]
>
```

lapack_zhesv

lapack_zhesv [Fonction]

lapack_zhesv(<matrice complexe> A ,<matrice complexe> B)

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice hermitienne de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/complex16/zhesv.f

```
Exemple :
> // exemple de la routine zhesv
> _affc=1$
> A = matrixC[
-1.84, 0.11-0.11*i, -1.78-1.18*i, 3.91-1.50*i:
0.11+0.11*i, -4.63, -1.84+0.03*i, 2.21+0.21*i:
-1.78+1.18*i, -1.84-0.03*i, -8.87, 1.58-0.90*i:
3.91+1.50*i, 2.21-0.21*i, 1.58+0.90*i, -1.36]$
> B = matrixC[
2.98-10.18*i:
-9.58+3.88*i:
-0.77-16.05*i:
7.79+5.48*i]$
> S = lapack_zhesv(A, B);
S  matrice complexe double-precision [ 1:4 , 1:1 ]
```

```
> afftab(S);
[(2+i*1)]
[(3-i*2)]
[(-1+i*2)]
[(1-i*1)]
>
```

lapack_zposv

lapack_zposv [Fonction]

lapack_zposv(<matrice complexe> A ,<matrice complexe> B)

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice hermitienne définie positive de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/complex16/zposv.f

```
Exemple :
> // exemple de la routine zposv
> _affc=1$
> A = matrixC[
3.23, 1.51-1.92*i, 1.90+0.84*i, 0.42+2.50*i:
1.51+1.92*i, 3.58, -0.23+1.11*i, -1.18+1.37*i:
1.90-0.84*i, -0.23-1.11*i, 4.09, 2.33-0.14*i:
0.42-2.50*i, -1.18-1.37*i, 2.33+0.14*i, 4.29]$
> B = matrixC[
3.93-6.14*i:
6.17+9.42*i:
-7.17-21.83*i:
1.99-14.38*i]$
> S = lapack_zposv(A, B);
S  matrice complexe double-precision [ 1:4 , 1:1 ]
> afftab(S);
[(1-i*1)]
[(-4.34355E-15+i*3)]
[(-4-i*5)]
[(2+i*1)]
>
```

lapack_zpbsv

lapack_zpbsv [Fonction]

lapack_zpbsv(<matrice complexe> A ,<matrice complexe> B)

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice hermitienne définie positive bande de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, http://www.netlib.org/lapack/complex16/zpbsv.f

```
Exemple :
> // exemple de la routine zpbsv
> _affc=1$
> A = matrixC[
16, 16-16*i, 0, 0:
16+16*i, 41, 18+9*i, 0:
0, 18-9*i, 46, 1+4*i:
0, 0, 1-4*i, 21]$
```

```
> B = matrixC[
64+16*i:
93+62*i:
78-80*i:
14-27*i]$
> S = lapack_zpbsv(A, B);
AB  matrice complexe double-precision [ 1:2 , 1:4 ]
S  matrice complexe double-precision [ 1:4 , 1:1 ]
> afftab(S);
[(2+i*1)]
[(1+i*1)]
[(1-i*2)]
[(1-i*1)]
>
```

lapack_zptsv

`lapack_zptsv` [Fonction]

`lapack_zptsv(`<matrice complexe> A `,`<matrice complexe> B`)`

Cette routine résout le système matriciel complexe $AX=B$, où A est une matrice tridiagonale hermitienne définie positive de rang N, et X et B sont des matrices à N lignes et NRHS colonnes.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zptsv.f`

```
Exemple :
> // exemple de la routine zptsv
> _affc=1$
> A = matrixC[
9.39, 1.08-1.73*i, 0, 0:
1.08+1.73*i, 1.69, -0.04+0.29*i, 0:
0, -0.04-0.29*i, 2.65, -0.33+2.24*i:
0, 0, -0.33-2.24*i, 2.17]$
> B = matrixC[
-12.42+68.42*i:
-9.93+0.88*i:
-27.30-0.01*i:
5.31+23.63*i]$
> S = lapack_zptsv(A, B);
S  matrice complexe double-precision [ 1:4 , 1:1 ]
> afftab(S);
[(-1+i*8)]
[(2-i*3)]
[(-4-i*5)]
[(7+i*6)]
>
```

16.1.2 Moindres Carres

Cas réel général, (voir [lapack_dgels], page 156)

Cas réel général, utilisant la décomposition en valeurs singulières (voir [lapack_dgelss], page 158)

Cas réel des problèmes (LSE) des moindres carrés à contraintes d'égalité, (voir [lapack_dgglse], page 159)

Cas réel des modèles de Gauss-Markov (GLM) linéaires, (voir [lapack_dggglm], page 160)

Cas complexe général, (voir [lapack_zgels], page 161)

Cas complexe général, utilisant la décomposition en valeurs singulières (voir [lapack_zgelss], page 162)

Cas complexe des problèmes (LSE) des moindres carrés à contraintes d'égalité, (voir [lapack_zgglse], page 163)

Cas complexe des modèles de Gauss-Markov (GLM) linéaires, (voir [lapack_zggglm], page 164)

lapack_dgels

`lapack_dgels` [Fonction]

`lapack_dgels(`<matrice reelle> *A* `,`<matrice reelle> *B* `,`<chaine> *TRANS*`)`

Cette routine résout un système surdéterminé ou sous-déterminé, impliquant une matrice à M lignes et N colonnes A, ou sa transposée, en utilisant une factorisation QR ou RQ de A. On doit avoir A de plein rang.

Quatre options sont proposées:

- cas 1: si *TRANS* = 'N' and M >= N: elle résout le problème des moindres carrés: min ||*B* - *A**X||
- cas 2: si *TRANS* = 'N' and M < N : elle renvoie la solution de norme Euclidienne minimale au système sous-déterminé *A**X=*B*
- cas 3: si *TRANS* = 'T' and M >= N: elle renvoie la solution de norme Euclidienne minimale au système non déterminé *A***T*X=*B*
- cas 4: si *TRANS* = 'T' and M < N : elle résout le problème des moindres carrés: min ||*B* - *A***T*X||

Formats de sortie:

- cas 1: Les lignes 1 à N de la matrice retournée contiennent les vecteurs solution des moindres carrés. Les lignes N+1 à M contiennent les carrés résiduels.
- cas 2: Les lignes 1 à N de la matrice retournée contiennent les vecteurs solution de norme Euclidienne minimales.
- cas 3: Les lignes 1 à M de la matrice retournée contiennent les vecteurs solution de norme Euclidienne minimales.
- cas 4: Les lignes 1 à M de la matrice retournée contiennent les vecteurs solution des moindres carrés. Les lignes M+1 à N contiennent les carrés résiduels.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgels.f`

```
Exemple :
> // probleme des moindres carres reel: min ||B-A*X||
> _affc=1$
> A = matrixR[
-3, 1:
1, 1:
4-7, 1:
5, 1]$
> B = matrixR[
70:
21:
110:
-35]$
```

```
> S = lapack_dgels(A, B, "N")$
> afftab(S);
[ - 15.7727]
[   41.5]
[   28.5243]
[ - 4.13405]
>
> // la somme des carres residuels s'obtient ici en calculant:
> // sqrt(S[3]**2+S[4]**2)
> resid=sqrt(S[3,1]**2+S[4,1]**2);
resid =    28.8223
>
> // solution de norme minimale au systeme sousdetermine AX=B
> A = matrixR[
1, 2, 3, 4:
5, 6, 7, 8:
9, 10, 11, 12]$
> B = matrixR[
30:
70:
110]$
> S = lapack_dgels(A, B, "N")$
> afftab(S);
[   1]
[   2]
[   3]
[   4]
>
> // attention au nombre de lignes de la solution
> size(B);
 3
> size(S);
 4
>
> // solution de norme minimale au systeme non determine A**T*X=B,
> // ou A**T est la transposee de A
> A = matrixR[
1, 5, 9:
2, 6, 10:
3, 7, 11:
4, 8, 12]$
> B = matrixR[
30:
70:
110]$
> S = lapack_dgels(A, B, "T")$
> afftab(S);
[   1]
[   2]
[   3]
[   4]
>
```

```
> // ici aussi, attention au nombre de lignes de la solution
> size(B);
 3
> size(S);
 4
>
> // probleme des moindres carres reel: min ||B-A**T*X||
> A = matrixR[
-3, 1, -7, 5:
1, 1, 1, 1]$
> B = matrixR[
70:
21:
110:
-35]$
> S = lapack_dgels(A, B, "T")$
> afftab(S);
[ - 12.1]
[   29.4]
[ - 6.80331]
[ - 4.23261]
>
> // la somme des carres residuels s'obtient ici en calculant:
> // sqrt(S[3]**2+S[4]**2)
> resid=sqrt(S[3,1]**2+S[4,1]**2);
resid =    8.01249
```

lapack_dgelss

`lapack_dgelss` [Fonction]

`lapack_dgelss(`<matrice reelle> *A* `,`<matrice reelle> *B* `,`<entier> *RCOND*`)`

Cette routine calcule la solution de norme minimale au problème réel des moindres carrés suivant: min || |*B*-*A**X| || en utilisant la décomposition en valeurs singulières de *A*, matrice à M lignes et N colonnes qu peut être de rang déficient.

B est une matrice à M lignes. *RCOND* est la précision utilisée. Mettre RCOND à -1 permet de calculer avec la précision machine.

Les lignes 1 à N de la matrice retournée contiennent les vecteurs solution des moindres carrées. Si *A* est de rang N, et que M>=N, alors les lignes N+1 à M contiennent les carrés résiduels.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgelss.f`

```
Exemple :
> // probleme des moindres carres reel: min ||B-A*X||
> _affc=1$
>
> A = [
-0.09, 0.14, -0.46, 0.68, 1.29:
-1.56, 0.20, 0.29, 1.09, 0.51:
-1.48, -0.43, 0.89, -0.71, -0.96:
-1.09, 0.84, 0.77, 2.11, -1.27:
0.08, 0.55, -1.13, 0.14, 1.74:
-1.59, -0.72, 1.06, 1.24, 0.34]$
> B = matrixR[
```

```
7.4:
4.2:
-8.3:
1.8:
8.6:
2.1]$
>
> // on selectionne une precision de 1.E-2
> S = lapack_dgelss(A, B, 1E-2)$
> afftab(S);
[   0.634385]
[   0.969928]
[ - 1.44025]
[   3.36777]
[   3.39917]
[ - 0.00347521]
>
> // avec la precision machine
> S = lapack_dgelss(A, B, -1)$
> afftab(S);
[ - 0.799745]
[ - 3.28796]
[ - 7.47498]
[   4.93927]
[   0.767833]
[ - 0.00347521]
>
>
```

lapack_dgglse

`lapack_dgglse` [Fonction]

`lapack_dgglse(`<matrice reelle> A `,`<matrice reelle> B `,`<matrice reelle> C `,`<matrice reelle> D`)`

`lapack_dgglse(`<matrice reelle> A `,`<matrice reelle> B `,`<matrice reelle> C `,`<matrice reelle> D `,`<identificateur> T `,`<identificateur> R `,`<identificateur> S`)`

Cette routine résout le problème linéaire des moindres carrés à contrainte d'égalité:

min || C - A*X ||, contrainte B*X = D

où A est une matrice M-N, B est une matrice P-N, C est un M-vecteur, and D est un P-vecteur.

On suppose P <= N <= M+P, et

rang(B) = P et rang((A)) = N. ((B))

Elle utilise une factorisation RQ généralisée des matrices (B,A) avec B = (0 R)*Q et A = Z*T*Q.

Dans la version complète de la fonction, vous pouvez récupérer T, R et le vecteur S des carrés résiduels, de taille M-N+P.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgglse.f`

```
Exemple :
> // exemple de la routine dgglse
> _affc=1$
```

```
> A = matrixR[
1, 2:
3, 4:
5, 6]$
> B = matrixR[
2.50, 0.00:
0.00, 2.50]$
> C = matrixR[
-1:
-2:
-3]$
> D = matrixR[
3.00:
4.00]$
> S = lapack_dgglse(A,B,C,D)$
> afftab(S);
[   1.2]
[   1.6]
```

lapack_dggglm

lapack_dggglm [Fonction]

lapack_dggglm(<matrice reelle> A ,<matrice reelle> B ,<matrice reelle> D ,<identificateur> X ,<identificateur> Y) lapack_dggglm(<matrice reelle> A ,<matrice reelle> B ,<matrice reelle> D ,<identificateur> X ,<identificateur> Y ,<identificateur> T ,<identificateur> R)

Cette routine résout un problème de modèle Gauss-Markov linéaire généralisé:

min || Y ||_2, contrainte $D = A^*X$ + B^*Y X où A est une matrice N-M, B est une matrice N-P, and D est un N-vecteur. On suppose M <= N <= M+P, rang(A) = M et rang(AB) = N.

Elle utilise une factorisation QR généralisée des matrices (B,A) avec B = Q*T*Z et A = Q*(R). (0)

En particulier, si B est carrée et inversible, alors le problème GLM est équivalent au problème des moindres carrés pondérés

min || inv(B)*(D-A^*X) ||_2 X

avec inv(B), l'inverse de B.

Dans la version complète de la fonction, vous pouvez récupérer T et R.

Pour plus d'informations, http://www.netlib.org/lapack/double/dggglm.f

```
Exemple :
> // exemple de la routine dggglm
> _affc=1$
> A = matrixR[
7687, 13450.888:
9999, 15499.08:
7594, 12450.12]$
> B = matrixR[
2.50, 11.77:
7, 7:
12.00, 2.50]$
> D = matrixR[
20777:
35713:
```

```
21777]$
> lapack_dggglm(A, B, D, X, Y)$
> afftab(X);
[   11.8512]
[ - 5.31511]
> afftab(Y);
[ - 200.172]
[   141.898]
```

lapack_zgels

`lapack_zgels` [Fonction]

`lapack_zgels(`<matrice complexe> *A* `,`<matrice complexe> *B* `,`<chaine> *TRANS*`)`

Cette routine résout un système complexe surdéterminé ou sous-déterminé, impliquant une matrice à M lignes et N colonnes *A*, ou sa transposée conjuguée, en utilisant une factorisation QR ou RQ de *A*. On doit avoir *A* de plein rang.

Quatre options sont proposées:

- cas 1: si *TRANS* = 'N' and M >= N: elle résout le problème des moindres carrés: min ||*B* - *A**X||
- cas 2: si *TRANS* = 'N' and M < N : elle renvoie la solution de norme Euclidienne minimale au système sous-déterminé *A**X=*B*
- cas 3: si *TRANS* = 'T' and M >= N: elle renvoie la solution de norme Euclidienne minimale au système non déterminé *A***H*X=*B*
- cas 4: si *TRANS* = 'T' and M < N : elle résout le problème des moindres carrés: min ||*B* - *A***H*X||

Formats de sortie:

- cas 1: Les lignes 1 à N de la matrice retournée contiennent les vecteurs solution des moindres carrés. Les lignes N+1 à M contiennent les carrés résiduels.
- cas 2: Les lignes 1 à N de la matrice retournée contiennent les vecteurs solution de norme Euclidienne minimales.
- cas 3: Les lignes 1 à M de la matrice retournée contiennent les vecteurs solution de norme Euclidienne minimales.
- cas 4: Les lignes 1 à M de la matrice retournée contiennent les vecteurs solution des moindres carrés. Les lignes M+1 à N contiennent les carrés résiduels.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgels.f`

```
Exemple :
> // resout un probleme des moindres carres complexe: min ||B-A*X||
> _affc=1$
> A = matrixC[
-0.57, -1.28, -0.39, 0.25:
-1.93, 1.08, -0.31, -2.14:
2.30, 0.24, 0.40, -0.35:
-1.93, 0.64, -0.66, 0.08:
0.15, 0.30, 0.15, -2.13:
-0.02, 1.03, -1.43, 0.50]$
> B = matrixC[
-2.67:
-0.55:
3.34:
```

```
-0.77:
0.48:
4.10+i]$
> S = lapack_zgels(A, B, "N")$
> afftab(S);
[(1.53387+i*0.158054)]
[(1.87075+i*0.0726528)]
[(-1.52407-i*0.621165)]
[(0.039183-i*0.0141075)]
[(-0.0085366-i*0.0685483)]
[(0.0204427+i*0.205957)]
>
```

lapack_zgelss

`lapack_zgelss` [Fonction]

`lapack_zgelss(`<matrice complexe> *A* `,`<matrice complexe> *B* `,`<réel> *RCOND*`)`

Cette routine calcule la solution de norme minimale au problème complexe des moindres carrés suivant: min || |*B*-*A**X| || en utilisant la décomposition en valeurs singulières de *A*, matrice à M lignes et N colonnes qu peut être de rang déficient.

B est une matrice à M lignes. *RCOND* est la précision utilisée. Mettre RCOND à -1 permet de calculer avec la précision machine.

Les lignes 1 à N de la matrice retournée contiennent les vecteurs solution des moindres carrées. Si *A* est de rang N, et que M>=N, alors les lignes N+1 à M contiennent les carrés résiduels.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgelss.f`

```
Exemple :
> // probleme des moindres carres reel: min ||B-A*X||
> _affc=1$
>
> A = [
-0.09, 0.14, -0.46, 0.68, 1.29:
-1.56, 0.20, 0.29, 1.09, 0.51:
-1.48, -0.43, 0.89, -0.71, -0.96:
-1.09, 0.84, 0.77, 2.11, -1.27:
0.08, 0.55, -1.13, 0.14, 1.74:
-1.59, -0.72, 1.06, 1.24, 0.34]$
> B = matrixC[
7.4:
4.2*i:
-8.3*i:
1.8*i:
8.6:
2.1*i]$
>
> // on sélectionne une précision de 1.E-2
> S = lapack_zgelss(A, B, 1E-2)$
> afftab(S);
[(-1.88522+i*2.51961)]
[(2.23426-i*1.26433)]
[(-2.22465+i*0.784398)]
[(-0.257364+i*3.62513)]
```

```
[(2.36045+i*1.03872)]
[(3.35419-i*3.35767)]
>
> // avec la précision machine
> S = lapack_zgelss(A, B, -1)$
> afftab(S);
[(307.51-i*308.309)]
[(920.819-i*924.107)]
[(1299.69-i*1307.17)]
[(-339.29+i*344.229)]
[(570.037-i*569.27)]
[(3.35419-i*3.35767)]
>
> // résultat faux. Attention au paramétrage de RCOND
>
```

lapack_zgglse

`lapack_zgglse` [Fonction]

`lapack_zgglse(`<matrice complexe> *A* `,`<matrice complexe> *B* `,`<matrice complexe> *C* `,`<matrice complexe> *D*`) lapack_zgglse(`<matrice complexe> *A* `,`<matrice complexe> *B* `,`<matrice complexe> *C* `,`<matrice complexe> *D* `,`<identificateur> *T* `,`<identificateur> *R* `,`<identificateur> *S*`)`

Cette routine résout le problème linéaire des moindres carrés à contrainte d'égalité:

min || *C* - *A**X ||, contrainte *B**X = *D*

où *A* est une matrice M-N, *B* est une matrice P-N, *C* est un M-vecteur, and *D* est un P-vecteur.

On suppose P <= N <= M+P, et

rang(B) = P and rang((A)) = N. ((B))

Elle utilise une factorisation RQ généralisée des matrices (B,A) avec B = (0 R)*Q et A = Z*T*Q.

Dans la version complète de la fonction, vous pouvez récupérer T, R et le vecteur S des carrés résiduels, de taille M-N+P.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgglse.f`

```
Exemple :
> // exemple de la routine zgglse
> _affc=1$
> A = matrixC[
1, 2:
3, 4:
5, 6]$
> B = matrixC[
2.50*i, 0.00:
0.00, 2.50]$
> C = matrixC[
-1:
-2*i:
-3]$
> D = matrixC[
3.00:
```

```
4.00*i]$
> S = lapack_zgglse(A,B,C,D)$
> afftab(S);
[(0-i*1.2)]
[(0+i*1.6)]
```

lapack_zggglm

lapack_zggglm [Fonction]

lapack_zggglm(<matrice complexe> A ,<matrice complexe> B ,<matrice complexe> D ,<identificateur> X ,<identificateur> Y) lapack_zggglm(<matrice complexe> A ,<matrice complexe> B ,<matrice complexe> D ,<identificateur> X ,<identificateur> Y ,<identificateur> T ,<identificateur> R)

Cette routine résout un problème de modèle Gauss-Markov linéaire généralisé:

min || Y ||_2, contrainte $D = A^*X$ + B^*Y X où A est une matrice N-M, B est une matrice N-P, and D est un N-vecteur. On suppose M <= N <= M+P, rang(A) = M et rang(AB) = N.

Elle utilise une factorisation QR généralisée des matrices (B,A) avec B = Q*T*Z et A = Q*(R). (0)

En particulier, si B est carrée et inversible, alors le problème GLM est équivalent au problème des moindres carrés pondérés

min || inv(B)*(D-A^*X) ||_2 X

avec inv(B), l'inverse de B.

Dans la version complète de la fonction, vous pouvez récupérer T et R.

Pour plus d'informations, http://www.netlib.org/lapack/complex16/zggglm.f

```
Exemple :
> // exemple de la routine zggglm
> _affc=1$
> A = matrixC[
7687, 13450.888:
9999, 15499.08:
7594, 12450.12]$
> B = matrixC[
2.50, 11.77*i:
7*i, 7:
12.00, 2.50]$
> D = matrixC[
20777*i:
35713*i:
21777*i]$
> lapack_zggglm(A, B, D, X, Y)$
> afftab(X);
[(-1.49403+i*10.5244)]
[(0.851288-i*4.50903)]
> afftab(Y);
[(45.469-i*168.668)]
[(80.5683+i*6.76462)]
>
```

16.1.3 Factorisations

Factorisation QR dans le cas général réel, (voir [lapack_dgeqrf], page 165)

lapack_dgeqrf

`lapack_dgeqrf` [Fonction]

`lapack_dgeqrf(`<matrice reelle> *A* `,`<identificateur> *Q* `,`<identificateur> *R*`)`

Cette routine calcule une factorisation QR d'une matrice réelle à M lignes et N colonnes *A*: A = Q*R.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgeqrf.f`

```
Exemple :
> // calcule une factorisation QR d'une matrice A
> _affc=1$
> A = matrixR[
2, 3, -1, 0, 20:
-6, -5, 0, 2, -33:
2, -5, 6, -6, -43:
4, 6, 2, -3, 49]$
> lapack_dgeqrf(A, Q, R)$
> afftab(Q);
[ - 0.258199  - 0.182574    0.208237  - 0.925547]
[   0.774597    0  - 0.535468  - 0.336563]
[ - 0.258199    0.912871  - 0.267734  - 0.168281]
[ - 0.516398  - 0.365148  - 0.773453    0.0420703]
> afftab(R);
[ - 7.74597  - 6.45497  - 2.32379    4.64758  - 44.9266]
[   0  - 7.30297    4.9295  - 4.38178  - 60.7972]
[   0    0  - 3.36155    2.85583  - 4.55148]
[   0    0    0    0.210352    1.89316]
```

lapack_dgeqlf

`lapack_dgeqlf` [Fonction]

`lapack_dgeqlf(`<matrice reelle> *A* `,`<identificateur> *Q* `,`<identificateur> *L*`)`

Cette routine calcule une factorisation QL d'une matrice réelle à M lignes et N colonnes *A*: A = Q*L.

On suppose M >= N.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgeqlf.f`

```
Exemple :
> // calcule une factorisation QL d'une matrice A
> _affc=1$
> A = matrixR[
2, 3, -1, 0:
-6, -5, 0, 2:
2, -5, 6, -6:
4, 6, 2, -3:
```

```
20, -33, -43, 49]$
> lapack_dgeqlf(A, Q, L)$
> afftab(Q);
[   0.258503  - 0.0987267  - 0.446322    0]
[ - 0.0598575    0.395467    0.782976  - 0.0404061]
[ - 0.292516  - 0.875179    0.329003    0.121218]
[ - 0.914354    0.236816  - 0.28182    0.0606092]
[ - 0.089356  - 0.108807  - 0.00892644  - 0.989949]
> afftab(L);
[ - 5.15342    0    0    0]
[ - 5.54949    7.11392    0    0]
[ - 6.2383  - 8.29521    2.24054    0]
[ - 19.0717    32.6279    43.4164  - 49.4975]
```

lapack_dpotrf

`lapack_dpotrf` [Fonction]

`lapack_dpotrf(`<matrice reelle> A `)`

Cette routine calcule la factorisation de Cholesky (L) d'une matrice symétrique réelle A:

A = L*L**T

Pour plus d'informations, `http://www.netlib.org/lapack/double/dpotrf.f`

```
Exemple :
> _affc=1$
>
> A=matrixR[4.16, -3.12 , 0.56,  -0.10:
 -3.12 ,  5.03 , -0.83 ,  1.18:
  0.56 , -0.83,   0.76 ,  0.34 :
 -0.10 ,  1.18 ,  0.34 ,  1.18 ];
A  matrice reelle double-precision [ 1:4 , 1:4 ]
>
> L=lapack_dpotrf(A);
L  matrice reelle double-precision [ 1:4 , 1:4 ]
> afftab(L);
[   2.03961    0    0    0]
[ - 1.52971    1.64012    0    0]
[   0.274563  - 0.249981    0.788749    0]
[ - 0.049029    0.67373    0.661658    0.534689]
```

lapack_zgeqrf

`lapack_zgeqrf` [Fonction]

`lapack_zgeqrf(`<matrice complexe> A `,`<identificateur> Q `,`<identificateur> R`)`

Cette routine calcule une factorisation QR d'une matrice complexe à M lignes et N colonnes A:

A = Q*R.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgeqrf.f`

```
Exemple :
> // calcule une factorisation QR d'une matrice A
> _affc=1$
> A = matrixC[
```

```
1+i, -5-i, 3+5*i:
0, 8*i, 27:
0, 0, 2+3*i]$
> lapack_zgeqrf(A, Q, R)$
> afftab(Q);
[(-0.707107-i*0.707107)    0    0]
[   0 (0-i*1)    0]
[   0    0 (-0.5547-i*0.83205)]
> afftab(R);
[ - 1.41421 (4.24264-i*2.82843) (-5.65685-i*1.41421)]
[   0  - 8 (0+i*27)]
[   0    0  - 3.60555]
>
```

lapack_zgeqlf

`lapack_zgeqlf` [Fonction]

`lapack_zgeqlf(`<matrice complexe> *A* `,`<identificateur> *Q* `,`<identificateur> *L*`)`

Cette routine calcule une factorisation QL d'une matrice complexe à M lignes et N colonnes *A*:

A = Q*L.

On suppose M >= N.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgeqlf.f`

```
Exemple :
> // calcule une factorisation QL d'une matrice A
> _affc=1$
> A = matrixC[
1+i, 0, 0:
-5-i, 8*i, 0:
3+5*i, 27, 2+3*i]$
> lapack_zgeqlf(A, Q, L)$
> afftab(Q);
[(-0.707107-i*0.707107)    0    0]
[   0 (0-i*1)    0]
[   0    0 (-0.5547-i*0.83205)]
> afftab(L);
[ - 1.41421    0    0]
[(1-i*5)  - 8    0]
[(-5.82435-i*0.27735) (-14.9769+i*22.4654)  - 3.60555]
```

lapack_zpotrf

`lapack_zpotrf` [Fonction]

`lapack_zpotrf(`<matrice complexe> *A* `)`

Cette routine calcule la factorisation de Cholesky (L) d'une matrice symétrique définie positive *A*:

A = L*L**H

Pour plus d'informations, `http://www.netlib.org/lapack/double/zpotrf.f`

```
Exemple :
>   A=matrixC[ 3.23+I* 0.00 , 1.51-I* 1.92 ,  1.90-I*-0.84 , 0.42-I*-2.50  :
```

```
  1.51+I* 1.92  ,  3.58+I* 0.00 ,  -0.23-I*-1.11 ,   -1.18-I*-1.37 :
  1.90+I*-0.84 ,  -0.23+I*-1.11  ,  4.09-I* 0.00 ,   2.33-I* 0.14  :
  0.42+I*-2.50 ,  -1.18+I*-1.37 ,   2.33+I* 0.14 ,   4.29+I* 0.00  ];
A  matrice complexe double-precision [ 1:4 , 1:4 ]
>
>   L=lapack_zpotrf(A);
L  matrice complexe double-precision [ 1:4 , 1:4 ]
>   afftab(L);
[         1.797220075561143                            0                            0
0]
[(     0.8401864749527325+i*      1.068316577423342)          1.316353439509685
0                          0]
[(      1.057188279741849-i*      0.467388502622712) (    -0.4701749470106329+i*     0
1.560392977137124                            0]
[(      0.233694251311356-i*      1.391037210186643) (    0.08335250923944196+i*    0.0
(     0.9359617337923402+i*     0.9899692192815739)          0.6603332973655888]
```

16.1.4 Decompositions en Valeurs Singulieres

Décomposition SVD dans le cas général réel, (voir [lapack_dgesvd], page 168)

Décomposition SVD dans le cas général réel utilisant un algorithme Divide and Conquer, (voir [lapack_dgesdd], page 169)

Décomposition SVD dans le cas général complexe, (voir [lapack_zgesvd], page 169)

Décomposition SVD dans le cas général complexe utilisant un algorithme Divide and Conquer, (voir [lapack_zgesdd], page 170)

lapack_dgesvd

`lapack_dgesvd` [Fonction]

`lapack_dgesvd(`<matrice reelle> *A* `,`<identificateur> *U* `,`<identificateur> *SIGMA* `,`<identificateur> *VT*`)`

Cette routine calcule la décomposition en valeurs singulières (SVD) d'une matrice réelle A: *A*=*U***SIGMA***VT* où *A* est une matrice M-N, *U* la matrice des vecteurs singuliers à gauche, *VT* la transposée de la matrice des vecteurs singuliers à droite, et *SIGMA* la matrice contenant sur sa diagonale les valeurs singulières de *A*.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgesvd.f`

```
Exemple :
> // DGESVD effectue la decomposition en valeurs singulieres de A
> _affc=1$
> A = matrixR[
1, 2:
3, 4]$
> lapack_dgesvd(A, U, SIGMA, VT)$
> afftab(U);
[ - 0.404554  - 0.914514]
[ - 0.914514    0.404554]
> afftab(SIGMA);
[   5.46499    0]
[   0    0.365966]
> afftab(VT);
[ - 0.576048  - 0.817416]
[   0.817416  - 0.576048]
```

```
>
```

lapack_dgesdd

`lapack_dgesdd` [Fonction]

`lapack_dgesdd(`<matrice reelle> *A* `,`<identificateur> *U* `,`<identificateur> *SIGMA* `,`<identificateur> *VT*`)`

Cette routine calcule la décomposition en valeurs singulières (SVD) d'une matrice réelle A: *A=U*SIGMA*VT* où *A* est une matrice M-N, *U* la matrice des vecteurs singuliers à gauche, *VT* la transposée de la matrice des vecteurs singuliers à droite, et *SIGMA* la matrice contenant sur sa diagonale les valeurs singulières de *A*.

Elle utilise un algorithme divide and conquer.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgesdd.f`

```
Exemple :
> // DGESDD effectue la decomposition en valeurs singulieres de A
> _affc=1$
> A = matrixR[
1, 2:
3, 4]$
> lapack_dgesdd(A, U, SIGMA, VT)$
> afftab(U);
[ - 0.404554  - 0.914514]
[ - 0.914514    0.404554]
> afftab(SIGMA);
[   5.46499    0]
[   0    0.365966]
> afftab(VT);
[ - 0.576048  - 0.817416]
[   0.817416  - 0.576048]
>
```

lapack_zgesvd

`lapack_zgesvd` [Fonction]

`lapack_zgesvd(`<matrice complexe> *A* `,`<identificateur> *U* `,`<identificateur> *SIGMA* `,`<identificateur> *VT*`)`

Cette routine calcule la décomposition en valeurs singulières (SVD) d'une matrice complexe A: *A=U*SIGMA*VT* où *A* est une matrice M-N, *U* la matrice des vecteurs singuliers à gauche, *VT* la matrice adjointe à celle des vecteurs singuliers à droite, et *SIGMA* la matrice contenant sur sa diagonale les valeurs singulières de *A*.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgesvd.f`

```
Exemple :
> // ZGESVD effectue la decomposition en valeurs singulieres de A
> _affc=1$
> A = matrixC[
1+i, -2-i:
3, 2*i]$
> lapack_zgesvd(A, U, SIGMA, VT)$
> afftab(U);
[(-0.158181-i*0.528862) (0.497893-i*0.668869)]
[(-0.824631+i*0.12356) (0.391736+i*0.388921)]
```

```
> afftab(SIGMA);
[   4.2049    0]
[   0    1.52278]
> afftab(VT);
[ - 0.751728 (0.259779-i*0.606152)]
[   0.659474 (0.29612-i*0.690947)]
>
```

lapack_zgesdd

`lapack_zgesdd` [Fonction]

`lapack_zgesdd(`<matrice complexe> *A* `,`<identificateur> *U* `,`<identificateur> *SIGMA* `,`<identificateur> *VT*`)`

Cette routine calcule la décomposition en valeurs singulières (SVD) d'une matrice complexe A: *A=U*SIGMA*VT* où *A* est une matrice M-N, *U* la matrice des vecteurs singuliers à gauche, *VT* la matrice adjointe à celle des vecteurs singuliers à droite, et *SIGMA* la matrice contenant sur sa diagonale les valeurs singulières de *A*.

Elle utilise un algorithme divide and conquer.

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgesdd.f`

16.1.5 Valeurs Propres et Vecteurs Propres

lapack_dsyev

`lapack_dsyev` [Fonction]

`lapack_dsyev(`<matrice reelle> *A*`,`<identificateur> *VALUES* `,`<identificateur> *VECTORS*`)`

Cette routine calcule les valeurs propres et les vecteurs propres d'une matrice *A* réelle et symétrique. Les valeurs propres sont stockées dans *VALUES*, et les vecteurs propres dans *VECTORS*.

Pour plus d'informations, `http://www.netlib.org/lapack/double/dsyev.f`

```
Exemple :
> // exemple de la routine dsyev
> _affc=1$
> A = matrixR[
451.27, 0:
0, 512.75]$
> lapack_dsyev(A, values, vectors);
vectors  matrice reelle double-precision [ 1:2 , 1:2 ]
```

```
> writes(values);
+4.5126999999999998E+02
+5.1275000000000000E+02
> afftab(vectors);
[   1    0]
[   0    1]
>
```

lapack_dsbev

lapack_dsbev [Fonction]

lapack_dsbev(<matrice reelle> *A*,<identificateur> *VALUES* ,<identificateur> *VECTORS*)

Cette routine calcule les valeurs propres et les vecteurs propres d'une matrice bande *A* réelle et symétrique. Les valeurs propres sont stockées dans *VALUES*, et les vecteurs propres dans *VECTORS*.

Pour plus d'informations, http://www.netlib.org/lapack/double/dsbev.f

```
Exemple :
> // exemple de la routine dsbev
> _affc=1$
> A = matrixR[
1, 2, 3, 0, 0:
2, 2, 3, 4, 0:
3, 3, 3, 4, 5:
0, 4, 4, 4, 5:
0, 0, 5, 5, 5]$
> lapack_dsbev(A, Values, Vectors);
> writes(Values);
-3.2473787952520272E+00
-2.6633015451836046E+00
+1.7511163179896112E+00
+4.1598880678262953E+00
+1.4999675954619729E+01
> afftab(Vectors);
[   0.0393846    0.623795    0.563458  - 0.516534    0.15823]
[   0.572127  - 0.257506  - 0.389612  - 0.595543    0.316058]
[ - 0.437179  - 0.590046    0.400815  - 0.147035    0.527682]
[ - 0.44235    0.430844  - 0.558098    0.0469667    0.552286]
[   0.533217    0.103873    0.242057    0.595564    0.540002]
>
```

lapack_dstev

lapack_dstev [Fonction]

lapack_dstev(<matrice reelle> *A*,<identificateur> *VALUES* ,<identificateur> *VECTORS*)

Cette routine calcule les valeurs propres et les vecteurs propres d'une matrice tridiagonale *A* réelle et symétrique. Les valeurs propres sont stockées dans *VALUES*, et les vecteurs propres dans *VECTORS*.

Pour plus d'informations, http://www.netlib.org/lapack/double/dstev.f

```
Exemple :
> // exemple de la routine dstev
> _affc=1$
```

```
> A = matrixR[
1, 1, 0, 0:
1, 4, 2, 0:
0, 2, 9, 3:
0, 0, 3, 16]$
> lapack_dstev(A, Values, Vectors);
> writes(Values);
+6.4756286546948838E-01
+3.5470024748920901E+00
+8.6577669890059994E+00
+1.7147667670632423E+01
> afftab(Vectors);
[   0.939572    0.338755  - 0.04937    0.00337683]
[ - 0.33114    0.86281  - 0.378064    0.0545279]
[   0.0852772  - 0.364803  - 0.855782    0.356769]
[ - 0.0166639    0.0878831    0.349668    0.932594]
>
```

lapack_dgeev

`lapack_dgeev` [Fonction]

`lapack_dgeev(`<matrice reelle> *A* `,`<identificateur> *VALUES* `,`<identificateur> *LVECTOR* `,`<identificateur> *RVECTOR*`)`

Cette routine calcule les valeurs propres et les vecteurs propres d'une matrice réelle *A*, qui peut ne pas être symétriques. Les valeurs propres sont stockées dans *VALUES*, le vecteur propre de gauche dans *LVECTOR* et le vecteur propre de droite dans *RVECTOR*.

Il résoud

*A*RVECTOR=VALUES*RVECTOR* et

*LVECTOR**T*A=VALUES*LVECTOR**T*

avec *LVECTOR**T* la matrice adjointe à *LVECTOR*

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgeev.f`

```
Exemple :
> // exemple de la routine dgeev
> _affc=1$
> A = matrixR[
10, 6:
4, 12]$
> lapack_dgeev(A, Values, LVector, RVector)$
> writes(Values);
+6.0000000000000000E+00
+1.6000000000000000E+01
> afftab(LVector);
[ - 0.707107  - 0.5547]
[   0.707107  - 0.83205]
> afftab(RVector);
[ - 0.83205  - 0.707107]
[   0.5547  - 0.707107]
>
```

lapack_dsygv

`lapack_dsygv` [Fonction]

`lapack_dsygv(<entier>` *ITYPE* `,<matrice reelle>` *A* `,<matrice reelle>` *B* `,<identificateur>` *VALUES* `,<identificateur>` *VECTORS*)

Cette routine calcule les valeurs propres et les vecteurs propres d'un problème de valeurs propres généralisées défini positif. *A* est symétrique, *B* est symétrique et définie positive. Les valeurs propres sont stockées dans *VALUES*, et les vecteurs propres dans *VECTORS*.

3 problèmes différents sont possibles:

*A**X=(lambda)**B**X, si *ITYPE* = 1

*A***B**X=(lambda)*X, si *ITYPE* = 2

*B***A**X=(lambda)*X, si *ITYPE* = 3

Pour plus d'informations, `http://www.netlib.org/lapack/double/dsygv.f`

```
Exemple :
> // exemple avec B*A*X = (lambda)*X
> _affc=1$
> B = matrixR[
10, 0:
0, 10]$
> A = matrixR[
451.27, 0:
0, 512.75]$
> lapack_dsygv(3, A, B, Values, Vectors);
Vectors  matrice reelle double-precision [ 1:2 , 1:2 ]
> writes(Values);
+4.5127000000000007E+03
+5.1275000000000009E+03
> afftab(Vectors);
[   3.16228    0]
[   0    3.16228]
```

lapack_dggev

`lapack_dggev` [Fonction]

`lapack_dggev(<matrice reelle>` *A* `,<matrice reelle>` *B* `,<identificateur>` *VALUES* `,<identificateur>` *LVECTOR* `,<identificateur>` *RVECTOR*)

Cette routine calcule les valeurs propres et les vecteurs propres d'un problème de valeurs propres généralisées défini positif. *A* ou *B* ne sont pas symétriques. Les valeurs propres sont stockées dans *VALUES*, le vecteur propre de gauche dans *LVECTOR* et le vecteur propre de droite dans *RVECTOR*.

Il résoud:

*A***RVECTOR*=*VALUES***B***RVECTOR*

et *LVECTOR***H**A*=*VALUES***LVECTOR***H**B*

avec *LVECTOR***H la matrice adjointe à *LVECTOR*

Pour plus d'informations, `http://www.netlib.org/lapack/double/dggev.f`

```
Exemple :
> // exemple de la routine dggev
> _affc=1$
```

```
> A = matrixR[
10, 6:
4, 12]$
> B = matrixR[
8, 10:
0.3, 12]$
> lapack_dggev(A, B, Values, LVector, RVector)$
> writes(Values);
+9.3655913978494620E-01 +3.9384647034270903E-01
+9.3655913978494620E-01 -3.9384647034270914E-01
> afftab(LVector);
[(0.629444-i*0.320052) (0.629444+i*0.320052)]
[(-0.704921-i*0.295079) (-0.704921+i*0.295079)]
> afftab(RVector);
[(0.7792-i*0.2208) (0.7792+i*0.2208)]
[(-0.283744-i*0.56193) (-0.283744+i*0.56193)]
>
```

lapack_dgges

`lapack_dgges` [Fonction]

`lapack_dgges(`<matrice reelle> *A* `,`<matrice reelle> *B* `,`<identificateur> *VALUES* `,`<identificateur> *S* `,`<identificateur> *T* `,`<identificateur> *VSL* `,`<identificateur> *VSR*`)`

Cette routine calcule les valeurs propres, les formes de Schur et les matrices des vecteurs de Schur d'un système généralisé de matrices réelles A et B, qui peuvent ne pas être symétriques. Les valeurs propres sont stockées dans *VALUES*, la forme de Schur généralisée de A dans S, celle de B dans T, la matrice des vecteurs de Schur de gauche dans *VSL* et celle de droite dans *VSR*.

Il résoud la factorisation de Schur généralisée:

$(A,B) = ((VSL)*S*(VSR)**\mathrm{T}, (VSL)*T*(VSR)**\mathrm{T})$

avec *VSR***T la matrice transposée à *VSR*

Pour plus d'informations, `http://www.netlib.org/lapack/double/dgges.f`

```
Exemple :
> // exemple de la routine dgges
> _affc=1$
> A = matrixR[
10, 6:
4, 12]$
> B = matrixR[
8, 10:
0.3, 12]$
> lapack_dgges(A, B, Values, S, T, VSL, VSR)$
> writes(Values);
+9.3655913978494620E-01 +3.9384647034270903E-01
+9.3655913978494620E-01 -3.9384647034270914E-01
> afftab(S);
[   15.3566    4.77834]
[ - 3.02259    5.31087]
> afftab(T);
[   16.6388    0]
[   0    5.58935]
```

```
> afftab(VSL);
[   0.735209    0.677841]
[   0.677841  - 0.735209]
> afftab(VSR);
[   0.365713    0.930728]
[   0.930728  - 0.365713]
>
```

lapack_zheev

lapack_zheev [Fonction]

lapack_zheev(<matrice complexe> *A*,<identificateur> *VALUES* ,<identificateur> *VECTORS*)

Cette routine calcule les valeurs propres et les vecteurs propres d'une matrice *A* complexe hermitienne. Les valeurs propres sont stockées dans *VALUES*, et les vecteurs propres dans *VECTORS*.

Pour plus d'informations, http://www.netlib.org/lapack/complex16/zheev.f

```
Exemple :
> // exemple de la routine zheev
> _affc=1$
> A = matrixC[
8+8*i, 8:
8, 8+8*i]$
> lapack_zheev(A, Values, Vectors);
Vectors  matrice complexe double-precision [ 1:2 , 1:2 ]
> writes(Values);
+0.0000000000000000E+00
+1.6000000000000000E+01
> afftab(Vectors);
[ - 0.707107    0.707107]
[   0.707107    0.707107]
```

lapack_zhbev

lapack_zhbev [Fonction]

lapack_zhbev(<matrice complexe> *A*,<identificateur> *VALUES* ,<identificateur> *VECTORS*)

Cette routine calcule les valeurs propres et les vecteurs propres d'une matrice bande *A* complexe hermitienne. Les valeurs propres sont stockées dans *VALUES*, et les vecteurs propres dans *VECTORS*.

Pour plus d'informations, http://www.netlib.org/lapack/complex16/zhbev.f

```
Exemple :
> // exemple de la routine zhbev
> _affc=1$
> A = matrixC[
8+8*i, 15+i:
15-i, 8+8*i]$
> lapack_zheev(A, Values, Vectors);
Vectors  matrice complexe double-precision [ 1:2 , 1:2 ]
> writes(Values);
-7.0332963783729099E+00
```

```
+2.3033296378372910E+01
> afftab(Vectors);
[(0.705541+i*0.047036) (0.705541+i*0.047036)]
[ - 0.707107    0.707107]
>
```

lapack_zgeev

`lapack_zgeev` [Fonction]

`lapack_zgeev(`<matrice complexe> *A* `,`<identificateur> *VALUES* `,`<identificateur> *LVECTOR* `,`<identificateur> *RVECTOR*`)`

Cette routine calcule les valeurs propres et les vecteurs propres d'une matrice complexe *A*. Les valeurs propres sont stockées dans *VALUES*, le vecteur propre de gauche dans *LVECTOR* et le vecteur propre de droite dans *RVECTOR*.

Il résoud

*A***RVECTOR*=*VALUES***RVECTOR* et

*LVECTOR***T**A*=*VALUES***LVECTOR***T

avec *LVECTOR***T la matrice transposée (je pense très fortement plutôt adjointe) à *LVECTOR*

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zgeev.f`

```
Exemple :
> //Fr exemple de la routine zgeev
> _affc=1$
> A = matrixC[
8, 0:
0, 8]$
> lapack_zgeev(A, Values, LV, RV);
> writes(Values);
+8.0000000000000000E+00 +0.0000000000000000E+00
+8.0000000000000000E+00 +0.0000000000000000E+00
> afftab(LV);
[   1    0]
[   0    1]
> afftab(RV);
[   1    0]
[   0    1]
>
```

lapack_zhegv

`lapack_zhegv` [Fonction]

`lapack_zhegv(`<entier> *ITYPE* `,`<matrice complexe> *A* `,`<matrice complexe> *B* `,`<identificateur> *VALUES* `,`<identificateur> *VECTORS*`)`

Cette routine calcule les valeurs propres et les vecteurs propres d'un problème de valeurs propres généralisées défini positif. *A* est hermitienne, *B* est hermitienne et définie positive. Les valeurs propres sont stockées dans *VALUES*, et les vecteurs propres dans *VECTORS*.

3 problèmes différents sont possibles:

*A**X=(lambda)**B**X, si *ITYPE* = 1

*A***B**X=(lambda)*X, si *ITYPE* = 2

*B***A**X=(lambda)*X, si *ITYPE* = 3

Pour plus d'informations, `http://www.netlib.org/lapack/complex16/zhegv.f`

Exemple :

```
> // exemple de la routine zhegv
> _affc=1$
> B = matrixC[
10,i:
-1*i, 10]$
> A = matrixC[
451.27, 0:
0, 512.75]$
> lapack_zhegv(3, A, B, Values, Vectors);
> afftab(Vectors);
[(0+i*2.69134) (0+i*1.66033)]
[ - 1.38287    2.84388]
> writes(Values);
+4.2492379742004214E+03
+5.3909620257995803E+03
```

17 Traitement numerique

17.1 Analyse en frequence

17.1.1 naf

`naf` [Commande]

`naf(`*KTABS*`,` *fichiersource*`,` *fichierresultat*`,` *XH*`,` *T0*`,` *NTERM*`,` *CX*`,` *CY*`);`

avec :

- *KTABS* : <entier> : nombre de données moins 1 dans le fichier source à lire.
- *fichiersource* : <chaine> : nom du fichier (avec son extension) contenant les données à traiter.
- *fichierresultat* : <chaine> : nom du fichier (avec ou sans extension) contenant les fréquences et les amplitudes trouvées.
- *XH* : <réel> : intervalle de temps entre 2 données (pas d'échantillonage).
- *T0* : <réel> : temps initial.
- *NTERM* : <entier> : nombre de fréquences à rechercher.
- *CX* : <entier> : numéro de la colonne de la partie réelle des données dans le fichier source. (ce numéro doit être supérieur ou égal à 1).
- *CY* : <entier> : numéro de la colonne de la partie imaginaire des données dans le fichier source. (ce numéro doit être supérieur ou égal à 1).

Elle effectue l'analyse en fréquence des données du fichier source et stocke les fréquences dans *fichierresultat*.sol. Elle utilise KTABS+1 données.

KTABS est arrondi à la plus proche valeur inférieure telle que KTABS = 6n avec n entier naturel positif.

Les paramètres utilisés sont stockés dans *fichierresultat*.par. Les resultats intermédiaires sont stockés dans *fichierresultat*.prt si `_naf_iprt`>=0. Elle emploie les variables globales `_naf_nulin`, `_naf_iprt`, `_naf_isec`, `_naf_iw`, `_naf_dtour`, `_naf_icplx`.

Le fichier *fichierresultat*.sol contient:

- sur la première ligne : la valeur de UNIANG.
- sur la deuxième ligne : le nombre de fréquences trouvées
- sur chaque ligne suivante : le numéro de la fréquence, la fréquence, le module de l'amplitude, la partie réelle de l'amplitude et la partie imaginaire de l'amplitude

```
Exemple :
Le fichier "tessin.out" contient des données
sur 32996 lignes avec une ligne d'entete.
tels que la colonne 2 contient la partie réelle des données.
la colonne 3 contient la partie imaginaire des données.
la première ligne du fichier est à ignorer.
Le temps initial est 0 et le pas est de 0.01.
Nous recherchons 10 frequences.
Le fichier resultat à pour nom "tesnaf".

> _naf_nulin=1;
> naf(32996,tessin.out, tesnaf, 0.01, 0, 10, 2, 3);

Le premier argument de naf n'est pas multiple de 6.
```

```
Le premier argument de naf devient 32994.
Les frequences trouvees sont sauves dans le fichier :tesnaf.sol
Les paramètres employees sont sauves dans le fichier :tesnaf.par

Le fichier tesnaf.par contient :
NOMFPAR = tesnaf.par
NOMFOUT =
NOMFDAT = tessin.out
NOMFSOL = tesnaf.sol
DTOUR   = 6. 83185307179586E+00
T0      = 0.000000000000000E+00
XH      = 1.000000000000000E-02
KTABS   = 32994
DNEPS   = 1.000000000000000E+100
NTERM   = 10
ICPLX   = 1
IW      = 1
ISEC    = 1
NULIN   = 1
ICX     = 2
ICY     = 3
IPRNAF  = -1
```

17.1.2 naftab

`naftab` [Commande]

`naftab`(*TX*, *TY*, *A*, *F*, *KTABS*, *XH*, *T0*, *NTERM*);

`naftab`(*TX*, *TY*, *A*, *F*, *KTABS*, *XH*, *T0*, *NTERM*, *TRESX*, *TRESY*);

`naftab`(*TX*, *TY*, *A*, *F*, *KTABS*, *XH*, *T0*, *NTERM*, *TRESX*, *TRESY*, `(`*FMIN1*, *FMAX1*, *NTERM1*`)`,...`);`

avec :

- *KTABS* : <entier> : nombre de données à traiter dans TX et TY.
- *TX* : <vec. réel> : partie réelle des données à traiter.
- *TY* : <vec. réel> : partie imaginaire des données à traiter.
- *A* : <vec. complexe> : tableau des amplitudes trouvées.
- *F* : <vec. réel> : tableau des fréquences trouvées.
- *XH* : <réel> : intervalle de temps entre 2 données (pas d'échantillonage).
- *T0* : <réel> : temps initial.
- *NTERM* : <entier> : nombre de frequences à rechercher.
- *TRESX* : <vec. réel> : partie réelle des résidus.
- *TRESY* : <vec. réel> : partie imaginaire des résidus.
- *FMIN1* : <réel> : fréquence minimale de la première fenêtre
- *FMAX1* : <réel> : fréquence maximale de la première fenêtre
- *NTERM1*: <entier> : nombre de fréquences à rechercher pour la première fenêtre.

Elle effectue l'analyse en fréquence des données du tableau (*TX*+i**TY*) et stocke les fréquences dans *F* et les amplitudes complexes dans *A*.

KTABS est arrondi à la plus proche valeur inférieure telle que KTABS = 6n+1 avec n entier naturel positif.

Elle emploie les variables globales `_naf_isec`, `_naf_iw`, `_naf_dtour`, `_naf_icplx` .

elle recherche une approximation de TX+iTY sous la forme

$$\sum_{l=0}^{size(F)} A[l]\exp(\imath F[l]\times t)$$

somme de l=0 à size(F) { A[l]*exp(i*F[l]*t) }

Dans ce cas, A est un vecteur numérique de complexes (<vec. complexe>).

Les triplets (*FMINx*, *FMAXx*, *NTERMx*) définissent une fenêtre. Seules les fréquences dans ces fenêtres sont recherchées.

```
Exemple :
Le fichier "tessin.out" contient des données
sur 32998 lignes avec une ligne d'entete.
tels que la colonne 2 contient la partie réelle des données.
la colonne 3 contient la partie imaginaire des données.
Le temps initial est 0 et le pas est de 0.01.
Nous recherchons 10 frequences.

> vnumR X,Y,F;
vnumC A;
read(tessin2.out, [1:32000],(X,2),(Y,3));
naftab(X,Y,A,F,32000,0.01, 0, 10);
B=abs(A);
Le premier argument de naf n'est pas multiple de 6.
Le premier argument de naf devient 31998.
CORRECTION DE  IFR = 1 AMPLITUDE  = 8.72192e-07
B     Tableau de reels : nb reels =6
> writes(F,B,A);
+9.999993149794888E-02  +1.000000E-01  +1.000E-01  +1.095970178673141E-06
-2.000000944035095E-01  +9.999999E-03  +9.999E-03  +1.312007532388499E-07
+3.000000832689856E-01  +1.000000E-03  +1.000E-03  -1.403292661135361E-08
-4.000000970924361E-01  +9.999995E-05  +9.999E-05  +1.695880029074994E-09
+4.999999805039361E-01  +1.000003E-05  +1.000E-05  +3.216502329016007E-11
-5.999999830866213E-01  +9.999979E-07  +9.999E-07  -1.796078221829677E-12
>
```

17.1.3 freqa

`freqa` [Commande]

`freqa(`*TFREQ*, *TFREQREF*, *TABCOMBI*, *TINDIC*, *ORDREMAX*, *EPSILON*, *TFREQRESIDU*`)`

avec :

- *TFREQ* : <vec. réel> : fréquences
- *TFREQREF* : <vec. réel> : fréquences fondamentales
- *TABCOMBI* : <identificateur> : tableau contenant les combinaisons
- *TINDIC* : <identificateur> : tableau de 0 et/ou de 1
- *ORDREMAX* : <entier positif> : ordre maximal de recherche des combinaisons
- *EPSILON* : <réel> : précision de la recherche
- *TFREQRESIDU* : <identificateur> : tableau de réels des fréquences résiduelles

Elle recherche pour chaque fréquence du tableau *TFREQ* une combinaison entière des fréquences fondamentales contenues dans *TFREQREF* avec une précision *EPSILON* et un ordre maximal de recherche *ORDREMAX*. Pour chaque fréquence de *TFREQ*, la recherche s'arrête dès qu'une combinaison est trouvée. La recherche s'effectue par ordre total croissant.

Si la combinaison est trouveé, alors

- *TINDIC*[...] = 1,
- *TABCOMBI*[][...] contient la combinaison
- *TFREQRESIDU*[...] contient la fréquence résiduelle

Si la combinaison n'est pas trouvée, alors

- *TINDIC*[...] =0,
- *TABCOMBI*[][...] =0,
- *TFREQRESIDU*[...] =0

En sortie, *TABCOMBI* est un tableau de `size(`*TFREQREF*`)` tableaux de `size(`*TFREQ*`)` entiers,

TINDIC est un tableau de `size(`*TFREQ*`)` entiers (0/1), <= *TFREQRESIDU* est un tableau de `size(`*TFREQ*`)` réels.

En sortie, tous les éléments de *TABCOMBI* vérifient

1<=sum(abs(*TABCOMBI*[i][]))<=*ORDREMAX*.

```
Exemple :
> freqfond=1,5;
> freqfond[1]=3.1354;
> freqfond[2]=5.452;
> freqfond[3]=7.888;
> freqfond[4]=11.111;
> freqfond[5]=19.777;

> freq=1,5;
> freq=freq*freqfond[1]+freq*freqfond[2]+freq*freqfond[3]+
>  freq*freqfond[4]+freq*freqfond[5];

> freqa(freq,freqfond,tabcomb, tabind,20,1E-6, freqres);

> %writesfreqa[[freq],[freqfond],[tabcomb],[tabind] , [freqres]];

> freqfond[1] = 15677/5000
> freqfond[2] = 1363/250
> freqfond[3] = 986/125
> freqfond[4] = 11111/1000
> freqfond[5] = 19777/1000
  Frequence               Frequence residuelle      combinaison
 ------------------------------------------------------------------
4.736340000000000E+01 +0.000000000000000E+00  1   1   1   1   1
9.472680000000000E+01 +0.000000000000000E+00  2   2   2   2   2
1.420902000000000E+02 +0.000000000000000E+00  3   3   3   3   3
1.894536000000000E+02 +0.000000000000000E+00  4   4   4   4   4
2.368170000000000E+02
```

17.1.4 freqareson

`freqareson` [Commande]

`freqareson(`*FREQ*, *TFREQREF*, *TABCOMBI*, *ORDREMAX*, *EPSILON*, *TFREQRESIDU*`)`

avec :

- *FREQ* : <réel> : fréquence
- *TFREQREF* : <vec. réel> : fréquences fondamentales
- *TABCOMBI* : <identificateur> : tableau contenant les combinaisons
- *ORDREMAX* : <entier positif> : ordre maximal de recherche des combinaisons
- *EPSILON* : <réel> : précision de la recherche
- *TFREQRESIDU* : <identificateur> : tableau de réels des fréquences résiduelles

Elle recherche pour la fréquence *FREQ* toutes les combinaisons entières des fréquences fondamentales contenues dans *TFREQREF* avec une précision *EPSILON* et un ordre maximal de recherche *ORDREMAX*.

En sortie, pour chaque combinaison trouveé,

- *TABCOMBI*[][...] contient la combinaison
- *TFREQRESIDU*[...] contient la fréquence résiduelle

En sortie, *TABCOMBI* est un tableau de `size(`*TFREQREF*`)` tableaux d'entiers,

En sortie, tous les éléments de *TABCOMBI* vérifient

sum(abs(*TABCOMBI*[i][]))<=*ORDREMAX*.

```
Exemple :
>vnumR freqfond;
>resize(freqfond,5);
>freqfond[1]=3.1354;
>freqfond[2]=5.452;
>freqfond[3]=2*freqfond[3];
>freqfond[4]=11.111;
>freqfond[5]=19.777;

>freq=2*freqfond[1]+freqfond[2]-freqfond[3]+freqfond[4]-freqfond[5];
>freqareson(freq,freqfond,tabcomb, 8,1E-6, freqres);
>writes("%+-g\t%+-g\t%+-g\t%+-g\t%+-g\n",tabcomb);
+2      -1      +0      +1      -1
+2      +1      -1      +1      -1
+2      -3      +1      +1      -1
```

17.1.5 sertrig

`sertrig` [Fonction]

`sertrig(`<vec. complexe> *TAMP*, <vec. réel> *TFREQ*, <variable> *VAR*`)`

`sertrig(`<vec. complexe> *TAMP*, <vec. réel> *TFREQ*, <variable> *VAR*, <réel> *FACT* `)`

avec :

- *TFREQ* : tableau de fréquences
- *TAMP* : tableau d'amplitudes
- *VAR* : variable
- *FACT* : réel

Les tableaux *TAMP* et *TFREQ* doivent être de même taille.

Elle réalise la somme de :

$$\sum_{j=1}^{size(TAMP)} TAMP[j] \times \exp^{(\imath TFREQ[j] \times VAR)}$$

ou

$$\sum_{j=1}^{size(TAMP)} TAMP[j] \times \exp^{(\imath 2\pi TFREQ[j] \times VAR/FACT)}$$

La partie exp(i*...) est représentée sous la forme d'une variable angulaire où l'argument est la fréquence et la phase est nulle. L'amplitude sera le coefficient de cette variable. Ces variables angulaires ont pour radical _Ex. Il y aura autant de variables angulaires créées qu'il y a de fréquences. La série contiendra autant de termes qu'il y a d'élément dans les tableaux.

```
Exemple :
Le F contient les frequences et le tableau A contient les amplitudes.
> writes(F,A);
+9.999993149794888E-02  +1.000000000456180E-01  +1.095970178673141E-06
-2.000000944035095E-01  +9.999999960679056E-03  +1.312007532388499E-07
+3.000000832689856E-01  +1.000000314882390E-03  -1.403292661135361E-08
-4.000000970924361E-01  +9.999995669530993E-05  +1.695880029074994E-09
+4.999999805039361E-01  +1.000003117384769E-05  +3.216502329016007E-11
-5.999999830866213E-01  +9.999979187109419E-07  -1.796078221829677E-12
> s=sertrig(A,F,t);
s(_EXt1,_EXt2,_EXt3,_EXt4,_EXt5,_EXt6) =
   (9.99997918710942E-07-i*1.79607822182968E-12)*_EXt6
 + (1.00000311738477E-05+i*3.21650232901601E-11)*_EXt5
 + (9.99999566953099E-05+i*1.69588002907499E-09)*_EXt4
 + (0.00100000031488239-i*1.40329266113536E-08)*_EXt3
 + (0.00999999996067906+i*1.3120075323885E-07)*_EXt2
 + (0.100000000045618+i*1.09597017867314E-06)*_EXt1
```

17.2 Transformee de Fourier

`fft` [Commande]

`fft` (*TX*, *TY*, *TAMP*, *TFREQ*, *XH*, *T0*, *DTOUR*, *IW*)

avec :

- *TX* : <vec. réel> : partie réelle du signal
- *TY* : <vec. réel> : partie imaginaire du signal
- *TAMP* : <vec. complexe> : amplitude trouvée
- *TFREQ* : <vec. réel> : fréquence trouvée
- *XH* : <réel> : pas d'échantillonnage
- *T0* : <réel> : temps initial
- *DTOUR* : <réel> : longueur d'un tour
- *IW* : <réel> : flag pour indiquer la présence de fenêtre (pour les valeurs, (voir Chapitre 3 [_naf_iw], page 9)).

Elle effectue la transformée de fourier rapide de *TX*+i**TY* et stocke les amplitudes et fréquences déterminées dans les tableaux *TAMP* et *TFREQ*.

Elle ne retient que les "N" permiers éléments de *TX* et *TY* tel que "N" soit la plus grande puissance de 2 contenue dans la taille du tableau *TX*.

Remarque: *TX* et *TY* doivent avoir la même taille.

```
Exemple :
>vnumC z;
 vnumR freq;
 xin=vnumR[7.:5.:6.:9.];
 yin=vnumR[0:0:0:0];
 fft(xin,yin,z,freq,1,0,pi,0);
 writes(z,freq);
> -2.500000000000000E-01  +0.000000000000000E+00  -1.570796326794897E+00
+2.499999999999993E-01  -1.000000000000000E+00  -7.853981633974483E-01
+6.750000000000000E+00  +0.000000000000000E+00  +0.000000000000000E+00
+2.500000000000007E-01  +1.000000000000000E+00  +7.853981633974483E-01
-2.500000000000000E-01  +0.000000000000000E+00  +1.570796326794897E+00
```

17.3 Transformee de Fourier Inverse

`ifft` [Commande]

`ifft` (*TAMP*, *TX*, *TY*, *XH*, *T0*)

avec :

- *TAMP* : <vec. complexe> : amplitude fournie
- *TX* : <vec. réel> : partie réelle du signal
- *TY* : <vec. réel> : partie imaginaire du signal
- *XH* : <réel> : pas d'échantillonnage
- *T0* : <réel> : temps initial

Elle effectue la transformée de fourier rapide inverse de *TAMP* et stocke le résultat dans *TX*+i**TY*.

TAMP doit etre un tableau à 2**N+1 éléments. En sortie, *TX* et *TY* ont 2**N éléments.

```
Exemple :
>vnumC z;
 vnumR freq;
 xin=vnumR[7.:5.:6.:9.];
 yin=vnumR[0:0:0:0];
 fft(xin,yin,z,freq,1,0,pi,0);
 ifft(z,xout,yout,1,0);
 writes(xout,yout);
> +7.000000000000000E+00    +0.000000000000000E+00
+5.000000000000000E+00  +0.000000000000000E+00
+6.000000000000000E+00  +0.000000000000000E+00
+9.000000000000000E+00  +0.000000000000000E+00
```

17.4 Integration numerique de séries ou de macros

`integnum` [Commande]

`integnum(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *FICHIER*,
`{ REAL ,(`*VAR1*, *SER1*, *VI1*`),...},`
`{ COMPLEX ,(`*VAR2*, *SER2*, *VI2*`),...});`

`integnum(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *FICHIER*,
`{ REAL ,(`*VAR1*, *SER1*, *VI1*`),...},`
`{ COMPLEX ,(`*VAR2*, *SER2*, *VI2*`),...},`

RESTARTSTATE);

avec :

- *T0* : <réel> : temps initial
- *TF* : <réel> : temps final
- *PAS* : <réel> : pas de sortie dans le fichier
- *PREC* : <réel> : précision
- *DOPRIMAX* : <réel> : pas maximum de l'intégrateur
- *PASDOPRI* : <réel> : pas de l'intégrateur
- *FICHIER* : <nom fichier> : fichier de resultat
- *VAR* : <variable> : variable
- *SER* : <série> : série ou opération (membre droit)
- *VI* : <constante> : valeur initiale
- *RESTARTSTATE* : <nom> : etat précédent de l'intégrateur.

Elle effectue l'intégration numérique des séries stockées dans les listes avec les valeurs initiales et stocke le résultat dans le fichier de résultat. L'intégrateur utilisé dépend de la variable globale `_integnum` (voir Section 3.11 [_integnum], page 13).

Les équations sont des triplets de la variable derivée, de la série et de la valeur initiale.

Pour intégrer un système dépendant du temps , noté `t` par exemple, il faut ajouter une équation au système sous la forme `(t,1,0)` et utiliser la variable `t` comme temps dans les autres équations du système.

Si *RESTARTSTATE* est spécifié mais n'existe pas, les conditions initiales utilisées sont celles du systèmes d'équations. Si *RESTARTSTATE* existe alors les conditions initiales utilisées sont celles contenues dans *PREVSTATE* et *T0* est ignoré. A la fin de l'intégration, *RESTARTSTATE* est mis à jour. *RESTARTSTATE* permet un redémarrage de l'intégration. Il est possible de changer le pas d'intégration, selon l'intégrateur.

`integnum` [Fonction]

<tableau de vec. réel> = integnum(*T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*,
{ REAL ,(*VAR1*, *SER1*, *VI1*),...},
{ COMPLEX ,(*VAR2*, *SER2*, *VI2*),...});

Tous les arguments sont identiques à la forme précédente, excepté *FICHIER* qui n'est pas présent.

Elle effectue l'intégration numérique des séries stockées dans les listes avec les valeurs initiales et retourne le résultat dans un tableau de vecteurs numériques de réels. L'intégrateur utilisé dépend de la variable globale `_integnum` (voir Section 3.11 [_integnum], page 13).

`integnum` [Commande]

integnum(*T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *FICHIER*,
{ REAL ,(*TVAR1*, *TSER1*, *TVI1*),...},
{ COMPLEX ,(*TVAR2*, *TSER2*, *TVI2*),...});

avec :

- *TVAR* : <tableau de variables> : tableau des variables dérivées
- *TSER* : <tableau> : tableau de séries (membre droit)
- *TVI* : <vec. num.> : tableau ou vecteur numérique de valeurs initiales

Tous les arguments sont identiques à la première forme, excepté *TVAR*, *TSER* et *TVI*. Les équations sont des triplets de tableau de variables derivées, tableau de séries et tableau de valeurs initiales.

`integnum` [Commande]

`<tableau de vec. réel> = integnum(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*,
`{ REAL ,(`*TVAR1*, *TSER1*, *TVI1*`),...},`
`{ COMPLEX ,(`*TVAR2*, *TSER2*, *TVI2*`),...});`

avec :

- *TVAR* : <tableau de variables> : tableau de variables
- *TSER* : <tableau> : tableau de séries (membre droit)
- *TVI* : <vec. num.> : tableau ou vecteur numériques de valeurs initiales

Tous les arguments sont identiques à la deuxième forme, excepté *TVAR*, *TSER* et *TVI*. Les équations sont des triplets de tableau de variables derivées, tableau de séries et tableau de valeurs initiales.

```
Exemple :
Nous souhaitons integrer le système :  { dx/dt = -y , dy/dt= x }
avec les valeurs initiales x=1 et y=0
entre 0 et 2*pi avec un pas de pi/10 et une précision de 1E-12.
Le pas de l'integrateur sera pi/20 et le pas maximum sera pi/10.
Les resultats seront stockes dans le fichier icossin.out
La solution est : x = cos(t) et y = sin(t)

>integnum(0,2*pi, pi/10, 1E-12, pi/10, pi/20,
          icossin.out, REAL, (x, -y, 1),(y,x, 0));

Le fichier icossin.out contient :
time                    x                       y
+0.000000000000000E+00 +1.000000000000000E+00 +0.000000000000000E+00
+3.141592653589793E-01 +9.510565162951542E-01 +3.090169943749471E-01
+6.283185307179586E-01 +8.090169943749489E-01 +5.877852522924729E-01
+9.424777960769379E-01 +5.877852522924752E-01 +8.090169943749478E-01
+1.256637061435917E+00 +3.090169943749500E-01 +9.510565162951550E-01
+1.570796326794897E+00 +2.567125951231441E-15 +1.000000000000003E+00
+1.884955592153876E+00 -3.090169943749455E-01 +9.510565162951576E-01
+2.199114857512855E+00 -5.877852522924724E-01 +8.090169943749527E-01
+2.513274122871834E+00 -8.090169943749485E-01 +5.877852522924791E-01
+2.827433388230814E+00 -9.510565162951568E-01 +3.090169943749536E-01
+3.141592653589793E+00 -1.000000000000005E+00 +5.403107187863445E-15
+3.455751918948772E+00 -9.510565162951610E-01 -3.090169943749436E-01
+3.769911184307752E+00 -8.090169943749564E-01 -5.877852522924716E-01
+4.084070449666731E+00 -5.877852522924830E-01 -8.090169943749488E-01
+4.398229715025710E+00 -3.090169943749571E-01 -9.510565162951583E-01
+4.712388980384690E+00 -8.301722685091165E-15 -1.000000000000008E+00
+5.026548245743669E+00 +3.090169943749417E-01 -9.510565162951644E-01
+5.340707511102648E+00 +5.877852522924709E-01 -8.090169943749602E-01
+5.654866776461628E+00 +8.090169943749494E-01 -5.877852522924867E-01
+5.969026041820607E+00 +9.510565162951601E-01 -3.090169943749604E-01
+6.283185307179586E+00 +1.000000000000011E+00 -1.085302505620242E-14
```

```
Exemple :
> // Au lieu de stocker le resultat dans un fichier,
> // on stocke le resultat dans un tableau de vecteurs q.
> p = pi/10;
```

```
p =         0.3141592653589793
> q = integnum(0, 2*pi, p, 1E-12, p, p/2,REAL,(x,-y,1),(y,x,0));

q [1:3 ] nb elements = 3

> stat(q);
Tableau de series
 q [ 1:3 ]
 liste des elements du tableau :
q [ 1 ] =
Vecteur numerique q de 21 reels  double-precision.
taille en octets du tableau: 168
q [ 2 ] =
Vecteur numerique q de 21 reels  double-precision.
taille en octets du tableau: 168
q [ 3 ] =
Vecteur numerique q de 21 reels  double-precision.
taille en octets du tableau: 168
> writes(q);
+0.0000000000000000E+00 +1.0000000000000000E+00 +0.0000000000000000E+00
+3.1415926535897931E-01 +9.5105651629515420E-01 +3.0901699437494706E-01
+6.2831853071795862E-01 +8.0901699437494889E-01 +5.8778525229247280E-01
+9.4247779607693793E-01 +5.8778525229247525E-01 +8.0901699437494767E-01
+1.2566370614359172E+00 +3.0901699437495000E-01 +9.5105651629515486E-01
+1.5707963267948966E+00 +2.6090241078691179E-15 +1.0000000000000027E+00
+1.8849555921538759E+00 -3.0901699437494545E-01 +9.5105651629515764E-01
+2.1991148575128552E+00 -5.8778525229247236E-01 +8.0901699437495267E-01
+2.5132741228718345E+00 -8.0901699437494845E-01 +5.8778525229247902E-01
+2.8274333882308138E+00 -9.5105651629515675E-01 +3.0901699437495334E-01
+3.1415926535897931E+00 -1.0000000000000053E+00 +5.1625370645069779E-15
+3.4557519189487724E+00 -9.5105651629516097E-01 -3.0901699437494379E-01
+3.7699111843077517E+00 -8.0901699437495633E-01 -5.8778525229247180E-01
+4.0840704496667311E+00 -5.8778525229248280E-01 -8.0901699437494901E-01
+4.3982297150257104E+00 -3.0901699437495678E-01 -9.5105651629515842E-01
+4.7123889803846897E+00 -7.9658502016854982E-15 -1.0000000000000080E+00
+5.0265482457436690E+00 +3.0901699437494201E-01 -9.5105651629516430E-01
+5.3407075111026483E+00 +5.8778525229247114E-01 -8.0901699437496011E-01
+5.6548667764616276E+00 +8.0901699437494945E-01 -5.8778525229248657E-01
+5.9690260418206069E+00 +9.5105651629516008E-01 -3.0901699437496022E-01
+6.2831853071795862E+00 +1.0000000000000107E+00 -1.0685896612017132E-14
>
```

`integnum` [Commande]

`integnum(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *FICHIER*, *MACRO*, *VI* `);`

avec :

- *T0* : <réel> : temps initial
- *TF* : <réel> : temps final
- *PAS* : <réel> : pas de sortie dans le fichier
- *PREC* : <réel> : précision
- *DOPRIMAX* : <réel> : pas maximum de l'intégrateur
- *PASDOPRI* : <réel> : pas de l'intégrateur

- *FICHIER* : <nom fichier> : fichier de resultat
- *MACRO* : <macro> : nom d'une macro
- *VI* : <vec. num.> : vecteur numérique de valeurs initiales

Elle effectue l'intégration numérique du système défini par la macro avec le vecteur de valeurs initiales et stocke le résultat dans le fichier de résultat. L'intégrateur utilisé dépend de la variable globale `_integnum` (voir Section 3.11 [_integnum], page 13).

La macro doit être définie de la façon suivante : `macro nom [N,T,Y,DY] {... }`

Elle effectuera l'evaluation de la dérivée de Y au temps T (DY=dY/dt=f(Y,T))

Cette macro sera appelée avec les arguments suivants : avec :

- N : dimension du système
- T : temps où doit être évalué le second membre.
- Y : un vecteur numérique de N éléments
- DY : un vecteur numérique de N éléments qui doit être initialisé par la macro. En sortie, il contiendra l'evaluation de la dérivée de Y au temps T (dy/dt=f(Y,T)).

La macro peut accéder à des identificateurs globaux.

`integnum` [Fonction]

`<tableau de vec. réel> = integnum(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *MACRO*, *VI* `);`

Tous les arguments sont identiques à la forme précédente, excepté *FICHIER* qui n'est pas présent.

Elle effectue l'intégration numérique du système défini par la macro avec le vecteur de valeurs initiales et retourne le résultat dans un tableau de vecteurs numériques de réels. L'intégrateur utilisé dépend de la variable globale `_integnum` (voir Section 3.11 [_integnum], page 13).

```
Exemple :
Nous souhaitons integrer le système :  { dx/dt = -y , dy/dt= x }
avec les valeurs initiales x=1 et y=0
entre 0 et 2*pi avec un pas de pi/10 et une précision de 1E-12.
Le pas de l'integrateur sera pi/20 et le pas maximum sera pi/10.
Les resultats seront stockes dans le fichier icossin.out
La solution est : x = cos(t) et y = sin(t)

 macro fcn [N,X,Y,DY]
 {
  DY[1]=-Y[2]$
  DY[2]=Y[1]$
 };

 vnumR v;
 resize(v,2);
 v[1]=1;
 v[2]=0;

 integnum(0,2*pi, pi/10, 1E-12, pi/10, pi/20, output.dat, fcn, v);

Le fichier icossin.out contient :
time                    x                       y
+0.000000000000000E+00 +1.000000000000000E+00 +0.000000000000000E+00
```

```
+3.141592653589793E-01 +9.510565162951542E-01 +3.090169943749471E-01
+6.283185307179586E-01 +8.090169943749489E-01 +5.877852522924729E-01
+9.424777960769379E-01 +5.877852522924752E-01 +8.090169943749478E-01
+1.256637061435917E+00 +3.090169943749500E-01 +9.510565162951550E-01
+1.570796326794897E+00 +2.567125951231441E-15 +1.000000000000003E+00
+1.884955592153876E+00 -3.090169943749455E-01 +9.510565162951576E-01
+2.199114857512855E+00 -5.877852522924724E-01 +8.090169943749527E-01
+2.513274122871834E+00 -8.090169943749485E-01 +5.877852522924791E-01
+2.827433388230814E+00 -9.510565162951568E-01 +3.090169943749536E-01
+3.141592653589793E+00 -1.000000000000005E+00 +5.403107187863445E-15
+3.455751918948772E+00 -9.510565162951610E-01 -3.090169943749436E-01
+3.769911184307752E+00 -8.090169943749564E-01 -5.877852522924716E-01
+4.084070449666731E+00 -5.877852522924830E-01 -8.090169943749488E-01
+4.398229715025710E+00 -3.090169943749571E-01 -9.510565162951583E-01
+4.712388980384690E+00 -8.301722685091165E-15 -1.000000000000008E+00
+5.026548245743669E+00 +3.090169943749417E-01 -9.510565162951644E-01
+5.340707511102648E+00 +5.877852522924709E-01 -8.090169943749602E-01
+5.654866776461628E+00 +8.090169943749494E-01 -5.877852522924867E-01
+5.969026041820607E+00 +9.510565162951601E-01 -3.090169943749604E-01
+6.283185307179586E+00 +1.000000000000011E+00 -1.085302505620242E-14

Au lieu de stocker le resultat dans un fichier,
on stocke le resultat dans un tableau de vecteurs q.

q = integnum(0,2*pi, pi/10, 1E-12, pi/10, pi/20, fcn, v);
```

17.5 Integration numerique d'une fonction externe

`integnumfcn` [Commande]

`integnumfcn(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *FICHIER*, *LIBRAIRIE*,
`{ REAL ,(`*VAR1*, *VI1*`),...},`
`{ COMPLEX ,(`*VAR2*, *VI2*`),...});`

avec :

- *T0* : <réel> : temps initial
- *TF* : <réel> : temps final
- *PAS* : <réel> : pas de sortie dans le fichier
- *PREC* : <réel> : précision
- *DOPRIMAX* : <réel> : pas maximum de l'intégrateur
- *PASDOPRI* : <réel> : pas de l'intégrateur
- *FICHIER* : <nom fichier> : fichier de resultat
- *LIBRAIRIE* : <nom fichier> : nom de la librairie dynamique (sans l'extension .so, .dll ou .dylib) où se situe les fonctions integfcnbegin, integfcn et integfcnend.
- *VAR* : <(tableau de) variables> : variable ou tableau de variables
- *VI* : <constante ou vec. num.> : valeur initiale ou vecteur de valeurs initiales

Elle effectue l'intégration numérique de la fonction integfcn localisée dans la librairie dynamique externe *LIBRAIRIE* en utilisant les valeurs initiales et stocke le résultat dans le fichier de résultat. L'intégrateur utilisé dépend de la variable globale `_integnum` (voir Section 3.11 [_integnum], page 13).

La variable derivée ne sert qu'à mettre un nom dans le fichier de résultat.

`integnumfcn` [Fonction]

`<tableau de vec. réel> = integnumfcn(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *LIBRAIRIE*,

`{ REAL ,(`*VAR1*, *VI1*`),...},`

`{ COMPLEX ,(`*VAR2*, *VI2*`),...});`

Tous les arguments sont identiques à la forme précédente, excepté *FICHIER* qui n'est pas présent.

Elle effectue l'intégration numérique de la fonction integfcn localisée dans la librairie dynamique externe *LIBRAIRIE* en utilisant les valeurs initiales et retourne le résultat dans un tableau de vecteurs numériques de réels. L'intégrateur utilisé dépend de la variable globale `_integnum` (voir Section 3.11 [_integnum], page 13).

Le nom des variables est ignorée par cette fonction.

Les fonctions de la librairie dynamique doivent avoir le prototype suivant :

- void integfcnbegin(const int * neq, void **data);
  - neq : la dimension du système.
  - *data peut être initialisé par cette fonction et le pointeur sera ensuite transmis à integfcn et integfcnend.

  Cette fonction est appelée avant le démarrage de l'intégration.
- int integfcn(const int * neq, const double *t, const double *y, double *yp, void* data);
  - neq : la dimension du système.
  - *t : le temps où doit être évalué le second membre.
  - y : un tableau de neq réels et contient la solution au temps (*t)
  - dy : un tableau de neq réels et doit être initialisé par integfcn. En sortie, il contiendra l'evaluation de la dérivée de y au temps t (dy/dt=f(y,t)).
  - data : la valeur transmise par integfcnbegin.

  Cette fonction doit initialiser dy tel que dy/dt=f(y,t) et doit retourner 1.
- void integfcnend(void *data);
  - data : la valeur transmise par integfcnbegin.

  Cette fonction est appelée à la fin de l'intégration.

Pour créer une librairie dynamique à partir d'un fichier source écrit en langage C, il faut généralement le compiler de la façon suivante :

- sous linux avec gcc : gcc -fPIC -shared fcn.c -o libfcn.so
- sous linux avec intel c++ : icc -fPIC -shared fcn.c -o libfcn.so
- sous MacOS X avec gcc : gcc -shared -dynamiclib fcn.c -o libfcn.dylib
- sous MacOS X avec intel c++ : icc -fPIC -dynamiclib fcn.c -o libfcn.dylib

```
Exemple :
La librairie libfcn.(so,dylib,dll) contient le résultat
```

```
de la compilation du fichier source fcn.c suivant :
#include <math.h>
void integfcnbegin(const int * neq, void **data)
{
}

int integfcn(const int * neq, const double *x, const double *y,
             double *yp, void* data)
{
 yp[0] = sqrt(1.E0+cos(y[1]));
 yp[1] = y[0]+y[1];
 return 1;
}

void integfcnend(void *data)
{
}

Pour intégrer le système { dx/dt = sqrt(1+cos(y)) , dy/dt = x+y }
avec comme condition initiale x=1 et y=0
et pour stocker le résultat dans le fichier output.dat :

> integnumfcn(0,2*pi, pi/10, 1E-12, pi/10, pi/20, output.dat,
              libfcn, REAL, (x, 1),(y,0));

Pour intégrer le même système
en stockant le resultat dans le tableau de tableaux numériques q :
> q = integnumfcn(0,2*pi, pi/10, 1E-12, pi/10, pi/20,
                  libfcn, REAL, (x, 1),(y,0));
```

`integnumfcn` [Commande]

`integnumfcn(` *T0*, *TF*, *PAS*, *PREC*, *DOPRIMAX*, *PASDOPRI*, *FICHIER*, *LIBRAIRIE*,
`{ REAL ,(`*VAR1*, *VI1*`),...}`,
`{ COMPLEX ,(`*VAR2*, *VI2*`),...}` ,
`evalnum, REAL ,`*TB*, *TB_0*, *TB_PAS*,*TF*, *TF_0*, *TF_PAS* `)`

avec :

- *T0* : <réel> : temps initial
- *TF* : <réel> : temps final
- *PAS* : <réel> : pas de sortie dans le fichier
- *PREC* : <réel> : précision
- *DOPRIMAX* : <réel> : pas maximum de l'intégrateur
- *PASDOPRI* : <réel> : pas de l'intégrateur
- *FICHIER* : <nom fichier> : fichier de resultat
- *LIBRAIRIE* : <nom fichier> : nom de la librairie dynamique (sans l'extension .so, .dll ou .dylib) où se situe les fonctions integfcnbegin, integfcn et integfcnend.
- *VAR* : <(tableau de) variables> : variable ou tableau de variables
- *VI* : <constante ou vec. num.> : valeur initiale ou vecteur de valeurs initiales

- *TB* : <tableau de vec. réel> : tableau de données tabulées (temporelles) pour les temps passés
- *TB_0* : <réel> : temps initial de TB
- *TB_PAS* : <réel> : pas de temps entre deux données de TB
- *TF* : <tableau de vec. réel> : tableau de données tabulées (temporelles) pour les temps futurs (compris entre *T0* et *TF*)
- *TF_0* : <réel> : temps initial de TF
- *TF_PAS* : <réel> : pas de temps entre deux données de TF

Elle effectue l'intégration numérique de la fonction integfcn localisée dans la librairie dynamique externe *LIBRAIRIE* en utilisant les valeurs initiales et les données tabulées. Elle stocke le résultat dans le fichier de résultat.

Elle fournit à la fonction integfcn les données tabulées *TB* ou *TF* pour le temps auquel est évalué le second membre. Par exemple, si le second membre doit être évalué au temps t, alors elle extrait de *TB* ou *TF* les données corespondantes au temps t. Pour le démarrage de la méthode d'ADAMS, elle interpole les données de *TB* et fournit ces valeurs interpolées à la fonction integfcn. Pour les temps entre *T0* et *TF*, elle extrait directement les données sans interpolation. Par exemple, si *TB* et *TF* sont des tableaux de 4 vecteurs de 1000 réels, la fonction integfcn sera appelée avec un vecteur de 4 nombres réels.

L'intégrateur requiert que la variable globale `_integnum` doit être égale à ADAMS (voir Section 3.11 [_integnum], page 13) . Elle requiert que *TB_0* et *TF_0* doivent être égaux à *T0*. De plus, *PAS* doit être un multiple de *TB_PAS* et *TF_PAS*.

La variable derivée ne sert qu'à mettre un nom dans le fichier de résultat.

Les fonctions de la librairie dynamique doivent avoir le prototype suivant :

- void integfcnbegin(const int * neq, void **data);
  - neq : la dimension du système.
  - *data peut être initialisé par cette fonction et le pointeur sera ensuite transmis à integfcnend.

  Cette fonction est appelée avant le démarrage de l'intégration.
- int integfcn(const int * neq, const double *t, const double *y, double *yp, const double* tabdata);
  - neq : la dimension du système.
  - *t : le temps où doit être évalué le second membre.
  - y : un tableau de neq réels et contient la solution au temps (*t)
  - dy : un tableau de neq réels et doit être initialisé par integfcn. En sortie, il contiendra l'evaluation de la dérivée de y au temps t (dy/dt=f(y,tabdata(t),t)).
  - tabdata : tableau de données issus de TB ou TF au temps (*t).

  Cette fonction doit initialiser dy tel que dy/dt=f(y,tabdata(t),t) et doit retourner 1.
- void integfcnend(void *data);
  - data : la valeur transmise par integfcnbegin.

  Cette fonction est appelée à la fin de l'intégration.

Pour créer une librairie dynamique à partir d'un fichier source écrit en langage C, il faut généralement le compiler de la façon suivante :

- sous linux avec gcc : gcc -fPIC -shared fcn.c -o libfcn.so

- sous linux avec intel c++ : icc -fPIC -shared fcn.c -o libfcn.so
- sous MacOS X avec gcc : gcc -shared -dynamiclib fcn.c -o libfcn.dylib
- sous MacOS X avec intel c++ : icc -fPIC -dynamiclib fcn.c -o libfcn.dylib

```
Exemple :
La librairie libfcn.(so,dylib,dll) contient le résultat
de la compilation du fichier source fcn.c suivant :
#include <math.h>
void integfcnbegin(const int * neq, void **data)
{
}

int integfcn(const int * neq, const double *x, const double *y,
             double *yp, const double *tabdata)
{
 yp[0] = sqrt(1.E0+cos(y[1])*tabdata[0]);
 yp[1] = y[0]*tabdata[1]+y[1];
 return 1;
}

void integfcnend(void *data)
{
}

Pour intégrer le système
{ dx/dt = sqrt(1+cos(y)*sin(a*t)) , dy/dt = cos(b*t)*x+y }
avec a=0.5, b=0.9 et comme condition initiale x=1 et y=0
et pour stocker le résultat dans le fichier output.dat :
> t0 = 0;
> tf = 2*pi;
> pas=pi/10;
> vnumR tabb[1:2], tabf[1:2];
> t=-30*pas,t0,pas;
> a=0.5; b=0.9;
> tb[1]=sin(a*t); tb[2]=cos(b*t);
> t=t0, tf+2*pas,pas;
> tf[1]=sin(a*t); tf[2]=cos(b*t);
>
> integnumfcn(t0,tf, pas, 1E-12,pas, pas/2, output.dat,
              libfcn, REAL, (x, 1),(y,0),
              evalnum, REAL, tb, t0, -pas, tf, t0, pas);
```

17.6 Integral

`integral` [Fonction]

`integral(`<vec. num.> *TY*, <réel> *XH*, <entier> *ORDRE*, <identificateur> *N*`)`

`integral(`<vec. num.> *TY*, <réel> *XH*, <entier> *ORDRE*, <identificateur> *N*, `"kahan" )`

avec :

- *TY* : vecteur numérique des valeurs

- *XH* : pas entre chaque valeur du tableau
- *ORDRE* : ordre d'integration de 1 à 6
- *N* : indice du dernier élément utilisé dans TY

Elle calcule et retourne l'intégrale, en utilisant la formule de Newton-Cotes, d'ordre *ORDRE*, du vecteur numérique *TY* dont les éléments sont espacés de *XH*.

Elle retourne dans *N* l'indice du dernier élément utilisé pour calculer l'intégrale. La valeur de *N* est donnée par : *N*=int((size(*TY*)-1)/*ORDRE*)*ORDRE+1

Si l'option "kahan" est mise, la sommation est effectuée en utilisant la méthode de Kahan (sommation compensée).

Référence : Pour les formules de Newton-Cotes, voir le livre "Introduction to Numerical Analysis", chapitre integration, de Stoer et Burlisch.

Kahan, William (January 1965), "Further remarks on reducing truncation errors", Communications of the ACM 8 (1): 40, `http://dx.doi.org/10.1145%2F363707.363723`

```
Exemple :
> t=0,10000;
t    Tableau de reels : nb reels =10001
> t=t*pi/10000;
> X=-cos(t)+1;
> for j=1 to 6 { s=integral(sin(t),pi/10000,j,N); delta=s-X[N];};
s = 1.99999998355066
delta = -1.64493392240672E-08
s = 1.99999999999999
delta = -7.54951656745106E-15
s = 1.99999995065198
delta = 5.55111512312578E-15
s = 2
delta = -4.44089209850063E-16
s = 2
delta = 8.88178419700125E-16
s = 1.99999921043176
delta = 5.77315972805081E-15
```

17.7 Iteration numerique

`iternum` [Commande]

`iternum(` *nbiteration*, *passortie*, *fichierresultat* ,
{ `REAL ,(` *variable1*, *serie1*, *valeurinitiale1*`),...`},
{ `COMPLEX ,(` *variable2*, *serie2*, *valeurinitiale2*`),...`});

avec :

- *nbiteration* : <entier> : nombre d'itérations
- *passortie* : <entier> : pas de sortie
- *fichierresultat* : <nom fichier> : fichier contenant le résultat
- *variable1* : <variable> : variable a iterer
- *serie1* : <opération> : serie (membre droit)
- *valeurinitiale1* : <constante> : valeur initiale de la variable

Elle effectue *nbiteration* iterations sur les équations spécifiées dans les listes. Elle écrit dans le fichier de résultat tous les *passortie* itérations.

Les équations sont des triplets de la variable itérée, de la série et de sa valeur initiale.

La valeur initiale peut être un nombre complexe ou réel.

```
Exemple :
> iternum(10,1,sortie.txt,REAL,(un,un+2,3));
Le fichier "sortie.txt" contient :
time                    un
+0.000000000000000E+00 +3.000000000000000E+00
                     1 +5.000000000000000E+00
                     2 +7.000000000000000E+00
                     3 +9.000000000000000E+00
                     4 +1.100000000000000E+01
                     5 +1.300000000000000E+01
                     6 +1.500000000000000E+01
                     7 +1.700000000000000E+01
                     8 +1.900000000000000E+01
                     9 +2.100000000000000E+01
                    10 +2.300000000000000E+01

> iternum(5,1,sortie.txt,REAL,(vn,un-1,3),(un,vn+un,2));
Le fichier "sortie2.txt" contient :
time                    vn                      un
+0.000000000000000E+00 +3.000000000000000E+00 +2.000000000000000E+00
                     1 +1.000000000000000E+00 +5.000000000000000E+00
                     2 +4.000000000000000E+00 +6.000000000000000E+00
                     3 +5.000000000000000E+00 +1.000000000000000E+01
                     4 +9.000000000000000E+00 +1.500000000000000E+01
                     5 +1.400000000000000E+01 +2.400000000000000E+01
```

17.8 Interpolation

`interpol` [Commande]

`interpol(LINEAR ,` *TX*, *DTX*, *TY*`)`

`interpol(QUADRATIC ,` *TX*, *DTX*, *TY*`)`

`interpol(SPLINE ,` *TX*, *DTX*, *TY*`)`

`interpol(HERMITE ,` *TX*, *DTX*, *TY*, *DEG*`)`

avec :

- TX : <vec. réel> : tableau des temps où sont connus *DTX*.
- DTX : <vec. réel> : tableau des valeurs connus *DTX*=f(*TX*).
- TY : <vec. réel> : tableau des temps où seront calculés les valeurs.
- DEG : <entier> : degré maximal du polynôme d'interpolation.

Elle effectue l'interpolation linéaire, spline, ou quadratique ou d'hermite. Elle retourne un vecteur de réels des valeurs interpolées aux temps *TY* à partir des données (*TX*, *DTX*).

Remarque : Elle suppose que *TX* et *TY* sont triés par ordre croissant ou décroissant.

Si *TY*[i] n'appartient pas à l'intervalle [*TX*[0], *TX*[size(*TX*)]], alors *DTY*[i] est extrapolé.

L'algorithme de l'interpolation d'hermite est décrite dans le livre de Stoer et Burlish, 1993, "Introduction to Numerical Analysis, Chap 2, pages 52-57".

```
Exemple :
> x1=0,2*pi,2*pi/59;
> x2=0,3,0.5;
> sl=interpol(LINEAR,x1,cos(x1),x2);
> writes("%.2E   %.4E %.4E\n",x2, cos(x2), sl);
0.00E+00 1.0000E+00 1.0000E+00
5.00E-01 8.7758E-01 8.7652E-01
1.00E+00 5.4030E-01 5.3958E-01
1.50E+00 7.0737E-02 7.0719E-02
2.00E+00 -4.1615E-01 -4.1576E-01
2.50E+00 -8.0114E-01 -8.0001E-01
3.00E+00 -9.8999E-01 -9.8920E-01
```

17.9 Changement de coordonnees

17.9.1 ell_to_xyz

`ell_to_xyz` [Commande]

`ell_to_xyz(` *ELL*, *MU*, *X*, *XP* `);`

avec :

- ELL : <tableau de vec. réel> : tableau de 6 vecteurs de réels contenant les éléments elliptiques.
  - ELL[1] : demi grand-axe
  - ELL[2] : longitude moyenne
  - ELL[3] : excentricite * cos(longitude du periphelie)
  - ELL[4] : excentricite * sin(longitude du periphelie)
  - ELL[5] : sin(inclinaison/2) * cos(longitude du noeud ascendant)
  - ELL[6] : sin(inclinaison/2) * sin(longitude du noeud ascendant)
- MU : <réel> : produit de la constante de gravitation et de la somme des masses.
- X : <tableau de vec. réel> : tableau de 3 vecteurs de réels contenant les positions.
- XP : <tableau de vec. réel> : tableau de 3 vecteurs de réels contenant les vitesses.

Elle convertit les éléments elliptiques *ELL* en positions *X* et vitesses *XP*.

```
Exemple :
/* calcul d'une seule valeur */
GM= 0.2959122082855911d-03;
mu = GM*1.05;
n=2*pi/221.6;
varpi = 138*pi/180;


vnumR ell[1:6];
resize(ell,1);
ell[1][1] = (mu/n**2)**(1./3.);
ell[2][1] = varpi;
ell[3][1] = 0.54*cos(varpi);
ell[4][1] = 0.54*sin(varpi);
ell[5][1] = 0.0;
ell[6][1] = 0.0;
```

```
ell_to_xyz(ell,mu,x,xp);
writes(x);
writes(xp);
```

Exemple :

```
/* calcul de plusieurs valeurs contenues dans un fichier */
mu = 1;

vnumR ell[1:6];
read(file1,ell);

ell_to_xyz(ell,mu,x,xp);

writes(x);
writes(xp);
```

17.9.2 xyz_to_ell

`xyz_to_ell` [Commande]

`xyz_to_ell(` *X*, *XP*, *MU*, *ELL* `);`

avec :

- X : <tableau de vec. réel> : tableau de 3 vecteurs de réels contenant les positions.
- XP : <tableau de vec. réel> : tableau de 3 vecteurs de réels contrnant les vitesses.
- MU : <réel> : produit de la constante de gravitation et de la somme des masses.
- ELL : <tableau de vec. réel> : tableau de 6 vecteurs de réels contenant les éléments elliptiques.
  - ELL[1] : demi grand-axe
  - ELL[2] : longitude moyenne
  - ELL[3] : excentricite * cos(longitude du periphelie)
  - ELL[4] : excentricite * sin(longitude du periphelie)
  - ELL[5] : sin(inclinaison/2) * cos(longitude du noeud ascendant)
  - ELL[6] : sin(inclinaison/2) * sin(longitude du noeud ascendant)

Elle convertit les positions *X* et vitesses *XP* en éléments elliptiques *ELL*.

Exemple :

```
mu = 1;

vnumR x[1:3],xp[1:3];
read(file2,x,xp);

xyz_to_ell(x,xp,mu,ell);

writes(ell);
```

18 Utilitaire

18.1 aide

`help` [Commande]

`help;`

`?;`

Elle affiche une aide.

Sous Unix et MacOS X, le programmes d'aide utilise le logiciel info (fourni en particulier avec Texinfo) pour afficher l'aide. Sous Windows, le fichier d'aide au format chm est affiché.

`help` [Commande]

`help <motreserve>;`

`? <motreserve>;`

Elle affiche l'aide du manuel de référence pour le mot réservé de TRIP. Cette fonction appelle le programme info (fourni en particulier avec Texinfo) sous Unix et MacOS X.

```
Exemple :
>  ? vartrip;
```

18.2 sortie

`exit` [Commande]

`quit;`

`exit;`

Cette commande arrête l'exécution de trip.

18.3 cls

`cls` [Commande]

`cls;`

Elle efface l'écran en cours (identique à la commande clear sous UNIX).

18.4 pause

`pause` [Commande]

`pause;`

Cette commande affiche un message et attend la frappe de la touche d'une touche. Si l'utilisateur tape `RET` alors l'exécution continue. Si l'utilisateur tape `q+RET` ou `e+RET` alors l'exécution s'arrete immédiatemment pour retourner au prompt.

`pause(<entier>` x`);`

Cette commande suspend l'excécution de TRIP pendant x secondes.

```
Exemple :
> for j=1 to 3 { j; pause; time_s; pause(2); time_t; };
1
Appuyez sur la touche 'return' pour continuer
ou la touche 'q' ou 'e' pour revenir au prompt ...

02.000s
```

```
2
Appuyez sur la touche 'return' pour continuer
ou la touche 'q' ou 'e' pour revenir au prompt ...

02.000s
3
Appuyez sur la touche 'return' pour continuer
ou la touche 'q' ou 'e' pour revenir au prompt ...

02.000s
```

18.5 msg

`msg` [Commande]

`msg` <chaine> ;

`msg(`<chaine> *textformat*`);`

Elle affiche le texte sans aucun formatage fourni à l'ecran, ce qui est utile pour les commentaires.

Le texte doit être une chaine ou un texte entre guillemets. Le texte peut être sur plusieurs lignes.

`msg` [Commande]

`msg(`<chaine> *textformat*`,` <réel> *x*`, ... );`

Elle affiche à l'ecran le texte formaté accompagné ou non de constantes réelles.

Les constantes réelles sont formatées. Le formatage est identique à celui de de la commande printf du langage C (voir Section 7.6 [str], page 33, pour les formats acceptés). Le texte doit être une chaine ou un texte entre guillemets. Le texte peut être sur plusieurs lignes.

Pour écrire un guillemet, il faut le doubler.

En mode numérique NUMRATMP seulement, les entiers ou les rationnels sont d'abord convertis en nombres réels double-précision avant l'affichage si le format est '%g', '%e' ou '%f'. Si le format est '%d' ou '%i', les entiers ou les rationnels sont écrits sans conversion.

La fonction `msg` (voir Section 7.7 [msg], page 34) a un comportement similaire mais écrit le résultat dans une chaine de caract 'eres.

```
Exemple :
> file="fichier1.dat"$
> msg"écriture du fichier "+file;
écriture du fichier fichier1.dat
> msg "je vais faire un guillemet :
""cette chaine est encadre par des guillemets"".";

je vais faire un guillemet :
"cette chaine est encadre par des guillemets".

> msg("pi=%g pi=%.8E\n",pi,pi);
pi=3.14159 pi=3.14159265E+00
```

18.6 error

`error` [Commande]

`error(`<chaine> *textformat*`);`

Elle génère une erreur en affichant le texte sans aucun formatage fourni à l'ecran.

Le texte doit être une chaine ou un texte entre guillemets. Le texte peut être sur plusieurs lignes.

```
Exemple :
> x=2;
x =                         2
> if (x==2) then { error("erreur : x=2"); };
TRIP[ 3 ] : erreur : x=2
Commande :'if (x==2) then { error("erreur : x=2"); };'
                            ^^^^^^^^^^^^^^^^^^^^^^
```

18.7 try

`try` [Commande]

`try { <corps> bodytry } catch { <corps> bodycatch };`

Normalement, elle exécute toutes les instructions de *bodytry*. Si une erreur se produit pendant cette exécution, alors elle ignore les instructions restantes de *bodytry* puis elle exécute les instructions de *bodycatch*.

Si la variable *_info* est égale à `on`, alors un message d'avertissement s'affiche lorsqu'une erreur se produit pendant l'exécution de *bodytry*. Si la variable *_info* est égale à `off`, alors aucun message d'avertissement ne s'affiche lorsqu'une erreur se produit pendant l'exécution de *bodytry*.

Si une erreur se produit pendant l'exécution des instructions de *bodycatch* alors une erreur est générée. Cette erreur peut être interceptée si la section try-catch est contenue dans une autre section try-catch.

Il est possible d'imbriquer une section try-catch au sein des instructions *bodytry* ou de *bodycatch*.

```
Exemple :
> // affiche un message mais poursuit l'execution
> _info on;
_info on
> b = 0;
b =                         0
> q = 2;
q =                            2
> try { s = "invalid"+q; b = 2; } catch { b = 1; };
TRIP[ 1 ] : Addition impossible
Commande :'try { s = "invalid"+q; b = 2; } catch { b = 1; };'
                     ^^^^^^^^^^^

 Information : Poursuite de l'execution (instruction catch)
b =                            1
>
> // n'affiche pas de message mais poursuit l'execution
> _info off;
_info off
> b = 0;
b =                         0
> q = 2;
q =                            2
> try { s = "invalid"+q;  b = 2;  } catch { b = 1; };
```

```
b =                          1
>
```

18.8 delete

`delete` [Commande]

`delete( <identificateur> );`

Elle supprime tout identificateur (série, variable, troncature, tableau, ...).

Remarque :

- Les dépendances de cet identificateur sont mis à 0.
- Les variables angulaires dépendant de cet identificateur sont converties en variable polynomiale.
- Les troncatures dépendant de cet identificateur sont supprimées.

```
Exemple :
> file="fichier1.dat";
file = "fichier1.dat"
> X=expi(x,1,0);
1*X

> delete(file);
> delete(x);
La variable angulaire X est convertie en variable polynomiale.
```

18.9 reset

`reset` [Commande]

`reset;`

Elle réinitialise complètement TRIP en remettant toutes les variables globales aux valeurs par défaut et elle supprime tous les identificateurs de la mémoire.

18.10 include

`include` [Commande]

`include` <nom fichier> ;

`include` <chaine> ;

Elle charge le fichier TRIP "fichier" et l'exécute.

Ce fichier doit se trouver dans le répertoire courant ou dans le répertoire spécifié par la variable `_path`.

Le fichier peut posséder une extension quelconque. L'extension recommandée étant `.t` .

Remarque : si le fichier est un identificateur de type chaine, alors elle charge le fichier dont le nom est le contenu de cette chaine.

```
Exemple :
>include ellip;
>include fct.t;
> file="fperplanumH";
file = "fperplanumH"
> include file; /*exécute le fichier "fperplanumH" */
```

18.11 @@

`@@` [Commande]

`@@;`

Elle réinitialise TRIP puis charge et réexécute le dernier fichier exécuté par la commande `include`.

18.12 vartrip

`vartrip` [Commande]

`vartrip;`

Elle affiche l'etat des variables globales de TRIP.

```
Exemple :
> vartrip;

    Etat des variables globales de Trip :

                _affc      =    6
                _affdist   =    0
                _affhomog  =    0
                _echo      =    0
                _path      =    /exemples/
                _history   =    /unixfiles/
                _hist      =    ON
                _naf_iprt  =    -1
                _naf_icplx =    1
                _naf_iw    =    1
                _naf_isec  =    1
                _naf_dtour =    6.283185307179586E+00
                _naf_nulin =    1
                _naf_tol   =    1.000000000000000E-10
                _time      =    OFF
                _comment   =    OFF
                _affvar    =    ( )
```

18.13 bilan

`bilan` [Commande]

`bilan;`

Elle liste toutes les identificateurs présents en mémoire en indiquant leur type:

SERIE	l'identificateur représente une série.
VARIABLE	l'identificateur est une variable.
CONST	l'identificateur est une constante.
MATRIXR	l'identificateur est une matrice numérique de réels.
MATRIXC	l'identificateur est une matrice numérique de complexes.
TAB	l'identificateur est un tableau.
TABVAR	l'identificateur est un tableau de variables.
VNUMR	l'identificateur est un vecteur numérique de réels.
VNUMC	l'identificateur est un vecteur numérique de complexes.
STRING	l'identificateur est une chaine de caractères.

EXTERNAL STRUCTURE	l'identificateur est une structure externe.
FILE	l'identificateur est un fichier.
REMOTE OBJECT	l'identificateur est un objet distant stocké sur un serveur SCSCP.
SCSCP CLIENT	l'identificateur est une connexion à un serveur SCSCP.

`bilan mem` [Commande]

`bilan mem;`

Elle liste pour chaque série ou tableau de série la place occupée en mémoire et le nombre de termes.

Le nombre total de termes et la mémoire totale occupée par ces séries sont également affichés.

```
Exemple :
> bilan;
A   VAR
ASR SERIE
ASRp    SERIE
RSA SERIE
X   VAR
_AsR    SERIE
_CosE   SERIE
_Expiv  SERIE
_RsA    SERIE
_RsASinv    SERIE
_SinE   SERIE
e   VAR
ep  VAR
g   VAR
ga  VAR
lp  VAR
x   VAR
> bilan mem;
          Nom                 Memoire (octets)              Nb de termes
--------------------       --------------------       --------------------
                 ASR                       1704                         64
                ASRp                       1704                         64
                 RSA                       2136                         81
                _AsR                       1704                         64
               _CosE                       2112                         80
              _Expiv                       1728                         64
                _RsA                       2136                         81
            _RsACosv                       2112                         80
           _RsAExpiv                       1728                         64
            _RsASinv                       2112                         80
               _SinE                       2112                         80
--------------------       --------------------       --------------------
                                          21288                        802
```

18.14 stat

`stat` [Commande]

`stat;`

Elle liste tous les identificateurs présents en mémoire. Elle les ordonne par leur type. Pour chaque identificateur, elle affiche une brve information. Pour les séries, elle affiche leur contenue.

```
Exemple :
> S = (1+x+y)**2$
> vnumR R$
> vnumC C$
> dim T[1:2]$
> stat;
Bilan des operations :
Series :
S(x,y) =
                          1
 +                        2*y
 +                        1*y**2
 +                        2*x
 +                        2*x*y
 +                        1*x**2


Variables :
Variable x type : 2     ordres :  2 2 2 2
 dependances :
 variables dependant de celle-ci :


Variable y type : 2     ordres :  3 3 3 3
 dependances :
 variables dependant de celle-ci :



Tableaux de series :

T [1:2 ] nb elements = 2


Tableaux de reels double-precision :
Vecteur numerique R de 0 reels  double-precision.

Tableaux de complexes double-precision :
Vecteur numerique C de 0 complexes double-precision.

>
```

`stat` [Commande]

`stat( <identificateur> );`

`stat( <identificateur> , "puismin" );`

`stat( <identificateur> , "puismax" );`

`stat( <identificateur> , "puismin" , "puismax" );`

Suivant le type d'identificateur, elle donne les informations sur le nombre de termes et l'espace occupé par l'identificateur.

Si l'option "puismin" est mise, elle donne le degré le plus bas d'une série par rapport à chaque variable.

Si l'option "puismax" est mise, elle donne le degré le plus élevé d'une série par rapport à chaque variable.

```
Exemple :
> s = (1+x+y)**2$
> stat(s);
 serie s (  x , y )
 nombre de variables : 2   taille du descripteur : 96 octets
 nombre de termes : 6   taille : 608 octets
> stat(s,"puismin","puismax");
 serie s (  x , y )
 nombre de variables : 2   taille du descripteur : 96 octets
 nombre de termes : 6   taille : 608 octets
 puismin : x ^  0 ,  y ^  0
 puismax : x ^  2 ,  y ^  2
```

18.15 save_env

`save_env` [Commande]

`save_env;`

Elle sauvegarde l'état des variables globales TRIP en utilisant un mécanisme de pile.

```
Exemple :
> _mode;
    _mode = POLP
> save_env;
> _mode=POLH;
    _mode = POLH
> rest_env;
> _mode;
    _mode = POLP
```

18.16 rest_env

`rest_env` [Commande]

`rest_env;`

Elle restaure l'état des variables globales TRIP sauvegardées par `save_env` en utilisant un mécanisme de pile.

```
Exemple :
> _path="/users/";
        _path = /users/
> save_env;
> _path="/users/toto/";
        _path = /users/toto/
> /*instructions utilisant le chemin specifique */;
```

```
> rest_env;
> _path;
        _path = /users/
```

18.17 random

`random` [Fonction]

`random(<entier> x)`

Elle retourne un nombre entier aléatoire compris entre 0 et x-1.

```
Exemple :
> random(10);
         1
> random(10);
         7
> random(10);
         0
> random(10);
         9
> random(10);
         8
```

18.18 nameof

`nameof` [Fonction]

`nameof(<identificateur> x)`

Elle retourne sous la forme d'une chaine le nom de l'objet.

Si l'objet est une expression, alors la chaine vide `""` est retournée.

```
Exemple :
> c=3$
> n1= nameof(c);
n1 = "c"
> s="abcde"$
> n2 = nameof(s);
n2 = "s"
>
```

18.19 typeof

`typeof` [Fonction]

`typeof(<identificateur> x)`

Elle retourne sous la forme d'une chaine le type de l'objet. Les chaines retournees sont decrites dans la commande `bilan` (voir Section 18.13 [bilan], page 203).

Si l'objet n'existe pas, alors la chaine vide `""` est retournée.

```
Exemple :
> typeof(2);
"CONST"
> typeof(1+x);
"SERIE"
> t=1,10;
```

```
t  Vecteur de reels double-precision : nb reels =10
> if (typeof(t)=="VNUMR") then { stat(t); };
Vecteur numerique t de 10 reels  double-precision.
taille en octets du tableau: 80
>
```

18.20 file_fullname

`file_fullname` [Fonction]

`file_fullname(`<chaine> *name*`)`

Elle construit un nom de fichier avec un chemin absolu. Toutes les utilisations ultérieures de la chaine retournée vont ignorer la variable *_path*.

```
Exemple :
> _path;
_path = /users/guest/data/
> vnumR t;
> fn=file_fullname("/tmp/mydata.txt");
fn  = nom de fichier '/tmp/mydata.txt'
> read(fn,t);
```

18.21 Temps d'execution

18.21.1 time_s

`time_s` [Commande]

`time_s;`

Elle initialise le temps de départ pour le calcul des fonctions `time_l` et `time_t`.

18.21.2 time_l

`time_l` [Commande]

`time_l;`

Elle affiche le temps CPU total en mode utilisateur consommé, le temps total écoulé et le temps CPU total en mode système consommé, depuis le dernier appel à `time_l`.

Si aucun appel à `time_l` n'a été effectué, alors c'est le temps écoulé depuis le dernier appel de `time_s`.

```
Exemple :
> time_s; for j=1 to 3 { (1+x+y+z+t+u+v)**18$ time_l; };
utilisateur 00.313s  - reel 00.183s  - systeme 00.057s  - (202.43% CPU)
utilisateur 00.307s  - reel 00.170s  - systeme 00.055s  - (213.11% CPU)
utilisateur 00.311s  - reel 00.173s  - systeme 00.055s  - (211.93% CPU)
```

18.21.3 time_t

`time_t` [Commande]

`time_t;`

Elle affiche le temps CPU total en mode utilisateur consommé, le temps total écoulé et le temps CPU total en mode système consommé, depuis le dernier appel à `time_s`.

`time_t` [Commande]

`time_t(`<identificateur> *usertime* `,` <identificateur> *realtime* `);`

Elle affiche le temps CPU total en mode utilisateur consommé, le temps total écoulé et le temps CPU total en mode système consommé, depuis le dernier appel à `time_s`. *usertime* contiendra la somme du temps CPU en mode utilisateur et système écoulé et *realtime* contient le temps réel écoulé.

```
Exemple :
> time_s; for j=1 to 3 { (1+x+y+z+t+u+v)**18$ time_t; };
  time_t(usertime, realtime);
utilisateur 00.310s  - reel 00.178s  - systeme 00.055s  - (205.29% CPU)
utilisateur 00.615s  - reel 00.347s  - systeme 00.111s  - (209.44% CPU)
utilisateur 00.928s  - reel 00.519s  - systeme 00.168s  - (211.23% CPU)
utilisateur 00.928s  - reel 00.519s  - systeme 00.168s  - (211.20% CPU)
> usertime;
usertime =                1.097452
> realtime;
realtime =      0.5196029999999999
```

18.22 Appel au shell

`!` [Commande]

`! <chaine> ;`

Elle exécute la commande shell contenue dans la chaine.

```
Exemple :
> dir ="ls -al";
dir = "ls -al"
> ! dir;
total 8136
drwxr-sr-x  20 xxxx   ttt         1536 Sep 27 17:16 .
drwxr-sr-x  22 xxxx   ttt          512 Sep 06 12:16 ..
-rw-r-----   1 xxxx   ttt          281 Sep 08 1998  .Guidefaults
-rw-------   1 xxxx   ttt          204 Sep 27 10:46 .Xauthority
```

`!` [Commande]

`str(! <chaine> );`

Elle exécute la commande shell contenue dans la chaine et retourne la sortie standard dans une chaine. Si la sortie d'erreur n'est pas vide, une erreur est générée.

```
Exemple :
> s=str(!"uname");
s = "Darwin
"
```

19 References

- GMP - GNU Multiple Precision Arithmetic Library
  version 6.0.0, 2014
  Torbjorn Granlund et al.
  `http://www.gmplib.org/`
- LTDL - GNU Libtool - The GNU Portable Library Tool
  version 2.4.6, 2014
  `http://www.gnu.org/software/libtool/`
- MPFR: A multiple-precision binary floating-point library with correct rounding
  2007, ACM Trans. Math. Softw. 33, 2 (Jun. 2007) 13.
  Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P.
  DOI= `http://doi.acm.org/10.1145/1236463.1236468`
  `http://www.mpfr.org`
- MPC - A library for multiprecision complex arithmetic with exact rounding
  Version 1.0.3, February 2015
  Andreas Enge and Mickaël Gastineau and Philippe Théveny and Paul Zimmermann
  `http://mpc.multiprecision.org/`
- Symbolic Computation Software Composability Protocol (SCSCP) specification
  Version 1.3, 2009
  S.Freundt, P.Horn, A.Konovalov, S.Linton, D.Roozemond.
  `http://www.symcomp.org/scscp`
- OpenMath content dictionary scscp1
  D. Roozemond
  `http://www.win.tue.nl/SCIEnce/cds/scscp1.html`
- OpenMath content dictionary scscp2
  D. Roozemond
  `http://www.win.tue.nl/SCIEnce/cds/scscp2.html`
- SCSCP C Library - A C/C++ library for Symbolic Computation Software Composibility Protocol
  Version 1.0.2, May 2016
  M. Gastineau
  `http://www.imcce.fr/trip/scscp/`

Annexe A Dictionnaires OpenMath

Voici la liste des symboles OpenMath supportés par le serveur SCSCP et le client SCSCP.

CD	**Symbol**
arith1	abs, divide, minus, plus, power, times, unary_minus
alg1	one, zero
bigfloat1	bigflat
arith1	abs, divide, minus, plus, power, times, unary_minus
complex1	argument, complex_cartesian, complex_polar, conjugate, imaginary, real
fieldname1	C, Q, R
interval1	interval_cc, integer_interval, interval
linalg2	matrix, matrixrow, vector
list1	list
logic1	not, and, xor, or, true, false
nums1	e,i, infinity, NaN, pi, rational
polyd1	poly_ring_d_named, SDMP, DMP, term
polyu	poly_u_rep, term
polyr	poly_r_rep, term
setname1	C, N, P, Q, R, Z
transc1	arccos, arccosh, arcsin, arcsinh, arctan, arctanh, cos, cosh, exp, ln, log, sin, sinh, tan, tanh

Voici la liste des symboles OpenMath supportés seulement par le serveur SCSCP.

CD	**Symbol**
scscp2	get_allowed_heads, get_transient_cd, get_signature, store, retrieve, unbind

Voici la liste des symboles OpenMath supportés seulement par le client SCSCP.

CD	**Symbol**
scscp2	symbol_set, signature, service_description,

Annexe B Index des Variables globales

Annexe C Index des commandes

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Annexe D Index complet

(Index n'existe pas)

Table des matières