Manuel de référence de Maxima

par W. Schelter et al.

Traduction française : A. Jaccomard, A. Osselet, D. Duparc et M. Lavaud.
Version du 03 mai 2003. La version la plus récente de ce document peut être obtenue sur le site web de l'association AsTeX, `http://www.univ-orleans.fr/EXT/ASTEX`

MAXIMA est un programme de calcul formel assez complet.

Il a été réalisé en COMMON LISP par William F. Schelter, à partir de la version originale de Macsyma réalisée au MIT et qui fut distribuée par le Department of Energy. Le DOE ayant accordé la permission à W. Schelter d'en faire des copies dérivées, et en particulier de le distribuer sous license publique GNU (voyez le fichier COPYING inclus dans la distribution), les fichiers de MAXIMA peuvent désormais être redistribués selon les termes de la licence publique GNU.

1 Introduction à MAXIMA

Lancez MAXIMA avec la commande "maxima". MAXIMA va afficher son numéro de version et une invite. Terminez chaque commande de MAXIMA par un point-virgule. Terminez la session avec la commande "quit();". Voici un exemple de session:

```
sonia$ maxima
GCL (GNU Common Lisp)  Version(2.3) Tue Mar 21 14:15:15 CST 2000
Licensed under GNU Library General Public License
Contains Enhancements by W. Schelter
Maxima 5.4 Tue Mar 21 14:14:45 CST 2000 (enhancements by W. Schelter)
Licensed under the GNU Public License (see file COPYING)
(C1) factor(10!);

                                  8  4  2
(D1)                             2  3  5  7
(C2) expand((x+y)^6);

           6        5       2  4       3  3       4  2      5      6
(D2)      y  + 6 x y  + 15 x  y  + 20 x  y  + 15 x  y  + 6 x  y + x
(C3) factor(x^6-1);

                                     2           2
(D3)               (x - 1) (x + 1) (x  - x + 1) (x  + x + 1)
(C4) quit();

sonia$
```

MAXIMA peut rechercher les pages d'information. Utilisez la commande ***describe*** pour afficher toutes les commandes et variables contenues dans une chaîne, et optionnellement leur documentation:

```
(C1) describe(factor);


 0: DONTFACTOR :(maxima.info)Definitions for Matrices and ..
 1: EXPANDWRT_FACTORED :Definitions for Simplification.
 2: FACTOR :Definitions for Polynomials.
 3: FACTORFLAG :Definitions for Polynomials.
 4: FACTORIAL :Definitions for Number Theory.
 5: FACTOROUT :Definitions for Polynomials.
 6: FACTORSUM :Definitions for Polynomials.
 7: GCFACTOR :Definitions for Polynomials.
 8: GFACTOR :Definitions for Polynomials.
 9: GFACTORSUM :Definitions for Polynomials.
 10: MINFACTORIAL :Definitions for Number Theory.
 11: NUMFACTOR :Definitions for Special Functions.
 12: SAVEFACTORS :Definitions for Polynomials.
 13: SCALEFACTORS :Definitions for Miscellaneous Options.
 14: SOLVEFACTORS :Definitions for Equations.
Enter n, all, none, or multiple choices eg 1 3 : 2 8;
```

```
Info from file /d/linux/local/lib/maxima-5.4/info/maxima.info:
 - Function: FACTOR (EXP)
     factors the expression exp, containing any number of variables or
     functions, into factors irreducible over the integers.
     FACTOR(exp, p) factors exp over the field of integers with an
     element adjoined whose minimum polynomial is p.  FACTORFLAG[FALSE]
     if FALSE suppresses the factoring of integer factors of rational
     expressions.  DONTFACTOR may be set to a list of variables with
     respect to which factoring is not to occur.  (It is initially
     empty).  Factoring also will not take place with respect to any
     variables which are less important (using the variable ordering
     assumed for CRE form) than those on the DONTFACTOR list.
     SAVEFACTORS[FALSE] if TRUE causes the factors of an expression
     which is a product of factors to be saved by certain functions in
     order to speed up later factorizations of expressions containing
     some of the same factors.  BERLEFACT[TRUE] if FALSE then the
     Kronecker factoring algorithm will be used otherwise the Berlekamp
     algorithm, which is the default, will be used.  INTFACLIM[1000] is
     the largest divisor which will be tried when factoring a bignum
     integer.  If set to FALSE (this is the case when the user calls
     FACTOR explicitly), or if the integer is a fixnum (i.e.  fits in
     one machine word), complete factorization of the integer will be
     attempted.  The user's setting of INTFACLIM is used for internal
     calls to FACTOR. Thus, INTFACLIM may be reset to prevent MACSYMA
     from taking an inordinately long time factoring large integers.
     NEWFAC[FALSE] may be set to true to use the new factoring routines.
     Do EXAMPLE(FACTOR); for examples.


 - Function: GFACTOR (EXP)
     factors the polynomial exp over the Gaussian integers (i.  e.
     with SQRT(-1) = %I adjoined).  This is like FACTOR(exp,A**2+1)
     where A is %I.
          (C1)  GFACTOR(X**4-1);
          (D1)        (X - 1) (X + 1) (X + %I) (X - %I)
(D1)                                FALSE
```

Pour utiliser un résultat dans un calcul ultérieur, vous pouvez l'affecter à une variable ou y faire référence en lui donnant automatiquement une étiquette. De plus, % fait référence au résultat de calcul le plus récent:

```
(C2) u:expand((x+y)^6);

           6       5       2  4       3  3       4  2      5      6
(D2)      y  + 6 x y  + 15 x  y  + 20 x  y  + 15 x  y  + 6 x  y + x
(C3) diff(u,x);

                5         4       2  3       3  2       4        5
(D3)         6 y  + 30 x y  + 60 x  y  + 60 x  y  + 30 x  y + 6 x
(C4) factor(d3);
```

```
                                                5
(D4)                                  6 (y + x)
```

MAXIMA connaît les nombres complexes et les constantes numériques:

```
(C6) cos(%pi);

(D6)                                  - 1

(C7) %e^(%i*%pi);

(D7)                                  - 1
```

MAXIMA peut exécuter du calcul différentiel et intégral:

```
(C8) u:expand((x+y)^6);

           6        5       2  4       3  3       4  2      5      6
(D8)      y  + 6 x y  + 15 x  y  + 20 x  y  + 15 x  y  + 6 x  y + x
(C9) diff(%,x);

                5         4       2  3       3  2       4        5
(D9)         6 y  + 30 x y  + 60 x  y  + 60 x  y  + 30 x  y + 6 x
(C10) integrate(1/(1+x^3),x);

                                          2 x - 1
                      2             ATAN(-------)
                   LOG(x  - x + 1)        SQRT(3)    LOG(x + 1)
(D10)            - --------------- + ------------- + ----------
                          6             SQRT(3)          3
```

MAXIMA peut résoudre des systèmes linéaires et des équations cubiques:

```
(C11) linsolve( [ 3*x + 4*y = 7, 2*x + a*y = 13], [x,y]);

                                7 a - 52        25
(D11)                    [x = --------, y = -------]
                                3 a - 8       3 a - 8
(C12) solve( x^3 - 3*x^2 + 5*x = 15, x);

(D12)              [x = - SQRT(5) %I, x = SQRT(5) %I, x = 3]
```

MAXIMA peut résoudre des ensembles non liléaires d'équations. Notez que si vous ne voulez pas afficher un résultat, vous pouvez terminer votre commande avec un *$* au lieu d'un *;*.

```
(C13) eq1: x^2 + 3*x*y + y^2 = 0$

(C14) eq2: 3*x + y = 1$

(C15) solve([eq1, eq2]);

                3 SQRT(5) + 7      SQRT(5) + 3
(D15) [[y = - -------------, x = -----------],
```

```
                    2                2

                                3 SQRT(5) - 7        SQRT(5) - 3
                           [y = -------------, x = - -----------]]
                                      2                    2
```

Sous le système X window, MAXIMA peut générer des tracés d'une ou plusieurs fonctions:

```
(C13) plot2d(sin(x)/x,[x,-20,20]);


(YMIN -3.0 YMAX 3.0 0.29999999999999999)
(D13)                                0
(C14) plot2d([atan(x), erf(x), tanh(x)], [x,-5,5]);


(YMIN -3.0 YMAX 3.0 0.29999999999999999)
(YMIN -3.0 YMAX 3.0 0.29999999999999999)
(YMIN -3.0 YMAX 3.0 0.29999999999999999)
(D14)                                0
(C15) plot3d(sin(sqrt(x^2+y^2))/sqrt(x^2+y^2),[x,-12,12],[y,-12,12]);

(D15)                                0
```

Placer le curseur au coin haut gauche de la fenêtre de tracé fait surgir un menu qui, parmi d'autres choses, vous permettra de créer un fichier PostScript du tracé (il sera placé par défaut dans votre répertoire personnel). Vous pouvez appliquer une rotation à un tracé en 3D.

2 Aide

2.1 Introduction à l'aide

La plus utile des commandes de l'aide en ligne est DESCRIBE qui fournit de l'aide sur toutes les commandes contenant une chaîne particulière. Ici par commande nous entendons un opérateur intégré tel que INTEGRATE ou FACTOR etc. Comme raccourci vous pouvez taper `? fact` au lieu de `describe("fact")`

```
(C3) ? inte;


 0: (maxima.info)Integration.
 1: Introduction to Integration.
 2: Definitions for Integration.
 3: INTERRUPTS.
 4: ASKINTEGER :Definitions for Simplification.
 5: DISPLAY_FORMAT_INTERNAL :Definitions for Input et Output.
 6: INTEGERP :Definitions for Miscellaneous Options.
 7: INTEGRATE :Definitions for Integration.
 8: INTEGRATION_CONSTANT_COUNTER :Definitions for Integration.
 9: INTERPOLATE :Definitions for Numerical.
Enter n, all, none, or multiple choices eg 1 3 : 7 8;

Info from file /d/linux2/local/share/info/maxima.info:
 - Function: INTEGRATE (EXP, VAR)
     integrates exp with respect to var or returns an integral
     expression (the noun form) if it cannot perform the integration
     (see note 1 below).  Roughly speaking three stages are used:
...
```

Dans cet exemple l'utilisateur demande les items 7 et 8. Notez le `;` qui suit les deux nombres. Il aurait pu taper `all` pour avoir de l'aide sur tous les items.

2.2 Lisp et Maxima

Tout Maxima est évidemment écrit en lisp. Il existe une convention pour nommer fonctions et variables: tous les symboles commençant par un signe "$" au niveau lisp, sont lus avec le "$" enlevé au niveau Macsyma. Par exemple, il y a deux fonctions lisp TRANSLATE et $TRANSLATE. Si au niveau macsyma vous entrez TRANSLATE(FOO); la fonction qui sera appelée est $translate. Pour accéder à l'autre fonction vous devez la préfixer par un "?". Notez que vous *ne* pouvez *pas* mettre un espace après le `?` puisque cela indiquerait que vous demandez de l'aide!

```
(C1) ?TRANSLATE(FOO);
```

Bien sûr, ceci peut ne pas faire ce que vous vouliez puisque c'est une fonction complètement différente.

Pour entrer une commande lisp vous pouvez utiliser

```
(C1) :lisp (foo 1 2)
```

ou pour obtenir une invite lisp utilisez `to_lisp();`, ou encore tapez `Ctrl-c` pour entrer en mode débogage. Ceci provoquera l'entrée dans une boucle lisp break. Vous pouvez maintenant évaluer $d2 et voir la valeur de la ligne étiquettée D2, dans son format lisp interne. Taper `:q` quittera le niveau haut, si vous êtres dans une boucle "break" de débogage. Si vous avez quitté maxima avec `to_lisp();` alors vous pourriez taper

```
MAXIMA>(run)
```

à l'invite lisp, pour redémarrer la session Maxima.

Si vous avez l'intention d'écrire des fonctions lisp à appeler au niveau macsyma vous devriez les nommer par des noms commençant par un "$". Notez que tous les symboles tapés au niveau lisp sont automatiquement lus en majuscules, sauf si vous faites quelque chose comme `|$odeSolve|` pour forcer le respect de la casse. Maxima interprète les symboles en casse mixte, si le symbole a déjà été lu ou lorsqu'il a été lu pour la première fois il n'y avait pas un symbole portant les mêmes lettres mais en majuscules. Ainsi si vous tapez

```
(C1) Integrate;
(D1) INTEGRATE
(C2) Integ;
(D2) Integ
```

Le symbole `Integrate` existe déjà en majuscules puisque c'est une primitive de Maxima, mais pas INTEG, qui n'existe pas encore, donc Integ est permis. Cela peut sembler un peu bizarre, mais nous voulons que l'ancien code de Maxima fonctionne encore, ce qui fait que les primitives de Maxima peuvent être en minuscules ou en majuscules. Un avantage de ce système est que si vous tapez en minuscules, vous verrez immédiatement quelles sont les fonctions et mots-clé de maxima.

Pour entrer des formes de Maxima au niveau lisp, vous pouvez utiliser la macro `#$`.

```
(setq $foo #$[x,y]$)
```

Cela aura le même effet que d'entrer

```
(C1)FOO:[X,Y];
```

sauf que foo n'apparaîtra pas dans la liste VALUES. Afin de voir foo au format affiché de macsyma vous pouvez taper

```
(displa $foo)
```

Dans cette documentation lorsque nous voulons nous référer à un symbole macsyma nous omettrons en général $ juste comme vous le feriez en entrant au niveau macsyma. Cela provoquera des confusions lorsque nous voudrons aussi nous référer à un symbole lisp. Dans ce cas nous essaierons d'utiliser les minuscules pour le symbole lisp et les majuscules pour le symbole macsyma. Par exemple LIST pour $list et list pour le symbole lisp dont le nom affiché est "list".

Comme les fonctions définies en langage MAXIMA ne sont pas des fonctions lisp ordinaires, vous devez utiliser mfuncall pour les appeler. Par exemple:

```
(D2)                    FOO(X, Y) := X + Y + 3
```

puis au niveau lisp

```
CL-MAXIMA>>(mfuncall '$foo 4 5)
12
```

Un certain nombre de fonctions lisp sont masquées dans le package maxima, car leur utilisation dans maxima n'est pas compatible avec leur définition en tant que fonction système. Par exemple typep se comporte différemment en common lisp et en Maclisp. Si vous voulez vous référer à la zeta typep de lisp dans le package maxima vous devrez utiliser global:typep (ou cl:typep en common lisp). Ainsi

```
(macsyma:typep '(1 2)) ==> 'list
(lisp:typep '(1 2))==> error (lisp:type-of '(1 2))==> 'cons
```

Pour voir quels symboles sont masqués regardez dans "src/maxima-package.lisp" ou faites un describe du package au niveau lisp.

2.3 Ramasse-miettes

Le calcul symbolique a tendance à créer beaucoup de résultats intermédiaires qui encombrent inutilement la mémoire, et il peut être nécessaire de les éliminer pour que certains programmes puissent être exécutés avec succès.

Sous GCL, sur les systèmes UNIX où l'appel système mprotect est disponible (y compris sous SUN OS 4.0 et certaines variantes de BSD) un ramasse-miettes stratifié est disponible. Ce qui limite le ramassage aux pages le plus récemment écrites. Voyez la documentation GCL sous ALLOCATE et GBC. Au niveau lisp faire (setq si::*notify-gbc* t) vous aidera à déterminer quelles zones vont avoir besoin de plus d'espace.

2.4 Documentation

La source de la documentation est dans '`.texi`' au format texinfo. \'A partir de ce format nous pouvons produire les fichiers info utilisés par les commandes en ligne `?` et `describe`. Des fichiers `html` et `pdf` peuvent aussi être produits.

Il y a de plus des exemples qui vous permettent de faire

```
example(integrate);
(C4) example(integrate);
(C5) test(f):=BLOCK([u],u:INTEGRATE(f,x),RATSIMP(f-DIFF(u,x)));
(D5) test(f) := BLOCK([u], u :
         INTEGRATE(f, x), RATSIMP(f - DIFF(u, x)));
(C6) test(SIN(x));
(D6)                                                    0
(C7) test(1/(x+1));
(D7)                                                    0
(C8) test(1/(x^2+1));
(D8)                                                    0
(C9) INTEGRATE(SIN(X)^3,X);
...
```

2.5 Définitions pour l'aide

DEMO (*file*) Fonction

est la même chose que BATCH mais fait une pause après chaque ligne de commande et continue lorsqu'un espace est tapé (vous pouvez devoir taper ; suivi d'un saut de ligne, si vous êtes sous xmaxima). Les fichiers de demo ont le suffixe `.dem`

DESCRIBE (*cmd*) Fonction

Cette commande affiche la documentation sur toutes les commandes qui contiennent la sous-chaîne "cmd". Ainsi

```
(C1) describe("integ");
 0: (maxima.info)Integration.
 1: Introduction to Integration.
 2: Definitions for Integration.
 3: ASKINTEGER :Definitions for Simplification.
..
Enter n, all, none, or multiple choices eg 1 3 : 2 3;
Info from file /d/linux2/local/share/info/maxima.info:
Definitions for Integration
===========================

 - Function: CHANGEVAR (EXP,F(X,Y),Y,X)
...
```

voir ⟨undefined⟩ [Introduction à l'aide], page ⟨undefined⟩

EXAMPLE (*command*) Fonction

démarre une démonstration sur la façon dont command travaille sur certaines expressions. Après chaque ligne de commande elle fait une pause et attend qu'un espace soit tapé, comme avec la commande DEMO.

3 Ligne de commande

3.1 Introduction à la ligne de commande

%TH (*i*) Fonction

est le i-ème calcul précédent: si l'expression suivante à calculer est D(j), %TH est D(j-i). C'est utile dans les fichiers BATCH ou pour faire référence à un groupe d'expressions D. Par exemple, si SUM est initialisée à 0 alors FOR I:1 THRU 10 DO SUM:SUM+%TH(I) donnera à SUM la somme des dix dernières expressions D.

"'" operator

- (apostrophe, guillemet simple) a pour effet d'empêcher l'évaluation. E.g. '(F(X)) signifie "ne pas évaluer l'expression F(X)". 'F(X) signifie "retourner la forme nominale de F appliquée à [X]".

"”" operator

- (deux apostrophes) provoque une évaluation supplémentaire. E.g. ”c4; ré-exécutera la ligne C4. ”(F(X)) signifie "évaluer l'expression F(X) une fois de plus". ”F(X) signifie "retourner la forme verbale de F appliquée à [X]".

3.2 Définitions pour la ligne de commande

ALIAS (*nouveaunom1, anciennom1, nouveaunom2, anciennom2, ...*) Fonction

fournit un nom alternatif (utilisateur ou système) pour une fonction, une variable, un tableau, etc. Un nombre pair quelconque d'arguments peut être utilisé.

DEBUG () Fonction

LISPDEBUGMODE(); DEBUGPRINTMODE(); et DEBUG(); donnent à l'utilisateur les possibilités de débogageutilisées par les programmeurs système. Ces outils sont puissants, et bien que certaines conventions soient différentes de celles du niveau macsyma usuel il semble que leur utilisation soit très intuitive. [Certains affichages peuvent être verbeux pour des terminaux lents, mais il y a des commutateurs pour contrôler cela.] Ces commandes ont été conçues pour un utilisateur devant déboguer du code macsyma traduit, comme telles elles sont précieuses. Voyez MACDOC;TRDEBG USAGE pour plus d'information. Consultez GJC pour davantage d'aide.

DEBUGMODE Variable

défaut: [FALSE] - fait entrer MACSYMA dans une boucle d'interruption dès qu'une erreur MACSYMA se produit si TRUE et termine ce mode si FALSE. Si fixé à ALL alors l'utilisateur peut examiner BACKTRACE qui contient la liste des fonctions actuellement entrées.

DEBUGPRINTMODE () Fonction

LISPDEBUGMODE(); DEBUGPRINTMODE(); et DEBUG(); donnent à l'utilisateur les possibilités de débogageutilisées par les programmeurs système. Ces outils sont puissants, et bien que certaines conventions soient différentes de celles du niveau macsyma usuel il semble que leur utilisation soit très intuitive. [Certains affichages peuvent être verbeux pour des terminaux lents, mais il y a des commutateurs pour contrôler cela.] Ces commandes ont été conçues pour un utilisateur devant déboguer du code macsyma traduit, comme telles elles sont précieuses. Voyez MACDOC;TRDEBG USAGE pour plus d'information. Consultez GJC pour davantage d'aide.

EV (*exp, arg1, ..., argn*) Fonction

est l'une des plus puissantes commandes de MACSYMA, et l'une des plus souples. Elle évalue l'expression exp dans l'environnement spécifié par l'argi. S'éxécute par étapes, comme ceci:

- (1) L'environnement est d'abord défini en parcourant les argi qui peuvent être: SIMP simplifie exp sans tenir compte du positionnement du commutateur SIMP qui empêche les simplifications si FALSE. NOEVAL suprime la phase d'évaluation de EV (voir le pas (4) ci-dessous). Ceci est utile conjointement avec les autres commutateurs et en provoquant la re-simplification de exp sans sa ré-évaluation. EXPAND provoque le développement. EXPAND(m,n) provoque le développement, en fixant les valeurs de MAXPOSEX et MAXNEGEX à m et n respectivement. DETOUT provoque la mise à l'écart du déterminant inverse de tout inverse de matrice calculé dans exp plutôt que sa division par chaque élément. DIFF provoque l'exécution de toutes les différentiations indiquées dans exp. DERIVLIST(var1,...,vark) provoque seulement les différentiations par rapport aux variables indiquées. FLOAT convertit les nombres rationnels non entiers en nombres réels (à virgule flottante). NUMER évalue en virgule flottante certaines fonctions mathématiques (y compris l'exponentiation) à arguments numériques. Il fait remplacer les variables par leur valeur dans exp. Il active aussi le commutateur FLOAT. PRED provoque l'évaluation des prédicats (expressions qui s'évaluent à TRUE ou FALSE). EVAL provoque une post-évaluation supplémentaire de exp (voir le pas (5) ci-dessous). E où E est un atome déclaré comme étant un EVFLAG lie E à TRUE pendant l'évaluation de exp. V:expression (ou encore V=expression) lie V à la valeur de expression pendant l'évaluation de exp. Notez que si V est une option MACSYMA, alors expression est utilisée pour sa valeur pendant l'évaluation de exp. Si plus d'un argument de EV est de ce type alors les liaisons se font en parallèle. Si V est une expression non atomique alors une substitution plutôt qu'une liaison est accomplie. E où E, un nom de fonction, a été déclarée être une EVFUN, applique E à exp. Tout autre nom de fonction (e.g. SUM) provoque l'évaluation des occurrences de ces noms dans exp comme s'ils étaient des verbes. De plus une fonction se produisant dans exp (disons F(args)) peut être définie localement dans le but de cette évaluation de exp en donnant F(args):=corps comme argument à EV. Si un atome non mentionné ci-dessus ou une variable indicée ou une expression indicée était donné en argument, il serait évalué et si le résultat est une équation ou une affectation alors la liaison indiquée ou la substitution est exécutée. Si le résultat est une liste alors les membres de

la liste sont traités comme s'ils étaient des arguments additionnels donnés à EV. Ceci permet de donner une liste d'équations (e.g. [X=1, Y=A**2]) ou une liste de noms d'équations (e.g. [E1,E2] où E1 et E2 sont des équations) comme celle retournée par SOLVE. Les argi de EV peuvent être donnés en ordre quelconque sauf pour la substitution d'équations qui sont traitées en séquence, de gauche à droite, et de EVFUNS qui sont composés, e.g. EV(exp,RATSIMP,REALPART) est traité comme REALPART(RATSIMP(exp)). Les commutateurs SIMP, NUMER, FLOAT, et PRED peuvent aussi être définis localement dans un bloc, ou globalement au "nveau haut" de MACSYMA de sorte qu'ils soient actifs juqu'à leur remise à zéro. Si exp est sous la forme CRE alors EV retournera un résultat sous forme CRE fournie par les commutateurs NUMER et FLOAT qui ne sont pas TRUE tous deux.

- (2) Pendant le pas (1), une liste est établie des variables non indicées apparaissant sur la partie gauche des équations dans les argi ou dans la valeur de certains argi si cette valeur est une équation. Les variables (aussi bien les variables indicées qui n'ont pas de fonctions de tableau associées, que les variables non indicées) dans l'expression exp sont remplacées par leurs valeurs globales, sauf pour celles apparaissant dans cette liste. En général, exp est juste une étiquette ou % (comme dans (C2) ci-dessous), donc ce pas récupère simplement l'expression nommée par l'étiquette, de sorte que EV puisse travailler sur elle.
- (3) Si des substitutions sont indiquées par les argi, elles sont faites maintenant.
- (4) L'expression résultante est alors ré-évaluée (sauf si l'un des argi était NOEVAL) et simplifiée conformément aux argi. Notez que tout appel de fonction dans exp sera exécuté après que les variables qu'il contient aient été évaluées et que EV(F(X)) puisse alors se comporter comme F(EV(X)).
- (5) Si l'un des argi était EVAL, les pas (3) et (4) sont répétés.

```
                        Exemples

(C1) SIN(X)+COS(Y)+(W+1)**2+'DIFF(SIN(W),W);
                           d                  2
(D1)  COS(Y) + SIN(X) + -- SIN(W) + (W + 1)
                           dW
(C2) EV(%,SIN,EXPAND,DIFF,X=2,Y=1);
                      2
(D2)       COS(W) + W  + 2 W + COS(1) + 1.90929742
```

Une syntaxe alternative de haut niveau a été fournie pour EV, par laquelle il suffit d'entrer ses arguments, sans le EV(). C'est à dire qu'on peut écrire simplement

```
exp, arg1, ...,argn.
```

Ce n'est pas permis en tant que partie d'une autre expression, i.e. dans les fonctions, les blocs, etc.

```
(C4) X+Y,X:A+Y,Y:2;
(D4)                 Y + A + 2
(Notez le procédé de liaison en parallèle)
(C5) 2*X-3*Y=3$
(C6) -3*X+2*Y=-4$
(C7) SOLVE([D5,D6]);
```

```
SOLUTION
                                    1
(E7)                          Y =  - -
                                    5

                               6
(E8)                        X = -
                               5
(D8)                     [E7, E8]
(C9) D6,D8;
(D9)                     - 4 =  - 4
(C10) X+1/X > GAMMA(1/2);
                          1
(D10)                X + - > SQRT(%PI)
                          X
(C11) %,NUMER,X=1/2;
(D11)                  2.5 > 1.7724539
(C12) %,PRED;
(D12)                        TRUE
```

EVFLAG Variable

défaut: [] - la liste des choses connues de la fonction EV. Un item sera lié à TRUE pendant l'exécution de EV s'il est mentionné dans l'appel à EV, e.g. EV(%,numer);. Les drapeaux initiaux evflags sont:

`FLOAT, PRED, SIMP, NUMER, DETOUT, EXPONENTIALIZE, DEMOIVRE, KEEPFLOAT, LISTARITH, TRIGEXPAND, SIMPSUM, ALGEBRAIC, RATALGDENOM, FACTORFLAG, %EMODE, LOGARC, LOGNUMER, RADEXPAND, RATSIMPEXPONS, RATMX, RATFAC, INFEVAL, %ENUMER, PROGRAMMODE, LOGNEGINT, LOGABS, LETRAT, HALFANGLES, EXPTISOLATE, ISOLATE_WRT_TIMES, SUMEXPAND, CAUCHYSUM, NUMER_PBRANCH, M1PBRANCH, DOTSCRULES, et LOGEXPAND.`

EVFUN Variable

- la liste des fonctions connues de la fonction EV qui seront appliquées si leur nom est mentionné. Les drapeaux initiaux evfuns sont FACTOR, TRIGEXPAND, TRIGREDUCE, BFLOAT, RATSIMP, RATEXPAND, RADCAN, LOGCONTRACT, RECTFORM, et POLARFORM.

INFEVAL symbole spécial

mène à un mode d'"évaluation infinie". EV évalue de façon répétée une expression jusqu'à ce qu'elle cesse de changer. Pour empêcher une variable, disons X, d'être évaluée dans ce mode, il suffit d'inclure X='X comme argument à EV. \'Evidemment des expressions comme EV(X,X=X+1,INFEVAL); vont créer une boucle infinie. CAVEAT EVALUATOR.

KILL (*arg1, arg2, ...*) Fonction

élimine ses arguments du système MACSYMA. Si argi est une variable (y compris un simple élément de tableau), une fonction, ou un tableau, l'item désigné avec

toutes ses propriétés est enlevé du noyau. Si argi=LABELS alors toute entrée, intermédiaire, et lignes en sortie jusqu'à présent (mais pas d'autres items nommés) sont éliminés. Si argi=CLABELS alors seules les lignes en sortie seront éliminées; si argi=ELABELS alors seules les lignes intermédiaires E-lines seront éliminées; si argi=DLABELS seules les lignes en sortie seront éliminées. Si argi est le nom de tout autre liste d'information (les éléments de la variable INFOLISTS de MACSYMA), alors tout item de cette classe (et ses propriétés) est KILLé et si argi=ALL alors chaque item de chaque liste d'information précédemment définie ainsi que toutes les LABELS est KILLé. Si argi=un nombre (disons n), alors les n dernières lignes (i.e. les lignes portant les derniers n numéros) sont supprimées. Si argi est de la forme [m,n] alors toutes les lignes portant des numéros entre m et n inclus sont tuées. Notez que KILL(VALUES) ou KILL(variable) ne libéreront pas l'espace occupé sauf si les étiquettes qui pointent sur les mêmes expressions sont aussi KILLées. Ainsi si une grosse expression a été affectée à X sur la ligne C7 on doit faire KILL(D7) ainsi que KILL(X) pour libéré l'espace occupé. KILL(ALLBUT(nom1,...,nomk) fera un KILL(ALL) sauf qu'il ne supprimera pas les noms spécifiés. (Note: nomi représente un nom tel que U, V, F, G, pas une liste d'info telle que FUNCTIONS). KILL ôte toutes les propriétés de l'argument donné, ainsi KILL(VALUES) supprimera toutes les propriétés associées à chaque item de la liste VALUES alors que l'ensemble de fonctions REMOVE (REMVALEUR,REMFONCTION,REMARRAY,REMRULE) enlève une propriété spécifique. Et cette dernière affiche aussi une liste de noms ou FALSE si l'argument spécifique n'existe pas alors que KILL a toujours la valeur "DONE" même si l'item nommé n'existe pas. Notez que supprimer des expressions n'aidera pas le problème qui survient sur MC indiqué par "NO CORE - FASLOAD" lorsque trop de fichiers FASL ont été chargés ou lorsque le niveau alloué est devenu trop haut. Dans aucun de ces cas, aucune quantité de suppression ne fera décroître la taille de ces espaces. Supprimer des expressions ne fait que vider certains de ces espaces sans les rendre plus petits.

LABELS (*char*) Fonction

prend un "char" C, D, ou E comme argument et génère une liste de toutes les étiquettes C-, D-, ou E-, respectivement. Si vous avez créé beaucoup d'étiquettes E- avec SOLVE, alors

```
FIRST(REST(LABELS(C)))
```

vous rappelle quelle était la dernière étiquette C. LABELS prendra comme arg tout nom symbolique, aussi si vous avez remis à zéro INCHAR, OUTCHAR, ou LINECHAR, elle retournera la liste des étiquettes dont le premier caractère correspond au premier caractère de l'argument que vous avez donné à LABELS. La variable, LABELS, par défaut: [], est une liste de lignes C, D, et E qui sont liées.

LASTTIME Variable

- le temps mis pour calculer la dernière expression, en millisecondes, présentée comme une liste de "time" et "gctime".

LINENUM Variable

- le numéro de la ligne de la dernière expression.

MYOPTIONS Variable

par défaut: [] - toutes les options remises à zéro par l'utilisateur (qu'elles soient ou non remises à leur valeur par défaut).

NOLABELS Variable

par défaut: [FALSE] - si TRUE alors aucune étiquette ne sera liée, sauf pour les lignes E générées par les fonctions solve. C'est surtout utile dans le mode "BATCH" où il élimine le besoin de faire un KILL(LABELS) afin de récupérer de l'espace.

OPTIONSET Variable

par défaut: [FALSE] - si TRUE, MACSYMA n'affichera pas de message lorsqu'une option MACSYMA est restaurée. Utile si l'utilisateur n'est pas sûr de l'orthographe d'une certaine option et veux s'assurer que la variable à laquelle il affecte une valeur est vraiment une variable d'option.

PLAYBACK (*arg*) Fonction

"rejoue" des lignes d'entrée et de sortie. Si arg=n (un nombre) les n dernières expressions (Ci, Di, et Ei compte chacune pour 1) sont "rejouées", alors que si arg est omis, toutes le sont. Si arg=INPUT seules les lignes en entrée sont rejouées. Si arg=[m,n] toutes les lignes portant les numéros m à n inclus sont rejouées. Si m=n alors [m] est un argument suffisant. Arg=SLOW place PLAYBACK en un mode lent semblable à celui de DEMO (comme opposé au BATCH "rapide"). Utile avec SAVE ou STRINGOUT lors de la création d'un fichier de stockage en second afin de désigner des expressions utiles. Si arg=TIME alors les temps de calcul sont affichés ainsi que les expressions. Si arg=GCTIME ou TOTALTIME, alors le détail complet des temps de calcul est affiché, comme avec SHOWTIME:ALL;. Arg=STRING échelonne (voir la fonction STRING) toutes les lignes en entrée lors du "play back" plutôt que de les afficher. Si ARG=GRIND le mode "grind" peut aussi être activé (pour le traitement des lignes en entrée) (voir GRIND). On peut inclure un nombre quelconque d'options comme dans PLAYBACK([5,10],20,TIME,SLOW).

PRINTPROPS (*a, i*) Fonction

affiche la propriété de l'indicateur i associé à l'atome a. a peut aussi être une liste d'atomes ou l'atome ALL auquel cas tous les atomesayant la propriété donnée seront utilisés. Par exemple, PRINTPROPS([F,G],ATVALEUR). PRINTPROPS est réservé aux propriétés qui ne pourraient autrement être affichées, i.e. pour ATVALEUR, ATOMGRAD, GRADEF, et MATCHDECLARE.

PROMPT Variable

par défaut: [_] est le symbomle d'invite de la fonction DEMO, du mode PLAYBACK(SLOW), et de (MACSYMA-BREAK).

QUIT () Fonction

supprime le MACSYMA actuel mais n'affecte pas les autres tâches de l'utilisateur; équivaut à la sortie de DCL en arrêtant le processus MACSYMA. On peut "quitter" le niveau haut de MACSYMA en tapant Contrôle-C Contrôle-G; Contrôle-C donne

l'invite d'interruption NIL, sur laquelle on peut taper Contrôle-G ou juste G. Taper X à l'invite d'interruption provoque l'arrêt d'un calcul commencé dans un MACSYMA-BREAK sans perturber le falcul principal suspendu.

REMFONCTION (*f1, f2, ...*) Fonction

enlève de MACSYMA les fonctions définies par l'utilisateur f1,f2,... . S'il n'y a qu'un argument ALL alors toutes les fonctions sont enlevées.

RESET () Fonction

remet à leurs valeurs par défaut toutes les options de MACSYMA. (Notez que ceci n'inclut pas les particularités des terminaux comme LINEL qui ne peuvent être modifiés que par affectation puisqu'ils ne sont pas considérés comme des particularités calculables de MACSYMA.)

RESTORE (*spécification-de-fichier*) Fonction

réinitialise toutes les quantités classées par les fonctions SAVE ou STORE, dans une session précédente de MACSYMA, du fichier donné par la spécification-de-fichier, sans les mettre dans le noyau.

SHOWTIME Variable

par défaut: [FALSE] - si TRUE le temps de calcul sera automatiquement affiché avec chaque expression sortie. Avec SHOWTIME:ALL, en plus du temps cpu MACSYMA affiche aussi maintenant (si non zéro) le temps passé en ramasse-miettes (gc) au cours du calcul. Ce temps est bien sûr inclus dans la durée affichée "time=". Il faut noter que comme "time=" ne comprend que le temps de calcul et non le temps d'affichage intermédiaire ou celui qu'il faut pour charger des fichiers hors du noyau, et comme il est difficile d'en tenir gc pour "responsable", le gctime affiché va inclure tout le temps gc supporté au cours du calcul et par conséquent peut dans de rares cas être même plus grand que "time=".

SSTATUS (*caractéristique,package*) Fonction

- signifie SET STATUS. Peut être utilisé en SSTATUS(FEATURE, HACK_PACKAGE) de sorte que STATUS(FEATURE, HACK_PACKAGE) retourne alors TRUE. Ce qui peut être utile au rédacteurs de package, pour garder une trace de ce qu'ils y ont mis.

TOBREAK () Fonction

provoque la ré-entrée dans une interruption MACSYMA, qui avait été quittée en tapant TOPLEVEL;. Si un argument quel qu'il soit est donné à TOBREAK, alors l'interruption sera abandonnée, ce qui est équivalent à TOBREAK() suivi immédiatement de EXIT;.

TOPLEVEL () Fonction

Pendant une interruption on peut taper TOPLEVEL;, qui provoque une entrée récursive au niveau haut dans MACSYMA. Les étiquettes seront alors liées comme habituellement. Tout sera identique à l'état du niveau haut précédent sauf que le calcul interrompu est sauvegardé.

TO_LISP () Fonction

entre dans le système LISP sous MACSYMA. Utile sur les systèmes où la touche Contrôle-flèche-haute n'est pas disponible pour cette fonction.

TTYINTFUN Variable

par défaut: [FALSE] - Gouverne la fonction qui sera lancée dès que le caractère Interruption-Utilisateur est frappé. Pour utiliser cette possibilité, on définit TTYINTFUN (le défaut FALSE signifie possibilité non utilisée) sur une fonction sans argument. Alors dès que, par exemple, ^U (contrôle-U) est frappé, cette fonction est exécutée. Supposons que vous ayez une instruction de boucle FOR avec incréments I, et que vous vouliez une méthode simple de vérifier la valeur de I pendant que l'instruction FOR s'exécute. Vous pouvez faire: TTYINTFUN:PRINTI$ PRINTI():=PRINT(I)$, alors dès que vous tapez ^U vous obtenez le test désiré.

TTYINTNUM Variable

par défaut: [21] (la valeur ascii de Contrôle-U (^U), U étant la 21ème lettre de l'alphabet). Ceci contrôle quel caractère devient celui de l'Interruption-Utilisateur. ^U a été choisi pour sa valeur mnémonique. La plupart des utilisateurs ne redéfiniront pas TTYINTNUM à moins qu'ils utilisent déjà ^U pour autre chose.

VALUES Variable

défaut:[] - tous les atomes liés, i.e. les variables utilisateur, pas les options ou les commutateurs de MACSYMA, (définis par : , :: , ou une liaison fonctionnelle).

4 Opérateurs

4.1 NARY

- Un opérateur NARY est utilisé pour représenter une fonction d'un nombre quelconque d'arguments, chacun d'eux étant séparé par une occurrence de l'opérateur, e.g. A+B ou A+B+C. La fonction NARY("x") est une fonction d'extension de syntaxe qui déclare x comme étant un opérateur NARY. Faites DESCRIBE(SYNTAX); pour avoir plus de détails. Les fonctions peuvent être déclarées (avec DECLARE) comme étant NARY. Si DECLARE(J,NARY); est lancée sur, par exemple, J(J(A,B),J(C,D)), le simplificateur renverra J(A, B, C, D).

4.2 NOFIX

- les opérateurs NOFIX sont utilisés pour représenter des fonctions sans argument. La simple présence d'un tel opérateur dans une commande provoque l'évaluation de la fonction correspondante. Par exemple, si on frappe "exit;" pour sortir d'une interruption MACSYMA, "exit" se comporte comme un opérateur NOFIX. La fonction NOFIX("x") est une fonction d'extension de syntaxe qui déclare x comme étant un opérateur NOFIX. Faites DESCRIBE(SYNTAX); pour avoir plus de détails.

4.3 OPERATOR

- Voir OPERATORS

4.4 POSTFIX

- les opérateurs POSTFIX, comme la variété PREFIX, représentent des fonctions à un seul argument, mais dans ce cas l'argument précède immédiatement une occurrence de l'opérateur dans la chaîne en entrée, e.g. 3!. La fonction POSTFIX("x") est une fonction d'extension de syntaxe qui déclare x comme étant un opérateur POSTFIX. Faites DESCRIBE(SYNTAX); pour avoir plus de détails.

4.5 PREFIX

- Un opérateur PREFIX signale une fonction à un argument, lequel suit immédiatement une occurrence de l'opérateur. PREFIX("x") est une fonction d'extension de syntaxe qui déclare x comme étant un opérateur PREFIX. Faites DESCRIBE(SYNTAX); pour avoir plus de détails.

4.6 Définitions pour les opérateurs

"!" opérateur

L'opérateur factoriel est le produit de tous les entiers de 1 à son argument. Ainsi 5! = 1*2*3*4*5 = 120. La valeur de l'option FACTLIM (par défaut: [-1]) donne le

plus grand factoriel qui est automatiquement développé. Si elle vaut -1 alors tous les entiers sont développés. Voir aussi les commandes FACTORIAL, MINFACTORIAL, et FACTCOMB.

"!!" opérateur

Représente le double factoriel qui est défini comme produit de tous les entiers consécutifs impairs (ou pairs) de 1 (ou 2) à l'argument impair (ou pair). Ainsi 8!! est 2*4*6*8 = 384.

"#" opérateur

L'opérateur logique "Non égal".

"." opérateur

L'opérateur point, pour la multiplication (non-commutative) de matrices. Lorsque "." est utilisé de cette façon, des espaces devront être laissés des deux côtés, e.g. A . B. Ceci le distingue entièrement d'un point décimal dans un nombre en virgule flottante. Faites APROPOS(DOT); pour avoir la liste des commutateurs qui affectent l'opérateur point. DESCRIBE(nom-commutateur); les expliquera.

":" opérateur

L'opérateur d'affectation. Par exemple A:3 donne à la variable A la valeur 3.

"::" opérateur

Opérateur d'affectation. :: assigne la valeur de l'expression à sa droite à la valeur de la quantité à sa gauche, qui doit s'évaluer en une variable atomique ou une variable indicée.

"::=" opérateur

Le "::=" est utilisé au lieu de ":=" pour indiquer que ce qui suit est une définition de macro, plutôt qu'une définition fonctionnelle ordinaire. Voir DESCRIBE(MACROS).

":=" opérateur

L'opérateur de définition de fonction. Par exemple F(X):=SIN(X) définit une fonction F.

"=" opérateur

indique une équation de MACSYMA. Pour le filtreur ("pattern matcher") de MACSYMA il dénote une relation totale entre deux expressions qui est vraie si et seulement si les expressions sont syntaxiquement identiques.

ADDITIVE symbole spécial

- Si DECLARE(F,ADDITIVE) a été exécutée, alors: (1) Si F n'a qu'un argument, dès que le simplificateur rencontre F appliquée à une somme, F sera distribuée sur cette somme, i.e. F(Y+X); sera simplifiée en F(Y)+F(X). (2) If F est une fonction à 2 arguments ou davantage, l'additivité est définie comme additivité sur le premier argument de F, comme dans le cas de 'SUM ou 'INTEGRATE, i.e. F(H(X)+G(X),X); sera simplifiée en F(H(X),X)+F(G(X),X). Cette simplification ne se produit pas lorsque F est appliquée à des expressions de la forme SUM(X[I],I,limite-inf,limite-sup).

ALLBUT mot-clé

travaille avec les commandes PART (i.e. PART, INPART, SUBSTPART, SUBSTINPART, DPART, et LPART). Par exemple,

```
if EXPR is E+D+C+B+A,
then PART(EXPR,[2,5]);
==> D+A
```

alors que

```
PART(EXPR,ALLBUT(2,5))==>E+C+B
```

Fonctionne aussi avec la commande KILL,

```
KILL(ALLBUT(nom1,...,nomk))
```

fera un KILL(ALL) sauf qu'il ne supprime pas les noms spécifiés. Note: nomi signifie un nom comme celui d'une fonction: U, F, FOO, ou G, mais pas une liste d'information telle que FUNCTIONS.

ANTISYMMETRIC déclaration

- La déclaration DECLARE(H,ANTISYMMETRIC); dit au simplificateur que H est antisymmétrique, p. ex. H(X,Z,Y) sera simplifiée en -H(X, Y, Z): c'est à dire qu'il donnera (-1)^n fois le résultat donné par SYMMETRIC ou COMMUTATIVE, où n est le nombre d'échanges entre deux arguments nécessaire pour le convertir sous cette forme.

CABS (*exp*) Fonction

retourne la valeur complexe absolue (le modulo complexe) de exp.

COMMUTATIVE déclaration

- La déclaration DECLARE(H,COMMUTATIVE); dit au simplificateur que H est une fonction commutative. E.g. H(X,Z,Y) sera simplifiée en H(X, Y, Z). C'est la même chose que SYMMETRIC.

ENTIER (*X*) Fonction

le plus grand entier <= X où X est numérique. FIX (comme dans FIXnum) en est un synonyme, ainsi FIX(X); est précisement le même.

EQUAL (*expr1,expr2*) Fonction

utilisé avec un "IS", retourne TRUE (ou FALSE) si et seulement si expr1 et expr2 sont égales (ou non égales) pour toutes les valeurs possibles de leurs variables (déterminées par RATSIMP). Ainsi IS(EQUAL((X+1)**2,X**2+2*X+1)) retourne TRUE alors que si X n'est pas lié IS((X+1)**2=X**2+2*X+1) retourne FALSE. Notez aussi que IS(RAT(0)=0) donne FALSE mais IS(EQUAL(RAT(0),0)) donne TRUE. Si une détermination ne peut être faite avec EQUAL alors une forme simplifiée mais équivalente est retournée, alors que = renvoie toujours TRUE ou FALSE. Toutes les variables se produisant dans exp sont supposées être à valeurs réelles. EV(exp,PRED) est équivalente à IS(exp).

```
(C1) IS(X**2 >= 2*X-1);
(D1)                                TRUE
(C2) ASSUME(A>1);
(D2)                                DONE
(C3) IS(LOG(LOG(A+1)+1)>0 ET A^2+1>2*A);
(D3)                                TRUE
```

EVAL Fonction

provoque une post-évaluation supplémentaire de exp.

EVENP (*exp*) Fonction

est TRUE si exp est un entier pair. FALSE est retournée dans tous les autres cas.

FIX (*x*) Fonction

un synonyme de ENTIER(X) - le plus grand entier <= X où X est numérique.

FULLMAP (*fn, exp1, ...*) Fonction

est semblable à MAP mais il retardera l'application aux sous-expressions jusqu'à ce que les principaux opérateurs ne soient plus les mêmes. L'utilisateur doit être conscient que FULLMAP est utilisé par le simplificateur de MACSYMA lors de certaines manipulations de matrices; ainsi, l'utilisateur peut voir un message d'erreur concernant FULLMAP alors même qu'il ne l'a pas explicitement appelée.

```
(C1) A+B*C$
(C2) FULLMAP(G,%);
(D2)               G(B) G(C) + G(A)
(C3)  MAP(G,D1);
(D3)                 G(B C) + G(A)
```

FULLMAPL (*fn, list1, ...*) Fonction

est semblable à FULLMAP mais ne réalise l'application que dans les listes et les matrices.

```
(C1) FULLMAPL("+",[3,[4,5]],[[A,1],[0,-1.5]]);
(D1)                       [[A + 3, 4], [4, 3.5]]
```

IS (*exp*) Fonction

tente de déterminer si exp (qui doit s'évaluer en un prédicat) est démontrable selon les faits contenus dans la base de données actuelle. IS renvoie TRUE si le prédicat est vrai pour toutes les valeurs de ses variables consistentes avec la base de données et retourne FALSE s'il est faux pour toutes ces valeurs. Autrement, son action dépend des paramètres du commutateur PREDERROR (par défaut: TRUE). IS se trompe si la valeur de PREDERROR is TRUE et retourne UNKNOWN si PREDERROR est FALSE.

ISQRT (*X*) Fonction
prend un argument entier et retourne la racine carrée entière de sa valeur absolue.

MAX (*X1, X2, ...*) Fonction
retourne le plus grand de ses arguments (ou une forme simplifiée si certains de ses arguments ne sont pas numériques).

MIN (*X1, X2, ...*) Fonction
retourne le plus petit de ses arguments (ou une forme simplifiée si certains de ses arguments ne sont pas numériques).

MOD (*poly*) Fonction
convertit le polynôme poly en une représentation modulaire respectant le modulo actuel qui est la valeur de la variable MODULUS. MOD(poly,m) spécifie un MODULUS m à utiliser pour convertir poly, en écrasant la valeur globale actuelle non désirée de MODULUS. Voir DESCRIBE(MODULUS); .

ODDP (*exp*) Fonction
est TRUE si exp est un entier impair. FALSE est retourné dans tous les autres cas.

PRED opérateur
(EVFLAG) provoque l'évaluation des prédicats (expressions qui s'évaluent en TRUE ou FALSE).

RANDOM (*X*) Fonction
retourne un entier aléatoire entre 0 et X-1. Si aucun argument n'est donné alors un entier aléatoire entre -2^(29) et 2^(29) -1 est retourné. Si X est FALSE alors la suite aléatoire est relancée depuis le début. Notez que la plage de valeurs du résultat retourné lorsqu'aucun argument n'est donné diffère dans NIL MACSYMA de celle de PDP-10 et Multics MACSYMA, qui est -2^(35) à 2^(35) -1. Cette plage est celle du type de donnée FIXNUM du LISP sous-jacent.

SIGN (*exp*) Fonction
tente de déterminer le signe de son expression spécifiée en s'appuyant sur les faits de la base de données actuelle. Elle retourne l'une des réponses suivantes: POS (positive), NEG (négative), ZERO, PZ (positive ou zéro), NZ (négative ou zéro), PN (positive ou négative), ou PNZ (positive, négative, ou zéro, i.e. ne sait pas).

SIGNUM (*X*) Fonction
si X<0 alors -1 sinon si X>0 alors 1 sinon 0. Si X n'est pas numérique alors une forme simplifiée mais équivalente est retournée. Par exemple, SIGNUM(-X) donne -SIGNUM(X).

SORT (*liste,prédicat-optionnel*) Fonction
trie la liste en utilisant un prédicat-optionnel convenable à deux arguments (comme "<" ou ORDERLESSP). Si ce prédicat-optionnel n'est pas donné le prédicat d'ordre interne de MACSYMA est utilisé.

SQRT (*X*) Fonction

la racine carrée de X. Elle est représentée en interne par X^(1/2). Voyez aussi ROOTSCONTRACT. RADEXPAND[TRUE] - si TRUE provoque la sortie du radical des nième racines des facteurs d'un produit qui sont des puissances de n, e.g. SQRT(16*X^2) deviendra 4*X seulement si RADEXPAND est TRUE.

SQRTDISPFLAG Variable

par défaut: [TRUE] - si FALSE provoque l'affichage de SQRT avec exposant 1/2.

SUBLIS (*liste,expr*) Fonction

permet de multiples substitutions en parallèle dans une expression. Exemple de syntaxe:

```
SUBLIS([A=B,B=A],SIN(A)+COS(B));
 => SIN(B) + COS(A)
```

La variable SUBLIS_APPLY_LAMBDA[TRUE] contrôle la simplification après SUBLIS. Pour une documentation complète, voyez le fichier SHARE2;SUBLIS INFO.

SUBLIST (*L,F*) Fonction

retourne la liste des éléments de la liste L pour lesquels la fonction F retourne TRUE. E.g., SUBLIST([1,2,3,4],EVENP); retourne [2,4].

SUBLIS_APPLY_LAMBDA Variable

par défaut:[TRUE] - contrôle si les substitués LAMBDA sont appliqués par simplification après utilisation de SUBLIS ou si vous devez faire un EV pour que les choses soient appliquées. TRUE signifie faire l'application.

SUBST (*a, b, c*) Fonction

substitue a pour b dans c. b doit être un atome, ou une sous-expression complète de c. Par exemple, X+Y+Z est une sous-expression complète de 2*(X+Y+Z)/W alors que X+Y ne l'est pas. Lorsque b n'a pas ces caractéristiques, on peut quelquefois utiliser SUBSTPART ou RATSUBST (voir ci-dessous). Alternativement, si b est de la forme e/f alors on pourra utiliser SUBST(a*f,e,c) alors que si b est de la forme e**(1/f) on pourra utiliser SUBST(a**f,e,c). La commande SUBST discerne aussi le X^Y dans X^-Y de sorte que SUBST(A,SQRT(X),1/SQRT(X)) donne 1/A. a et b peuvent aussi être des opérateurs d'une expression entourée de " ou elles peuvent être des noms de fonction. Si on veut substituer la variable indépendante des formes dérivées alors la fonction AT (voir ci-dessous) devra être utilisée. Note: SUBST est un alias de SUBSTITUTE. SUBST(eq1,exp) ou SUBST([eq1,...,eqk],exp) sont d'autres formes permises. Les eqi sont des équations indiquant les substitutions à effectuer. Dans chaque équation, la partie droite sera substituée à la gauche dans l'expression exp. EXPTSUBST[FALSE] si TRUE permet des substitutions comme Y pour %E**X dans %E**(A*X). OPSUBST[TRUE] si FALSE, SUBST ne tentera pas de substituer dans l'opérateur d'une expression. E.g. (OPSUBST:FALSE, SUBST(X^2,R,R+R[0])); fonctionnera.

```
(C1) SUBST(A,X+Y,X+(X+Y)**2+Y);
                                    2
(D1)                      Y + X + A
(C2) SUBST(-%I,%I,A+B*%I);
(D2)                               A - %I B
```

Notez que C2 est une façon d'obtenir le conjugué complexe d'une expression analytique. Pour d'autres exemples, faire EXAMPLE(SUBST);

SUBSTINPART (*x, exp, n1, ...*) Fonction

est commes SUBSTPART mais travaille sur la représentation interne de exp.

```
(C1) X.'DIFF(F(X),X,2);
                                      2
                                     d
(D1)                          X . (--- F(X))
                                      2
                                     dX
(C2) SUBSTINPART(D**2,%,2);
                                         2
(D2)                                  X . D
(C3) SUBSTINPART(F1,F[1](X+1),0);
(D3)                                F1(X + 1)
```

Information supplémentaire: si le dernier argument d'une fonction PART est une liste d'indices alors plusieurs sous-expressions sont choisies, chacune correspondant à un index de la liste. Ainsi

```
(C1) PART(X+Y+Z,[1,3]);
(D1) Z+X
```

PIECE contient la valeur de la dernière expression sélectionnée par l'utilisation des fonctions part. Il est défini pendant l'exécution de la fonction et peut ainsi être référencé dans la fonction elle-même, comme montré ci-dessous. Si PARTSWITCH[FALSE] est mise à TRUE alors END est retournée lorsqu'une partie sélectionnée d'une expression n'existe pas, autrement un message d'erreur est donné.

```
(C1)  27*Y**3+54*X*Y**2+36*X**2*Y+Y+8*X**3+X+1;
            3         2       2            3
(D1)    27 Y  + 54 X Y  + 36 X  Y + Y + 8 X  + X + 1
(C2)  PART(D1,2,[1,3]);
                  2
(D2)           54 Y
(C3)  SQRT(PIECE/54);
                Y
(C4)  SUBSTPART(FACTOR(PIECE),D1,[1,2,3,5]);
                           3
(D4)            (3 Y + 2 X) + Y + X + 1
(C5) 1/X+Y/X-1/Z;
                                  1   Y   1
(D5)                            - - + - + -
                                  Z   X   X
```

```
(C6) SUBSTPART(XTHRU(PIECE),%,[2,3]);
                                  Y + 1   1
(D6)                              ----- - -
                                    X     Z
```

Et aussi, fixer l'option INFLAG à TRUE et appeler PART/SUBSTPART est équivalent à l'appel INPART/SUBSTINPART.

SUBSTPART (*x, exp, n1, ..., nk*) Fonction

substitue x dans la sous-expression choisie par le reste des arguments comme dans PART. Il retourne la nouvelle valeur de exp. x peut être un opérateur devant être substitué à un opérateur de exp. Dans certains cas il doit être entouré de " (e.g. SUBSTPART("+",A*B,0); -> B + A).

```
(C1) 1/(X**2+2);
                                     1
(D1)                               ------
                                    2
                                   X  + 2
(C2) SUBSTPART(3/2,%,2,1,2);
                                     1
(D2)                               --------
                                    3/2
                                   X    + 2
(C3) A*X+F(B,Y);
(D3)                            A X + F(B, Y)
(C4) SUBSTPART("+",%,1,0);
(D4)                          X + F(B, Y) + A
```

Et aussi, fixer l'option INFLAG à TRUE et appeler PART/SUBSTPART est équivalent à l'appel INPART/SUBSTINPART.

SUBVARP (*exp*) Fonction

est TRUE si exp est une variable indicée, par exemple A[I].

SYMBOLP (*exp*) Fonction

retourne TRUE si "exp" est un "symbole" ou un "nom", autrement FALSE. En fait, SYMBOLP(X):=ATOM(X) AND NOT NOMBREP(X)$.

UNORDER () Fonction

stoppe le crénelage créé par la dernière utilisation des commandes de classement ORDERGREAT et ORDERLESS. ORDERGREAT et ORDERLESS ne doivent pas être utilisées plus d'une fois chacune sans appel à UNORDER. Faites DESCRIBE(ORDERGREAT); et DESCRIBE(ORDERLESS);, et aussi EXAMPLE(UNORDER); pour les spécifications.

VECTORPOTENTIAL (*givencurl*) Fonction

Retourne le potentiel vecteur d'un vecteur curl donné, dans le système de coordonnées courant. POTENTIALZEROLOC a un rôle semblable a celui de POTENTIAL,

mais l'ordre des parties gauches des équations doit être une permutation cyclique des variables de coordonnées.

XTHRU (*exp*) Fonction

combine tous les termes de exp (qui doit être une somme) sur un dénominateur commun sans développer les sommes de produits et exponentiées comme le fait RATSIMP. XTHRU annule les facteurs communs du numérateur et du dénominateur des expressions rationnelles mais seulement si les facteurs sont explicites. Quelquefois il vaut mieux utiliser XTHRU avant d'appliquer RATSIMP à une expression afin de provoquer l'annulation des facteurs explicites du pgcd du numérateur et du dénominateur, en simplifiant ainsi l'expression à passer à RATSIMP.

```
(C1) ((X+2)**20-2*Y)/(X+Y)**20+(X+Y)**-19-X/(X+Y)**20;
                                                    20
                      1            X        (X + 2)   - 2 Y
(D1)              --------- - --------- + ---------------
                         19           20              20
                  (Y + X)      (Y + X)         (Y + X)
(C2) XTHRU(%);
                                      20
                                (X + 2)   - Y
(D2)                            -------------
                                         20
                                  (Y + X)
```

ZEROEQUIV (*exp,var*) Fonction

teste si l'expression exp dans la variable var est équivalente à zéro. Retourne TRUE, FALSE, ou DONTKNOW (ne sait pas). Par exemple ZEROEQUIV(SIN(2*X) - 2*SIN(X)*COS(X),X) retourne TRUE et ZEROEQUIV(%E^X+X,X) retourne FALSE. D'autre part ZEROEQUIV(LOG(A*B) - LOG(A) - LOG(B),A) retournera DONTKNOW à cause de la présence d'un paramètre supplémentaire. Les restrictions sont: (1) Ne pas utiliser de fonctions que MACSYMA ne sait pas comment différencier et évaluer. (2) Si l'expression a des pôles sur la ligne réelle, il peut y avoir des erreurs dans le résultat (mais il est peu probable que cela se produise). (3) Si l'expression contient des fonctions qui ne sont pas solutions d'équations différentielles du premier ordre (e.g. des fonctions de Bessel) les résultats peuvent être incorrects. (4) L'algorithme utilise l'évaluation en des points choisis au hasard pour des sous-expressions soigneusement sélectionnées. C'est toujours une tâche quelque peu hasardeuse, bien que l'algorithme essaie de minimiser l'erreur potentielle.

5 Expressions

5.1 Introduction aux expressions

Un certain nombre de mots réservés ne peuvent être utilisés comme noms de variable. Leur utilisation pourrait provoquer une erreur de syntaxe mystérieuse.

INTEGRATE	NEXT	FROM	DIFF
IN	AT	LIMIT	SUM
FOR	AND	ELSEIF	THEN
ELSE	DO	OR	IF
UNLESS	PRODUCT	WHILE	THRU
STEP			

La plupart des objets dans MAXIMA sont des expressions. Une suite d'expressions peut être réduite en une seule expression en les séparant par des virgules et en les entourant de parenthèses. C'est semblable à la **C** *expression virgule*.

```
(C29) x:3$
(C30) joe:(x:x+1,x:x*x);
(D30) 16
(C31) joe:(if (x >17) then 2 else 4);
(D31) 4
(C32) joe:(if (x >17) then x:2 else joe:4,joe+x);
(D32) 20
```

Même les boucles dans MAXIMA sont des expressions, bien que la valeur qu'elles retournent est un `DONE` non utilisable:

```
(C33) joe:(x:1,for i from 1 thru 10 do (x:x*i));
(D33) DONE
```

où vous vouliez sans doute inclure un troisième terme dans l'*expression virgule* qui en fait renvoie la valeur:

```
(C34) joe:(x:1,for i from 1 thru 10 do (x:x*i),x);
(D34) 3628800
```

5.2 Affectation

- Il y a deux opérateurs d'affectation dans MACSYMA, : et :: . E.g. A:3 donne à la variable A la valeur 3. :: assigne la valeur de l'expression de son membre droit à la valeur de la quantité de son membre gauche, qui doit s'évaluer en une variable atomique ou une variable indicée.

5.3 Complexes

- Une expression complexe est spécifiée dans MACSYMA en ajoutant la partie réelle de l'expression à %I fois la partie imaginaire. Ainsi les racines de l'équation X^2-4*X+13=0 sont 2+3*%I et 2-3*%I. Notez que la simplification de produits d'expressions complexes peut être effectuée en développant le produit. La simplification des quotients, racines, et des autres fonctions d'expressions complexes peut en général se faire avec les fonctions REALPART, IMAGPART, RECTFORM, POLARFORM, ABS, et CARG.

5.4 Inégalités

- MACSYMA a les opérateurs d'inégalité usuels: inférieur à: < plus grand que: > plus grand que ou égal à: >= inférieur ou égal à: <=

5.5 Syntaxe

- Il est possible d'ajouter de nouveaux opérateurs à MACSYMA (infixé, préfixé, postfixé, unaire, ou fixé avec précédences données), de supprimer des opérateurs existants, ou de redéfinir la précédence d'opérateurs existants. Bien que la syntaxe de MACSYMA soit adéquate pour la plupart des applications ordinaires, il est possible de définir de nouveaux opérateurs ou d'éliminer ceux qui sont prédéfinis. Le méchanisme d'extension est plutôt direct et devrait être évident d'après les exemples ci-dessous.

```
(C1) PREFIX("DDX")$
(C2) DDX Y$
     /* means                    "DDX"(Y) */
(C3) INFIX("<-")$
(C4) A<-DDX Y$
    /* means                "<-"(A,"DDX"(Y)) */
```

À chacun des types d'opérateur, sauf SPECIAL, correspond une fonction de création qui donnera au lexème spécifié les propriétés d'analyse correspondantes. Ainsi "PREFIX("DDX")" fera de "DDX" un opérateur préfixé exactement comme "-" ou "NOT". Bien sûr, certaines fonctions d'extension requièrent des informations additionnelles comme le mot-clé correspondant à un opérateur fixé. De plus, les puissances liées et des parties du discours doivent être spécifiées pour tous les mots-clé définis, ce qui se fait en passant des arguments supplémentaires aux fonctions d'extension. Si un utilisateur ne spécifie aucun de ces paramètres additionnels, MACSYMA leur donnera des valeurs par défaut. Les valeurs par défaut des six fonctions d'extension avec puissances liées et des parties du discours (entourées de crochets) sont résumées ci-dessous.

```
PREFIX(opérateur, rbp[180], rpos[ANY], pos[ANY])
POSTFIX(opérateur, lbp[180], lpos[ANY], pos[ANY])
INFIX(opérateur, lbp[180], rbp[180], lpos[ANY], rpos[ANY],pos[ANY])
NARY(opérateur, bp[180], argpos[ANY], pos[ANY])
NOFIX(opérateur, pos[ANY])
MATCHFIX(opérateur, match, argpos[ANY], pos[ANY])
```

Ces valeurs par défaut ont été fournies afin qu'un utilisateur ne voulant pas se préoccuper lui-même avec ces parties de discours ou ces puissances liées puisse simplement omettre ces arguments aux fonctions d'extension. Ainsi ce qui suit sont des équivalents:

```
PREFIX("DDX",180,ANY,ANY)$
PREFIX("DDX",180)$
PREFIX("DDX")$
```

Il est aussi possible de supprimer les propriétés syntaxiques d'un opérateur avec les fonctions REMOVE ou KILL. Plus précisément, "REMOVE("DDX",OP)" ou "KILL("DDX")" retourneront "DDX" à l'état de l'opérande; mais dans le second cas toutes les autres propriétés de "DDX" seront aussi supprimées.

```
(C20) PREFIX("DDX",180,ANY,ANY)$

(C21) DDXYZ;

(D21)                                DDX YZ

(C26) "ddx"(u):=u+4;

(D26)                            DDX u := u + 4
(C27) ddx 8;

(D27)                                  12
```

5.6 Définitions pour les expressions

AT (*exp, liste*) Fonction

va évaluer exp (qui peut être une expression quelconque) avec les valeurs des variables spécifiées dans la liste des équations ou l'équation unique semblable à celle donnée à la fonction ATVALUE. Si une sous-expression dépend de l'une des variables de la liste mais qu'elle n'a pas encore de "atvalue" spécifiée et qu'elle ne peut être évaluée alors une forme nominale de AT sera retournée qui sera affichée en deux dimensions. Faites EXAMPLE(AT); pour avoir un exemple.

BOX (*expr*) Fonction

retourne expr enfermée dans une boîte. La boîte fait en réalité partie de l'expression.

```
BOX(expr,label)
```

enferme expr dans une boîte étiquetée. label est un nom qui sera tronqué à l'affichage s'il est trop long. BOXCHAR["] - est le caractère utilisé pour dessiner la boîte dans cette fonction et dans DPART et LPART.

BOXCHAR Variable

défaut: ["] est le caractère utilisé pour dessiner la boîte des fonctions BOX, DPART et LPART.

CONSTANT opérateur spécial

- fait de ai une constante telle que %PI.

CONSTANTP (*exp*) Fonction

est TRUE si exp est une constante (i.e. composée de nombres et %PI, %E, %I ou toute variable liée à une constante ou déclarée constante), sinon FALSE. Toute fonction dont les arguments sont des constantes est aussi considérée comme étant une constante.

CONTRACT (*exp*) Fonction

exécute toutes les contractions possibles de exp, qui peut être une combinaison quelconque bien formée de sommes et produits. Cette fonction utilise l'information donnée à la fonction DEFCON. Comme tous les tenseurs sont considérés comme étant symétriques pour tous les indices, ceus-ci sont triés par ordre alphabétique. De plus tous les indices fictifs sont enlevés en utilisant les symboles !1,!2,... pour permettre à l'expression d'être simplifiée le plus possible en mettant sous forme canonique les termes équivalents. Pour obtenir les meilleurs résultats exp devra être entièrement développée. RATEXPAND est la façon la plus rapide de développer produits et puissances de sommes s'il n'y a pas de variables dans les dénominateurs des termes. Le commutateur GCD devra être FALSE si des annulations de pgcd ne sont pas nécessaires.

DECLARE (*a1, f1, a2, f2, ...*) Fonction

donne aux atomes ai le drapeau fi. Les ai fi peuvent aussi être des listes d'atomes et de drapeaux, dans ce cas chacun des atomes a toutes les propriétés. Les drapeaux possibles avec leur signification sont:

CONSTANT - fait de ai une constante telle que %PI.

MAINVAR - fait de ai une MAINVAR. L'échelle de l'ordre des atomes: nombres < constantes (e.g. %E,%PI) < scalaires < autres variables < mainvars.

SCALAR - fait de ai un scalaire.

NONSCALAR - fait se comporter ai comme une liste ou une matrice par rapport à l'opérateur point.

NOUN - fait de la fonction ai un nom de sorte qu'elle ne sera pas évaluée automatiquement.

EVFUN - ai est connue de la fonction EV de sorte qu'elle sera appliquée si son nom est mentionné. Les evfuns initiaux sont

```
FACTOR, TRIGEXPAND,
TRIGREDUCE, BFLOAT, RATSIMP, RATEXPAND, et RADCAN
```

EVFLAG - ai est connue de la fonction EV de sorte qu'elle sera liée à TRUE pendant l'exécution de EV si elle est mentionnée. Les evflags initiaux sont

```
FLOAT, PRED, SIMP, NUMER, DETOUT, EXPONENTIALIZE, DEMOIVRE,
KEEPFLOAT, LISTARITH, TRIGEXPAND, SIMPSUM, ALGEBRAIC,
RATALGDENOM, FACTORFLAG, %EMODE, LOGARC, LOGNUMER,
RADEXPAND, RATSIMPEXPONS, RATMX, RATFAC, INFEVAL, %ENUMER,
PROGRAMMODE, LOGNEGINT, LOGABS, LETRAT, HALFANGLES,
EXPTISOLATE, ISOLATE_WRT_TIMES, SUMEXPAND, CAUCHYSUM,
NUMER_PBRANCH, M1PBRANCH, DOTSCRULES, et LOGEXPAND
```

BINDTEST - permet à ai de signaler une erreur si elle est utilisée dans un calcul non lié. DECLARE([var1, var2, ...], BINDTEST) permet à MACSYMA de donner un message d'erreur dès qu'un "vari" non lié se trouve dans un calcul. MACSYMA reconnaît actuellement et utilise les caractéristiques suivantes des objets:

```
EVEN, ODD, INTEGER, RATIONAL, IRRATIONAL, REAL, IMAGINARY,
et COMPLEX
```

Les particularités utiles des fonctions comprennent:

```
INCREASING, DECREASING, ODDFUN (fonction impaire), EVENFUN (fonction paire),
COMMUTATIVE (ou SYMMETRIC), ANTISYMMETRIC, LASSOCIATIVE et RASSOCIATIVE
```

DECLARE(F,INCREASING) est à tous égards équivalente à

```
ASSUME(KIND(F,INCREASING))
```

Les ai et fi peuvent aussi être des listes d'objets ou de caractéristiques. La commande

```
FEATUREP(objet,caractéristique)
```

peut être utilisée pour déterminer si un objet a été déclaré comme ayant "caractéristiques". Voyez DESCRIBE(FEATURES); .

DISOLATE (*exp, var1, var2, ..., varN*) Fonction

est semblable à ISOLATE(exp, var) (faites DESCRIBE(ISOLATE);) sauf qu'elle permet à l'utilisateur d'isoler plus d'une variable simultanément. Ce peut être utile, par exemple, si on tentait de changer des variables d'une intégration multiple, et que cette modification de variable implique deux ou davantage de variables d'intégration. Pour accéder à cette fonction, faites LOAD(DISOL)$. Une démonstration est disponible avec DEMO("disol"); .

DISPFORM (*exp*) Fonction

retourne la représentation externe de exp (wrt its main operator "wrt" son opérateur principal?). Devrait être utile conjointement avec PART qui traite aussi de la représentation externe. Supposez que EXP soit -A . Alors la représentation interne de EXP est "*"(-1,A), alors que sa représentation externe est "-"(A). DISPFORM(exp,ALL) convertit l'expression tout entière (pas seulement le haut niveau) au format externe. Par exemple, si EXP:SIN(SQRT(X)), alors FREEOF(SQRT,EXP) et FREEOF(SQRT,DISPFORM(EXP)) donnent TRUE, alors que FREEOF(SQRT,DISPFORM(EXP,ALL)) donne FALSE.

DISTRIB (*exp*) Fonction

distribue les sommes sur les produits. Elle diffère de EXPAND en ce qu'elle travaille seulement au niveau haut d'une expression, i.e. elle ne fait pas de récursion et elle est plus rapide que EXPAND. Elle diffère de MULTTHRU en ce qu'elle développe toutes les sommes à ce niveau. Par exemple,

```
DISTRIB((A+B)*(C+D)) -> A C + A D + B C + B D
MULTTHRU ((A+B)*(C+D)) -> (A + B) C + (A + B) D
DISTRIB (1/((A+B)*(C+D))) ->  1/ ((A+B) *(C+D))
EXPAND(1/((A+B)*(C+D)),1,0) -> 1/(A C + A D + B C + B D)
```

DPART (*exp, n1, ..., nk*) Fonction

sélectionne la même sous-expression que PART, mais au lieu de retourner seulement la valeur de cette sous-expression, elle retourne toute l'expression avec la sous-expression sélectionnée affichée dans une boîte. La boîte fait en réalité partie de l'expression.

```
(C1) DPART(X+Y/Z**2,1,2,1);
                          Y
(D1)                     ---- + X
                           2
```

```
*****
* Z *
*****
```

EXP (*X*) Fonction

la fonction exponentielle. Elle est représentée en interne comme %E^X.

DEMOIVRE[FALSE] - si TRUE %E^(A+B*%I) devient %E^A*(COS(B)+%I*SIN(B)) si B est indépendante de %I. A et B ne sont pas développés.

%EMODE[TRUE] - si TRUE %E^(%PI*%I*X) sera simplifiée comme suit: elle deviendra COS(%PI*X)+%I*SIN(%PI*X) si X est un entier ou un multiple de 1/2, 1/3, 1/4, ou 1/6 et sera ainsi simplifiée davantage. Pour d'autres valeurs numériques de X elle deviendra %E^(%PI*%I*Y) où Y est X-2*k pour un certain entier k tel que ABS(Y)<1. Si %EMODE est FALSE aucune simplification de %E^(%PI*%I*X) n'aura lieu.

%ENUMER[FALSE] - si TRUE %E sera convertie en 2.718... dès que NUMER est TRUE. Par défaut cette conversion n'aura lieu que si l'exposant de %E^X s'évalue en un nombre.

EXPTISOLATE Variable

par défaut: [FALSE] - si TRUE, ISOLATE(expr,var); va examiner les exposants des atomes (comme %E) qui contiennent var.

EXPTSUBST Variable

par défaut: [FALSE] - si TRUE permet à des substitutions telle que Y pour %E**X dans %E**(A*X) d'avoir lieu.

FREEOF (*x1, x2, ..., exp*) Fonction

amène TRUE si les xi ne se trouvent pas dans exp et FALSE autrement. Les xi sont des atomes ou des noms indicés, des fonctions (e.g. SIN(X)), ou des opérateurs entre guillemets ". Si 'var' est une "variable fictive" de 'exp', alors FREEOF(var,exp); retournera TRUE. Les "variables fictives" sont des objets mathématiques comme l'indice d'une somme ou d'un produit, une variable limite, et une variable d'intégration définie. Exemple: FREEOF(I,'SUM(F(I),I,0,N)); retourne TRUE. Faites EXAMPLE(FREEOF); pour d'autres exemples.

GENFACT (*X, Y, Z*) Fonction

est le factoriel généralisé de X qui est: X*(X-Z)*(X-2*Z)*...*(X-(Y-1)*Z). Ainsi, pour l'entier X, GENFACT(X,X,1)=X! et GENFACT(X,X/2,2)=X!!

IMAGPART (*exp*) Fonction

retourne la partie imaginaire de l'expression exp.

INDICES (*exp*) Fonction

retourne une liste de deux éléments. Le premier est une liste d'indices libres dans exp (ceux qui n'apparaissent qu'une seule fois); le second est la liste d'indices factices dans exp (ceux qui apparaissent exactement deux fois).

INFIX (*op*) Fonction

- les opérateurs INFIX sont utilisés pour représenter des fonctions à deux arguments, l'un donné avant l'opérateur et l'autre après, e.g. A^2 . La fonction INFIX("x") est une fonction d'extension de syntaxe déclarant x comme étant un opérateur INFIX. Faites DESCRIBE(SYNTAX); pour plus de détails.

INFLAG Variable

par défaut: [FALSE] - si mit à TRUE, les fonctions for, part, extraction regarderont la forme interne de exp. Notez que le simplificateur ré-ordonne les expressions. Ainsi FIRST(X+Y) sera X si INFLAG est TRUE et Y si INFLAG est FALSE (FIRST(Y+X) donne le même résultat). Et aussi, mettre INFLAG à TRUE et appeler PART/SUBSTPART revient à appeler INPART/SUBSTINPART. Les fonctions affectées par le paramétrage de INFLAG sont: PART, SUBSTPART, FIRST, REST, LAST, LENGTH, la construction FOR ... IN, MAP, FULLMAP, MAPLIST, REVEAL et PICKAPART.

INPART (*exp, n1, ..., nk*) Fonction

est semblable à PART mais agit sur la représentation interne de l'expression plutôt que sur la forme affichée et peut donc être plus rapide puisqu'aucun formatage n'est fait. Il faut prendre soin de respecter l'ordre des sous-expressions dans les sommes et les produits (car l'ordre des variables dans la forme interne est souvent différent de celui de la forme affichée) et aussi avec le moins unaire, la soustraction, et la division (car ces opérateurs sont enlevés de l'expression). PART(X+Y,0) ou INPART(X+Y,0) amènent +, bien qu'afin de se référer à l'opérateur il doit être entre guillemets ". Par exemple ...IF INPART(D9,0)="+" THEN ...

```
(C1)  X+Y+W*Z;
(D1)                    W Z + Y + X
(C2)  INPART(D1,3,2);
(D2)                    Z
(C3)  PART(D1,1,2);
(D3)                    Z
(C4) 'LIMIT(F(X)**G(X+1),X,0,MINUS);
                                            G(X + 1)
(D4)                         LIMIT   F(X)
                             X ->0-
(C5) INPART(%,1,2);
(D5)                               G(X + 1)
```

ISOLATE (*exp, var*) Fonction

retourne exp avec des sous-expressions qui sont des sommes et qui ne contiennent pas var remplacée par des étiquettes d'expressions intermédiaires (celles-ci étant des symboles atomiques tels que E1, E2, ...). C'est souvent utile pour éviter le développement non nécessaire de sous-expressions qui ne contiennent pas la variable intéressante. Comme les étiquettes intermédiaires sont liées aux sous-expressions elles peuvent toutes être substituées en retour par évaluation de l'expression dans laquelle elles se

trouvent. EXPTISOLATE[FALSE] si TRUE force ISOLATE à examiner les exposants des atomes (comme %E) qui contiennet var. ISOLATE_WRT_TIMES[FALSE] si TRUE, alors ISOLATE isolera aussi les produits "wrt". P. ex. comparez les deux paramétrages du commutateur sur ISOLATE(EXPAND((A+B+C)^2),C); . Faites EXAMPLE(ISOLATE); pour les exemples.

ISOLATE_WRT_TIMES Variable

par défaut: [FALSE] - si TRUE, alors ISOLATE va aussi isoler les produits "wrt". P. ex. comparez les deux paramétrages du commutateur sur ISOLATE(EXPAND((A+B+C)^2),C); .

LISTCONSTVARS Variable

par défaut: [FALSE] - si TRUE force LISTOFVARS à inclure %E, %PI, %I, et toutes les variables déclarées comme constantes dans la liste qu'elle retourne si elles apparaîssent dans l'expression sur laquelle LISTOFVARS est appelée. Le défaut est de les omettre.

LISTDUMMYVARS Variable

par défaut: [TRUE] - si FALSE, les "variables factices" dans l'expression ne seront pas incluses dans la liste retournée par LISTOFVARS (la signification de " variables factices" est donnée dans DESCRIBE(FREEOF): les "variables factices" sont des objets mathématiques tels que les indices d'une somme ou d'un produit, la variable limite, et la variable d'intégration définie). Exemple: LISTOFVARS('SUM(F(I),I,0,N)); donne [I,N] si LISTDUMMYVARS est TRUE, et [N] si LISTDUMMYVARS est FALSE.

LISTOFVARS (*exp*) Fonction

renvoie une liste des variables dans exp. LISTCONSTVARS[FALSE] si TRUE force LISTOFVARS à inclure %E, %PI, %I, et toutes les variables déclarées comme constantes dans la liste renvoyée si elles apparaissent dans exp. Elles sont omises par défaut.

```
(C1) LISTOFVARS(F(X[1]+Y)/G**(2+A));
(D1)                              [X[1], Y, A, G]
```

LOPOW (*exp, v*) Fonction

le plus petit exposant de v qui apparaît explicitement dans exp. Ainsi

```
LOPOW((X+Y)**2+(X+Y)**A,X+Y) ==> MIN(A,2)
```

.

LPART (*label, expr, n1, ..., nk*) Fonction

est semblable à DPART mais utilise une boîte étiquetée. Cette boîte est semblable à celle produite par DPART mais elle a un nom sur la ligne du haut.

MULTTHRU (*exp*) Fonction

multiplie un facteur (qui doit être une somme) de exp par les autres facteurs de exp. C'est à dire que exp est f1*f2*...*fn où au moins un facteur, disons fi, est une somme de termes. Chaque terme de cette somme est multiplié par les autres facteurs dans le produit (c-à-d. tous les facteurs sauf fi). MULTTHRU ne développe pas les sommes exponentielles. Cette fonction est la méthode la plus rapide pour distribuer des produits (commutatifs ou non commutatifs) sur des sommes. Comme les quotients sont représentés comme des produits MULTTHRU peut aussi être utilisé pour diviser des sommes par des produits. MULTTHRU(exp1, exp2) multiplie chacun des termes de exp2 (qui doit être une somme ou une équation) par exp1. Si exp1 n'est pas elle-même une somme alors cette forme est équivalente à MULTTHRU(exp1*exp2).

```
(C1) X/(X-Y)**2-1/(X-Y)-F(X)/(X-Y)**3;
                1         X          F(X)
(D1)        - ----- + -------- - --------
              X - Y           2          3
                        (X - Y)    (X - Y)
(C2) MULTTHRU((X-Y)**3,%);
                     2
(D2)        - (X - Y)  + X (X - Y) - F(X)
(C3) RATEXPAND(D2);
                            2
(D3)                      - Y  + X Y - F(X)
(C4) ((A+B)**10*S**2+2*A*B*S+(A*B)**2)/(A*B*S**2);
                          10  2              2  2
                 (B  + A )   S  + 2 A B S + A  B
(D4)             --------------------------------
                                    2
                                 A B S
(C5) MULTTHRU(%);
                                              10
                          2   A B   (B  + A)
(D5)                      - + --- + -------
                          S    2      A B
                             S
(notez que (B+A)**10 n'est pas développée)
(C6) MULTTHRU(A.(B+C.(D+E)+F));
(D6)                A . F + A . (C . (E + D)) + A . B
(comparez à l'exemple semblable sous EXPAND)
```

NOUNIFY (*f*) Fonction

retourne la forme nominale de la fonction nommée f. Nécessaire si on veut se référer au nom d'une fonction verbale comme si il était un substantif. Notez que certaines fonctions verbales retourneront leur formes nominales si elles ne peuvent être évaluées sur certains arguments. C'est aussi la forme retournée si l'appel de la fonction est précédé d'une apostrophe.

NTERMS (*exp*) Fonction

donne le nombre de termes que exp aurait si elle était complètement développée et qu'aucune suppression ou combinaison de termes ne se soient produites. Notez que les expressions comme SIN(E), SQRT(E), EXP(E), etc. comptent juste pour un terme quel que soit le nombre de termes de E (si c'est une somme).

OPTIMIZE (*exp*) Fonction

retourne une expression qui produit les mêmes valeurs et effets de bord que exp mais le fait plus efficacement en évitant le recalcul de sous-expressions communes. OPTIMIZE a aussi l'effet de bord de "comprimer" son argument de sorte que toutes les sous-expressions communes sont partagées. Faites EXAMPLE(OPTIMIZE); pour les exemples.

OPTIMPREFIX Variable

par défaut: [%] - Le préfixe utilisé pour générer des symboles avec la commande OPTIMIZE.

ORDERGREAT (*V1, ..., Vn*) Fonction

définit des alias pour les variables V1, ..., Vn de telle sorte que V1 > V2 > ... > Vn > toute autre variable non mentionnée comme argument. Voir aussi ORDERLESS. Avertissement: faire EXAMPLE(ORDERGREAT); pour quelques particularités.

ORDERGREATP (*exp1,exp2*) Fonction

retourne TRUE si exp2 précède exp1 dans l'ordre défini par la fonction ORDERGREAT (voir DESCRIBE(ORDERGREAT);).

ORDERLESS (*V1, ..., Vn*) Fonction

définit des alias pour les variables V1, ..., Vn de telle sorte que V1 < V2 < ... < Vn < toute autre variable non mentionnée comme argument. Ainsi l'échelle ordonnée complète est: constantes numériques < constants déclarées < scalaires déclarés < premier argument de ORDERLESS < ... < dernier argument de ORDERLESS < variables commençant par A < ... < variables commençant par Z < dernier argument de ORDERGREAT < ... < premier argument de ORDERGREAT < MAINVARs déclarées. Avertissement: faire EXAMPLE(ORDERLESS); pour les particularités. Pour un autre mode d'ordonnancement voir DESCRIBE(MAINVAR);.

ORDERLESSP (*exp1,exp2*) Fonction

retourne TRUE si exp1 précède exp2 dans l'ordre défini par la commande ORDERLESS (voir DESCRIBE(ORDERLESS);).

PART (*exp, n1, ..., nk*) Fonction

traite la forme affichée de exp. Elle procure la partie de exp comme spécifiée par les indices n1,...,nk. La première partie n1 de exp est obtenue, puis la partie n2 de cela, etc. Le résultat est partie nk de ... partie n2 de partie n1 de exp. Ainsi PART(Z+2*Y,2,1) donne 2. PART peut être utilisée pour obtenir un élément d'une liste, une ligne d'une matrice, etc. Si le dernier argument d'une fonction Part est une

liste d'indices alors plusieurs sous-expressions sont choisies, chacune correspondant à un indice de la liste. Ainsi PART(X+Y+Z,[1,3]) est Z+X. PIECE contient la dernière expression sélectionnée après utilisation des fonctions Part. Elle est définie pendant l'exécution de la fonction et peut ainsi être référée dans la fonction elle-même, comme montré ci-dessous. Si PARTSWITCH[FALSE] est mise à TRUE alors END est retournée lorsqu'une partie sélectionnée d'une expression n'existe pas, autrement un message d'erreur est envoyé. Pour les exemples, faites EXAMPLE(PART);

PARTITION (*exp, var*) Fonction

retourne une liste de deux expressions. Elles sont (1) les facteurs de exp (si c'est un produit), les termes de exp (si c'est une somme), ou la liste (si c'est une liste) qui ne contient pas var et, (2) les facteurs, les termes, ou la liste qui le contiennent.

```
(C1) PARTITION(2*A*X*F(X),X);
(D1)                 [ 2 A , X F(X) ]
(C2) PARTITION(A+B,X);
(D2)                 [ A + B , 0 ]
(C3) PARTITION([A,B,F(A),C],A);
(D3)                [[B,C],[A,F(A)]]
```

PARTSWITCH Variable

par défaut: [FALSE] - si TRUE alors END est retournée lorsqu'une partie sélectionnée d'une expression n'existe pas, autrement un message d'erreur est envoyé.

PICKAPART (*exp,depth*) Fonction

va affecter E étiquettes à toutes les sous-expressions de exp, jusqu'à une profondeur entière spécifiée. Utile pour traiter de grandes expressions et pour assigner automatiquement des parties d'une expression à une variable sans avoir à utiliser les fonctions part.

```
(C1) EXP:(A+B)/2+SIN(X^2)/3-LOG(1+SQRT(X+1));
                                                    2
                                                 SIN(X )   B + A
(D1)               - LOG(SQRT(X + 1) + 1) + ------- + -----
                                                    3        2
(C2) PICKAPART(%,1);
(E2)                     - LOG(SQRT(X + 1) + 1)
                                    2
                                 SIN(X )
(E3)                             -------
                                    3
                                 B + A
(E4)                             -----
                                   2
(D4)                         E4 + E3 + E2
```

PIECE Variable

- conserve la dernière expression sélectionnée lors de l'utilisation des fonctions Part. Elle est définie pendant l'exécution de la fonction et peut ainsi être référencée dans la fonction elle-même.

POWERS (*expr, var*) Fonction

donne les puissances de var se produisant dans expr. Pour l'utiliser, faites LOAD(POWERS);. Pour les détails d'utilisation, faites PRINTFILE("powers.usg");.

PRODUCT (*exp, ind, bas, haut*) Fonction

donne le produit des valeurs de exp lorsque l'indice ind varie de bas à haut. L'évaluation est semblable à celle de SUM. Aucune simplification de produits n'est disponible à ce moment. Si haut vaut un de moins que bas, nous avons un "produit vide" et PRODUCT retourne 1 plutôt que de se perdre. Voir aussi DESCRIBE(PRODHACK).

```
(C1)  PRODUCT(X+I*(I+1)/2,I,1,4);
(D1)              (X + 1) (X + 3) (X + 6) (X + 10)
```

REALPART (*exp*) Fonction

donne la partie réelle de exp. REALPART et IMAGPART travaillent sur des expressions impliquant des fonctions trigonométiques et hyperboliques, ainsi que SQRT, LOG, et l'exponentiation.

RECTFORM (*exp*) Fonction

retourne une expression de la forme A + B*%I, où A et B sont purement réels.

REMBOX (*expr, arg*) Fonction

supprime les boîtes de expr en accord avec arg. Si arg est UNLABELED (pas d'étiquette) alors toutes les boîtes non étiquetées sont enlevées. Si arg est le nom d'une certaine étiquette alors seules les boîtes portant cette étiquette sont supprimées. Si arg est omis alors toutes les boîtes, étiquetées ou non, sont supprimées.

SUM (*exp, ind, bas, haut*) Fonction

exécute une sommation sur les valeurs de exp lorsque l'indice ind varie de bas à haut. Si les limites haute et basse diffèrent d'un entier alors chaque terme de la somme est évalué et ajouté aux autres termes. Autrement, si SIMPSUM [FALSE] est TRUE le résultat est simplifié. Cette simplification peut quelquefois produire une forme fermée. Si SIMPSUM est FALSE ou si 'SUM est utilisée, la valeur est une forme somme nominale qui est une représentation de la notation sigma utilisée en mathématique. Si haut vaut un de moins que bas, nous avons une "somme vide" et SUM retourne 0 plutôt que de se perdre. Les sommes peuvent être différenciées, ajoutées, soustraites, ou multipliées avec simplification automatique exécutée. Voyez aussi DESCRIBE(SUMHACK). CAUCHYSUM[FALSE] si TRUE force l'utilisation du produit de Cauchy lors de la multiplication de sommes plutôt que le produit usuel.

Dans le produit de Cauchy l'indice de la sommation interne est une fonction de celui de la sommation externe plutôt que de varier indépendamment. GENINDEX[I] est le préfixe alphabétique utilisé pour générer la variable de sommation suivante. GENSUMNUM[0] est le suffixe numérique utilisé pour générer la variable de sommation suivante. S'il est mis à FALSE alors l'indice consistera seulement en GENINDEX sans suffixe numérique. Faites EXAMPLE(SUM); pour les exemples. Voyez aussi SUMCONTRACT, INTOSUM, BASHINDICES, et NICEINDICES.

LSUM (*exp, ind, list*) Fonction

exécute la somme de EXP pour chaque élément IND de la LIST.

```
(C10) lsum(x^i,i,[1,2,7]);

                                7    2
(D10)                          x  + x  + x
```

Si le dernier élément de l'argument LIST ne s'évalue pas, ou ne s'évalue pas à une liste Maxima alors la réponse est laissée sous forme nominale.

```
(C13) lsum(i^2,i,rootsof(x^3-1));

                        ====
                        \       2
(D13)                    >     i
                        /
                        ====
                                     3
                        i in ROOTSOF(x  - 1)
```

VERB symbole spécial

- l'opposé de "noun", i.e. une forme de fonction qui "fait quelque chose" ("action" - pour la plupart des fonctions, le cas usuel). P. ex. INTEGRATE intègre une fonction, sauf si elle est déclarée être un "noun", auquel cas elle représente l'INTEGRAL de la fonction. Voir NOUN, NOUNIFY, et VERBIFY.

VERBIFY (*f*) Fonction

retourne le nom de la fonction f sous sa forme verbale (voir aussi VERB, NOUN, et NOUNIFY).

6 Simplification

6.1 Définitions pour simplification

APPLY_NOUNS (*exp*) Fonction

provoque l'application des formes nominales dans une expression. P. ex. EXP:'DIFF(X^2/2,X); APPLY_NOUNS(EXP); renvoie X. Cette fonction donne le même résultat que EV(EXP,NOUNS); sauf qu'elle peut être plus rapide et utilise moins de mémoire. Elle peut aussi être utilisée en code traduit, où EV peut causer des problèmes. Notez qu'elle est appelée APPLY_NOUNS, non EV_NOUNS, car ce qu'elle fait est d'APPLiquer les règles correspondant à la forme nominale des opérateurs, ce qui n'est pas une évaluation.

ASKEXP Variable

par défaut: [] contient l'expression sur laquelle ASKSIGN est appelée. Un utilisateur peut entrer une interruption MACSYMA avec ^A et inspecter cette expression afin de répondre aux questions posées par ASKSIGN.

ASKINTEGER (*exp,<arg-optionnel>*) Fonction

exp est toute expression macsyma correcte et arg-optionnel est EVEN, ODD, ou INTEGER (par défaut INTEGER). Cette fonction tente de déterminer d'après la base de donnée si exp est EVEN (paire), ODD (impaire) ou juste un INTEGER (entier). Si elle n'y arrive pas, l'information sera demandée à l'utilisateur et elle sera entrée si possible dans la base de donnée.

ASKSIGN (*exp*) Fonction

tente en premier de déterminer si l'expression spécifiée est positive, négative, ou nulle. Si elle ne le peut, elle pose à l'utilisateur les questions nécessaires pour compléter sa déduction. La réponse de l'utilisateur est enregistrée dans la base de donnée pour la durée du calcul courant (une "C-line"). La valeur de ASKSIGN est POS, NEG ou ZERO.

DEMOIVRE Variable

par défaut: [FALSE] si TRUE provoque

```
%E^(A+B*%I) ==> %E^A*(COS(B)+%I*SIN(B))
```

si B est indépendante de %I. A et B ne sont pas développées. DEMOIVRE:TRUE; est la méthode pour inverser l'effet de EXPONENTIALIZE:TRUE;

DEMOIVRE(exp) provoquera la conversion sans positionner le commutateur ou avoir à ré-évaluer l'expression avec EV.

DOMAIN Variable

par défaut: [REAL] - si mit à COMPLEX, SQRT(X^2) restera SQRT(X^2) au lieu de renvoyer ABS(X). La notion d'un "domaine" de simplification est encore dans l'enfance, et ne contrôle pas beaucoup plus que ceci pour le moment.

EXPAND (*exp*) Fonction

provoque la multiplication des produits de sommes et des sommes exponentielles, le découpage en leurs termes respectifs des numérateurs des expressions rationnelles qui sont des sommes, et la distribution de la multiplication (commutative et non commutative) sur l'addition à tous les niveaux de exp. Pour les polynômes on doit en général utiliser RATEXPAND qui a un algorithme plus efficace (voir DESCRIBE(RATEXPAND);).

MAXNEGEX[1000] et MAXPOSEX[1000] contrôlent respectivement le maximum négatif et positif des exposants, qui seront développés.

EXPAND(exp,p,n) développe exp, en utilisant p pour MAXPOSEX et n pour MAXNEGEX. C'est utile afin de développer une partie et non la totalité d'une expression.

EXPON[0] - l'exposant de la plus grande puissance négative qui est automatiquement développé (indépendamment des appels à EXPAND). Par exemple si EXPON est 4 alors (X+1)**(-5) ne sera pas automatiquement développée.

EXPOP[0] - l'exposant positif le plus grand qui sera automatiquement développé. Ainsi (X+1)**3, dès qu'entré, sera automatiquement développé seulement si EXPOP est plus grand ou égal à 3. Si on veut que (X+1)**N soit développé lorsque N est plus grand que EXPOP alors exécuter EXPAND((X+1)**N) ne fonctionnera que si MAXPOSEX n'est pas inférieur à N. L e drapeau EXPAND utilisé avec EV (voir EV) provoque le développement.

Le fichier SHARE1;FACEXP FASL contient plusieurs fonctions apparentées (FACSUM et COLLECTTERMS en sont deux) qui fournissent à l'utilisateur la possibilité de structurer les expressions par développement contrôlé. De brèves descriptions des fonctions sont disponibles dans SHARE1;FACEXP USAGE. Une démonstration est disponible en faisant BATCH("facexp.mc")$.

EXPANDWRT (*exp,var1,var2,...*) Fonction

développe exp par rapport aux vari. Tous les produits impliquant les vari apparaissent explicitement. La forme retournée ne contiendra plus de produits de sommes des expressions qui ne sont pas indépendantes des vari. Les vari peuvent être des variables, des opérateurs, ou des expressions. Par défaut, les dénominateurs ne sont pas développés, mais ceci peut être contrôlé au moyen du commutateur EXPANDWRT_DENOM. Faites LOAD(STOPEX); pour utiliser cette fonction.

EXPANDWRT_DENOM Variable

par défaut:[FALSE] - contrôle le traitement des expressions rationnelles avec EXPANDWRT. Si TRUE, alors le numérateur et le dénominateur de l'expression seront développés en accord avec les arguments de EXPANDWRT, mais si EXPANDWRT_DENOM est FALSE, alors seul le numérateur sera développé de cette façon. Faites LOAD(STOPEX) pour utiliser cette fonction.

EXPANDWRT_FACTORED (*exp, var1, var2, ..., varN*) Fonction

est semblable à EXPANDWRT, mais traite un peu différemment les expressions qui sont des produits. EXPANDWRT_FACTORED exécutera le développement requis seulement sur les facteurs de exp qui contiennent les variables données par sa liste d'arguments. Faites LOAD(STOPEX) pour utiliser cette fonction.

EXPON Variable

par défaut: [0] - l'exposant de la plus grande puissance négative qui est automatiquement développée (indépendamment d'appels à EXPAND). Par exemple si EXPON vaut 4 alors (X+1)**(-5) ne sera pas automatiquement développée.

EXPONENTIALIZE Variable

défaut: [FALSE] - si TRUE convertit toutes les fonctions circulaires et hyperboliques sous forme exponentielle (DEMOIVRE:TRUE; fait l'inverse). EXPONENTIALIZE(exp) provoque la conversion sous forme exponentielle d'une expression sans positionner le commutateur ou avoir à ré-évaluer l'expression avec EV.

EXPOP Variable

défaut: [0] - le plus grand exposant positif qui sera automatiquement développé. Ainsi (X+1)**3, dès qu'entré, sera automatiquement développé seulement si EXPOP est plus grand ou égal à 3. Si on veut que (X+1)**n soit développé lorsque n est plus grand que EXPOP alors exécuter EXPAND((X+1)**n) ne fonctionnera que si MAXPOSEX n'est pas inférieur à n.

FACTLIM Variable

défaut: [-1] donne le plus grand factoriel qui sera automatiquement développé. Si FACTLIM est -1 alors tous les entiers sont développés.

INTOSUM (*expr*) Fonction

prend tout ce que multiplie une sommation, et le met dans la sommation. Si l'indice est utilisé à l'extérieur de l'expression, alors la fonction essaie de trouver un indice raisonnable, comme ce qui est fait pour SUMCONTRACT. C'est essentiellement l'idée inverse de la propriété OUTATIVE des sommations, mais notez qu'elle ne supprime pas cette propriété, elle la contourne seulement. Dans certains cas, un SCANMAP(MULTTHRU,expr) peut être nécessaire avant le INTOSUM.

LASSOCIATIVE déclaration

- DECLARE(G,LASSOCIATIVE); signale au simplificateur que G est associative à gauche. P. ex. G(G(A,B),G(C,D)) sera simplifié en G(G(G(A,B),C),D).

LINEAR déclaration

- L'une des OPPROPERTIES de MACSYMA. Pour f univariable ainsi déclarée, l'"expansion" F(X+Y) -> F(X)+F(Y), F(A*X) -> A*F(X) se produit, où A est une "constante". Pour des fonctions F de 2 arguments ou plus, la "linéarité" est définie comme dans le cas de 'SUM ou 'INTEGRATE, i.e. F(A*X+B,X) -> A*F(X,X)+B*F(1,X) pour A,B FREEOF X. (LINEAR est juste ADDITIVE + OUTATIVE.)

MAINVAR déclaration

- Vous pouvez déclarer des variables comme étant MAINVAR. L'ordre de classement des atomes est essentiellement: nombres < constantes (p. ex. %E,%PI) < scalaires

< autres variables < mainvars. P. ex. comparez EXPAND((X+Y)^4); avec (DECLARE(X,MAINVAR), EXPAND((X+Y)^4)); .

Note: il faut faire attention si vous décidez d'utiliser la propriété ci-dessus. P. ex. si vous soustrayez une expression dans laquelle X est une MAINVAR de l'une dans laquelle X n'est pas une MAINVAR, une re-simplification p. ex. avec EV(expression,SIMP) peut être nécessaire si une annulation doit se produire. Et aussi, si vous sauvegardez avec SAVE une expression dans laquelle X est une MAINVAR, vous devrez probablement aussi faire SAVE X.

MAXAPPLYDEPTH Variable

défaut: [10000] - la profondeur maximale à laquelle APPLY1 et APPLY2 vont chercher.

MAXAPPLYHEIGHT Variable

défaut: [10000] - la hauteur maximale qu'atteindra APPLYB1 avant de renoncer.

MAXNEGEX Variable

défaut: [1000] - le plus grand exposant négatif qui sera développé par la commande EXPAND (voir aussi MAXPOSEX).

MAXPOSEX Variable

défaut: [1000] - le plus grand exposant qui sera développé par la commande EXPAND (voir aussi MAXNEGEX).

MULTIPLICATIVE déclaration

- Si DECLARE(F,MULTIPLICATIVE) a été exécutée, alors:

(1) Si F est univariable, dès que le simplificateur trouve F appliquée à un produit, F sera distribuée sur ce produit, i.e. F(X*Y); se simplifiera en F(X)*F(Y).

(2) Si F est une fonction de 2 arguments ou plus, la multiplicativité est définie comme multiplicativité sur le premier argument de F, i.e. F(G(X)*H(X),X); sera simplifiée en F(G(X),X)*F(H(X),X). Cette simplification n'a pas lieu si F est appliquée à des expressions de la forme PRODUCT(X[I],I,limit-inf,limite-sup).

NEGDISTRIB Variable

défaut: [TRUE] - si TRUE permet à -1 d'être distribué sur une expression. P. ex. -(X+Y) devient -Y-X. Le mettre à FALSE permet à -(X+Y) d'être affichée sous sa forme. C'est parfois utile mais faites très attention: comme pour le drapeau SIMP, celui-ci est un drapeau que vous ne voudrez pas mettre à FALSE à la légère ou nécessairement pour autre chose qu'une utilisation locale dans votre MACSYMA.

NEGSUMDISPFLAG Variable

défaut: [TRUE] - si TRUE, X-Y s'affiche comme X-Y au lieu de -Y+X. Le mettre à FALSE empêche l'exécution du test d'affichage pour la différence de deux expressions. Une application en est que A+%I*B et A-%I*B peuvent toutes deux être affichée de la même façon.

NOEVAL symbole spécial

- supprime la phase d'évaluation de EV. Utile conjointement avec d'autres commutateurs et pour re-simplifier des expressions sans avoir à les ré-évaluer.

NOUN déclaration

- L'une des options de la commande DECLARE. Elle déclare une fonction comme étant un "nom", ce qui signifie qu'elle ne sera pas évaluée automatiquement.

NOUNDISP Variable

défaut: [FALSE] - si TRUE les NOUN seront affichés avec un guillemet simple (une apostrophe). Ce commutateur est toujours TRUE lors de l'affichage des définitions de fonction.

NOUNS symbole spécial

(EVFLAG) lorsqu'utilisé comme option de la commande EV, convertit toutes les formes "nominales" se produisant dans l'expression sous EV en forme "verbales", i.e. elles sont évaluées. Voir aussi NOUN, NOUNIFY, VERB, et VERBIFY.

NUMER symbole spécial

provoque l'évaluation en virgule flottante de certaines fonctions mathématiques (y compris l'exponentiation) à arguments numériques. Les variables dans exp qui ont reçues des valeurs numériques sont remplacées par leurs valeurs. Et le commutateur FLOAT est activé.

NUMERVAL (*var1, exp1, var2, exp2, ...*) Fonction

donne aux vari les valeurs numériques expi, qui sont évaluées et substituées aux variables dans toutes les expressions les contenant si le commutateur NUMER est TRUE (voir la fonction EV).

OPPROPERTIES Variable

- la liste des propriétés spéciales des opérateurs traités par le simplificateur de MACSYMA: LINEAR, ADDITIVE, MULTIPLICATIVE, OUTATIVE, EVENFUN, ODDFUN, COMMUTATIVE, SYMMETRIC, ANTISYMMETRIC, NARY, LASSOCIATIVE, et RASSOCIATIVE.

OPSUBST Variable

défaut:[TRUE] - si FALSE, SUBST n'essaiera pas de substituer l'opérateur d'une expression. P. ex. (OPSUBST:FALSE, SUBST(X^2,R,R+R[0])); fonctionnera.

OUTATIVE déclaration

- Si DECLARE(F,OUTATIVE) a été exécutée, alors: (1) Si F est univariable, dès que le simplificateur trouve F appliquée à un produit, ce produit sera partitionné en facteurs qui sont constants et en facteurs qui ne le sont pas et les facteurs constants seront retirés. I.e. F(A*X); sera simplifiée en A*F(X) où A est une constante. Les facteurs constants non atomiques ne seront pas éliminés. (2) Si F est une fonction à

2 arguments ou plus, l'"outativité" est définie comme dans le cas de 'SUM ou 'INTEGRATE, i.e. F(A*G(X),X); sera simplifiée en A*F(G(X),X) pour A ne dépendant pas de X. Initialement, 'SUM, 'INTEGRATE, et 'LIMIT sont déclarées comme étant OUTATIVE.

POSFUN déclaration

- POSitive FUNction, p. ex. DECLARE(F,POSFUN); IS(F(X)>0); -> TRUE.

PRODHACK Variable

défaut: [FALSE] - si TRUE alors PRODUCT(F(I),I,3,1); donnera 1/F(2), à cause de l'identité PRODUCT(F(I),I,A,B) = 1/PRODUCT(F(I),I,B+1,A-1) lorsque A>B.

RADCAN (*exp*) Fonction

simplifie exp, qui peut contenir des logs, des exponentielles, et des radicaux, en la convertissant en une forme qui est canonique sur une grande classe d'expressions et un classement donné des variables; c'est à dire, toutes les formes fonctionnellement équivalentes sont (mapped appliquées?) en une forme unique. Pour une classe d'expressions un peu plus importante, RADCAN produit une forme (régulière?, rationnelle?).

Deux expressions équivalentes dans cette classe n'auront pas nécessairement la même apparence, mais RADCAN rendra nulle leur différence. Pour certaines expressions RADCAN peut prendre beaucoup de temps. C'est le coût à payer pour explorer certaines relations entre composants d'une expression pour des simplifications basées sur la factorisation et le développement en fractions partielles des exposants.

%E_TO_NUMLOG (défaut: [FALSE]) - lorsque TRUE, pour un nombre rationnel "r", et une expression "x", %E^(r*LOG(x)) sera simplifiée en x^r .

RADEXPAND[TRUE] mis à FALSE empêche certaines transformations: RADCAN(SQRT(1-X)) restera SQRT(1-X) et ne deviendra pas %I SQRT(X-1). RADCAN(SQRT(X^2-2*X+1)) restera SQRT(X^2-2*X + 1) et ne sera pas transformée en X-1. Faites EXAMPLE(RADCAN); pour voir des exemples.

RADEXPAND Variable

défaut: [TRUE] - si mis à ALL sortira du radical les nièmes racines des facteurs d'un produit qui sont puissances de n. P. ex. si RADEXPAND est ALL, SQRT(16*X^2) deviendra 4*X . Plus particulièrement, considérez SQRT(X^2). (a) Si RADEXPAND est ALL ou que ASSUME(X>0) a été exécuté, SQRT(X^2) deviendra X. (b) Si RADEXPAND est TRUE et DOMAIN est REAL (son défaut), SQRT(X^2) deviendra ABS(X). (c) Si RADEXPAND est FALSE, ou que RADEXPAND est TRUE et DOMAIN est COMPLEX, SQRT(X^2) sera renvoyée. (La notion de DOMAIN avec paramètre REAL ou COMPLEX est toujours dans l'enfance. Notez que son paramétrage n'intervient ici que si RADEXPAND est TRUE).

RADPRODEXPAND Variable

- ce commutateur a été renommé RADEXPAND.

RADSUBSTFLAG Variable

défaut: [FALSE] - si TRUE permet à RATSUBST de faire des substitutions telle que U pour SQRT(X) en X.

RASSOCIATIVE déclaration

- Si DECLARE(G,RASSOCIATIVE); est exécutée, dit au simplificateur que G est associative à droite. P. ex. G(G(A,B),G(C,D)) se simplifiera en G(A,G(B,G(C,D))).

SCSIMP (*exp,règle1, règle2,...,règlen*) Fonction

(Sequential Comparative Simplification [Stoute], simplification séquentielle comparative) prend une expression (son premier argument) et un ensemble d'identités, ou règles (ses autres arguments) et essaie de simplifier. Si une expression plus petite est obtenue, le processus recommence. Dès que toutes les simplifications ont été essayées, elle retourne la réponse originale. Faites EXAMPLE(SCSIMP);.

SIMP Fonction

simplifie exp sans tenir compte de l'option du commutateur SIMP qui empêche la simplification si FALSE.

SIMPSUM Variable

défaut: [FALSE] - si TRUE, le résultat d'une SUM est simplifié. Cette simplification peut quelquefois produire une forme fermée. Si SIMPSUM est FALSE ou si 'SUM est utilisée, la valeur est une forme nominale de somme qui est une représentation de la notation sigma utilisée en mathématiques.

SUMCONTRACT (*expr*) Fonction

combinera toutes les sommes d'une addition qui ont des limites supérieures et inférieures qui diffèrent de constantes. Le résultat sera une expression contenant une sommation pour chaque ensemble de telles sommations ajoutée à tous les termes supplémentaires appropriés qui ont été extraits pour former cette somme. SUMCONTRACT combinera toutes les sommes compatibles et utilisera l'un des indices de l'une des sommes s'il le peut, puis essaiera de former un indice raisonable s'il ne peut en utiliser un existant. Il peut être nécessaire d'exécuter un INTOSUM(expr) avant le SUMCONTRACT.

SUMEXPAND Variable

défaut: [FALSE] - si TRUE, produits de sommes et sommes exponentielles sont convertis en sommes imbriquées. Par exemple:

```
SUMEXPAND:TRUE$
SUM(F(I),I,0,M)*SUM(G(J),J,0,N); ->
                    'SUM('SUM(F(I1)*G(I2),I2,0,N),I1,0,M)
SUM(F(I),I,0,M)^2; -> 'SUM('SUM(F(I3)*F(I4),I4,0,M),I3,0,M)
```

Si FALSE, ils ne sont pas touchés. Voir aussi CAUCHYSUM.

SUMHACK Variable

défaut: [FALSE] - si TRUE alors SUM(F(I),I,3,1); donnera -F(2), par l'identité SUM(F(I),I,A,B) = - SUM(F(I),I,B+1,A-1) lorsque A>B.

SUMSPLITFACT Variable

défaut: [TRUE] - si FALSE MINFACTORIAL sera appliqué après un FACTCOMB.

SYMMETRIC déclaration

- Si DECLARE(H,SYMMETRIC); est exécutée, dit au simplificateur que H est une fonction symétrique. P. ex. H(X,Z,Y) sera simplifiée en H(X, Y, Z). Identique à COMMUTATIVE.

UNKNOWN (*exp*) Fonction

retourne TRUE si et seulement si exp contient un opérateur ou une fonction inconnu(e) du simplificateur intégré.

7 Tracé de courbe

7.1 Définitions pour le tracé de courbe

IN_NETMATH [FALSE] Variable

Si non NIL, alors plot2d sortira une représentation du tracé convenant aux fonctions openplot (de tracé "ouvert").

OPENPLOT_CURVES *liste rest-options* Fonction

Prend une liste de courbes telle que

```
[[x1,y1,x2,y2,...],[u1,v1,u2,v2,...],..]
ou
 [[[x1,y1],[x2,y2],...],.. ]
```

et les trace. Semblable à xgraph_curves, mais utilise les routines de tracé ouvert. Des arguments de symboles supplémentaires peuvent être donnés comme `"{xrange -3 4}"` Ce qui suit est le tracé de deux courbes, utilisant des gros points, la première est étiquetée `jim` et la seconde `jane`.

```
openplot_curves([["{plotpoints 1} {pointsize 6} {label jim}
      {text {xaxislabel {joe is nice}}}"] ,
      [1,2,3,4,5,6,7,8],
      ["{label jane} {color pink } "], [3,1,4,2,5,7]]);
```

D'autres mots-clé spéciaux sont xfun, color, plotpoints, linecolors, pointsize, nolines, bargraph, labelposition, xaxislabel, et yaxislabel.

PLOT2D (*expr,range,...,options,..*) Fonction
PLOT2D ([*expr1,expr2,..,exprn*],*xrange,...,options,..*) Fonction

EXPR est une expression à tracer sur l'axe des y comme fonction d'une variable. RANGE est de la forme [var,min,max] et expr est supposée être une expression à tracer par rapport à VAR. Dans la seconde forme de la fonction une liste d'expressions peut être donnée à tracer par rapport à VAR. La troncature dans la direction des y sera exécutée, pour l'échelle par défaut des y. Elle peut être spécifiée comme option ou en utilisant SET_PLOT_OPTION.

```
plot2d(sin(x),[x,-5,5]);
plot2d(sec(x),[x,-2,2],[y,-20,20],[nticks,200]);
```

xgraph_curves(liste) Fonction

trace le graphe de la liste des 'ensembles de points' donnés dans la liste en utilisant xgraph.

Un ensemble de points peut être de la forme

```
[x0,y0,x1,y1,x2,y2,...] ou
[[x0,y0],[x1,y1],....]
```

Il peut aussi contenir des symboles donnant des étiquettes ou d'autres informations.

```
xgraph_curves([pt_set1,pt_set2,pt_set3]);
```

trace le graphe des trois ensembles de points sous forme de trois courbes.

```
pt_set:append(["NoLines: True","LargePixels: true"],
        [x0,y0,x1,y1,...])
```

supprimera les lignes entre les points de l'ensemble de points [et des suivants], et utilisera de grands pixels. Voyez la page de manuel concernant xgraph pour d'autres options à spécifier.

```
pt_set:append([concat("\"","x^2+y")],[x0,y0,x1,y1,...])
```

fera placer une "étiquette" "x^2+y" pour cet ensemble de points particulier. Le " du début apprend à xgraph que c'est une étiquette.

```
pt_set:append([concat("TitleText: Sample Data")],[x0,...])
```

donne au tracé le titre principal "Sample Data" au lieu de "Maxima PLot".

Pour faire un bargraphe avec des barres larges de .2 unités, et pour tracer deux telles bargraphes peut-être différentes:

```
xgraph_curves(
       [append(["BarGraph: true","NoLines: true","BarWidth: .2"],
               create_list([i-.2,i^2],i,1,3)),
        append(["BarGraph: true","NoLines: true","BarWidth: .2"],
           create_list([i+.2,.7*i^2],i,1,3))
]);
```

Un fichier temporaire '`xgraph-out`' est utilisé.

PLOT_OPTIONS Variable

Les membres de cette liste indiquent les valeurs par défaut du tracé. Elles peuvent être modifiées avec SET_PLOT_OPTION

```
[X, - 3, 3]
[Y, - 3, 3]
```

sont les échelles de x et y respectivement.

[TRANSFORM_XY, FALSE] si non FALSE, devra être la sortie de

```
make_transform([x,y,z], [f1(x,y,z),f2(x,y,z),f3(x,y,z)])
```

qui produit une transformation (from 3 space to 3 space???), qui sera appliquée au graphe. (A built in one is polar_xy which ???) donne le même résultat que

```
make_transform([r,th,z],[r*cos(th),r*sin(th),z])
```

[RUN_VIEWER,TRUE] si non FALSE, demande d'exécuter le programme de visualisation - ne pas seulement sortir un fichier de donnée.

[GRID,30,30] signifie que plot3d devra diviser l'échelle des x en 30 intervalles et de même pour l'échelle des y.

[COLOUR_Z,false] applique les couleurs faites avec plot_format ps.

[PLOT_FORMAT,OPENMATH] est pour plot3d et actuellement OPENMATH, GNUPLOT, PS, et GEOMVIEW sont reconnus.

Il existe des visualiseurs du domaine public de bonne qualité pour ces formats, entre autres openmath, izic, gnuplot, ghostview, et geomview.

Le visualiseur Openmath se trouve dans la distribution, et est basé sur tcl/tk. L'exécutable est '`maxima/bin/omplotdata`'. Le visualiseur vous permet de zoomer,

de glisser, et d'effectuer une rotation (en 3 dimensions). Ce format est aussi celui utilisé par netmath, pour réaliser des tracés de courbe avec Netmath (voir '`http://www.ma.utexas.edu/users/wfs/netmath.html`')

geomview vient du Geometry Center de l'Université du Minnesota, et est disponible depuis '`http://www.geom.umn.edu/software/download/geomview.html`' ou par ftp anonyme depuis '`ftp://ftp.geom.umn.edu/pub/software/geomview/`'. Il n'est pas actuellement aussi joli que izic, mais fournit un excellent support pour de multiples objets et éclairages.

gnuplot est partout comme l'est ghostview. Nous fournissons aussi mgnuplot, l'interface tcl pour gnuplot, qui vous permet la rotation du tracé avec la souris et une échelle.

izic est disponible par ftp depuis zenon.inria.fr. Contactez l'un des auteurs '`{fournier,kajler,mourrain}@sophia.inria.fr.`'

Il a une belle couleur (d'ombre? gouraud shading?), et un très rapide tracé en treillis (wireframe?). Il tourne sous X windows.

PLOT3D (*expr,xrange,yrange,...,options,..*) Fonction
PLOT3D ([*expr1,expr2,expr3*],*xrange,yrange,...,options,..*) Fonction

```
plot3d(2^(-u^2+v^2),[u,-5,5],[v,-7,7]);
```

trace la courbe de z = 2^(-u^2+v^2) avec u et v variant entre [-5,5] et [-7,7] respectivement, et avec u sur l'axe des x, v sur l'axe des y.

Un exemple du second modèle d'arguments est

```
plot3d([cos(x)*(3+y*cos(x/2)),sin(x)*(3+y*cos(x/2)),y*sin(x/2)],
   [x,-%pi,%pi],[y,-1,1],['grid,50,15])
```

qui dessine un ruban de moebius, paramétré par les 3 expressions données en premier argument à plot3d. Un argument additionnel optionnel [grid,50,15] donne le nombre de (grille de rectangles??) dans les directions x et y.

```
/* REal part of z ^ 1/3 */
 plot3d(r^.33*cos(th/3),[r,0,1],[th,0,6*%pi],
     ['grid,12,80],['PLOT_FORMAT,ps],
     ['TRANSFORM_XY,POLAR_TO_XY],['VIEW_DIRECTION,1,1,1.4],
     ['COLOUR_Z,true])
```

Ici VIEW_DIRECTION indique la direction dans laquelle nous prenons une projection. Nous le faisons en fait d'infiniment loin, mais parallèlement à la ligne de VIEW_DIRECTION à l'origine. C'est couramment utilisé avec le plot_format 'ps', car les autres visualiseurs permettent la rotation interactive de l'objet.

Voici un autre exemple du ruban de moebius:

```
plot3d([cos(x)*(3+y*cos(x/2)),
          sin(x)*(3+y*cos(x/2)),y*sin(x/2)],
           [x,-%pi,%pi],[y,-1,1],['grid,50,15]);
```

ou d'une bouteille de klein:

```
plot3d([5*cos(x)*(cos(x/2)*cos(y)+sin(x/2)*sin(2*y)+3.0) - 10.0,
          -5*sin(x)*(cos(x/2)*cos(y)+sin(x/2)*sin(2*y)+3.0),
           5*(-sin(x/2)*cos(y)+cos(x/2)*sin(2*y))],
```

```
[x,-%pi,%pi],[y,-%pi,%pi],['grid,40,40])
```

ou d'un tore

```
plot3d([cos(y)*(10.0+6*cos(x)),
           sin(y)*(10.0+6*cos(x)),
           -6*sin(x)], [x,0,2*%pi],[y,0,2*%pi],
            ['grid,40,40])
```

Nous pouvons aussi sortir sous gnuplot:

```
plot3d(2^(x^2-y^2),[x,-1,1],[y,-2,2],[plot_format,gnuplot])
```

Vous avez parfois besoin de définir une fonction pour tracer l'expression. Tous les arguments de plot3d sont évalués avant d'être passés à plot3d, ainsi essayer de réaliser une expression qui fait juste ce que vous voulez peut être difficile, et il est aussi facile de créer une fonction.

```
M:MATRIX([1,2,3,4],[1,2,3,2],[1,2,3,4],[1,2,3,3])$
f(x,y):=float(M[?round(x),?round(y)]);
plot3d(f,[x,1,4],[y,1,4],['grid,4,4]);
```

PLOT2D_PS (*expr,range*) Fonction

écrit vers pstream une suite de commandes postscript qui tracent EXPR sur RANGE. EXPR devra être une expression à une variable. RANGE devra être de la forme [variable,min,max] sur laquelle tracer expr. Voir CLOSEPS.

CLOSEPS () Fonction

Devra en général être appelée à la fin d'une séquence de commandes de tracé. Elle ferme le flot de sortie courant PSTREAM, et le met à NIL. Elle peut aussi être appelée au début d'un tracé, pour s'assurer de refermer pstream s'il était ouvert. Toutes les commandes qui écrivent sur pstream, l'ouvrent si nécessaire. CLOSEPS est séparée des autres commandes de tracé, car nous pouvons vouloir tracer 2 séries ou superposer plusieurs tracés, et devons ainsi garder le flot ouvert.

SET_PLOT_OPTION (*option*) Fonction

option a le format de l'un des éléments de la liste PLOT_OPTIONS. Ainsi

```
SET_PLOT_OPTION([grid,30,40])
```

changera la grille par défaut utilisée par plot3d. Notez que si le symbole grid a une valeur, alors vous devrez l'"apostropher" ici:

```
SET_PLOT_OPTION(['grid,30,40])
```

de sorte que la valeur ne soit pas substituée.

PSDRAW_CURVE (*ptliste*) Fonction

Dessine une courbe connectant les points de la PTLIST. Celle-ci peut avoir la forme [x0,y0,x1,y1,...] ou [[x0,y0],[x1,y1],...] La fonction JOIN est pratique pour lier ensemble une liste de x et une liste de y.

PSDRAW_CURVE appelle simplement la fonction plus primitive PSCURVE. Voici sa définition:

```
(defun $psdraw_curve (lis)
  (p "newpath")
  ($pscurve lis)
  (p "stroke"))
```

?DRAW2D peut aussi être utilisée pour produire une liste

```
points1:?draw2d(1/x,[.05,10],.03)
```

PSCOM (*com*) Fonction

COM sera insérée dans le fichier poscript, par exemple

```
pscom("4.5 72 mul 5.5 72 mul translate  14 14 scale");
```

8 Entrée et sortie

8.1 Introduction aux entrées/sorties

8.2 Fichiers

- Un fichier est simplement une zone sur un périphérique de stockage particulier qui contient des données ou du texte. Les seuls périphériques de stockage qui sont utilisés sur les machines MC sont les disques et les bandes. Les fichiers sur disques sont groupés figurativement en "répertoires". Un répertoire est juste une liste de tous les fichiers stockés sous un nom donné par l'utilisateur. Faites DESCRIBE(FILEOP); pour avoir des détails sur la façon dont vous pouvez inspecter vos fichiers avec MACSYMA. D'autres commandes qui manipulent des fichiers sont: SAVE, FASSAVE, STORE, LOAD, LOADFILE, RESTORE, UNSTORE, STRINGOUT, BATCH, BATCON, DEMO, WRITEFILE, CLOSEFILE, DELFILE, REMFILE, et APPENDFILE.

8.3 PLAYBACK

Il est possible de ré-exécuter les lignes entrées dans une fenêtre temporaire de déroulement, et ainsi de ne pas perdre le travail courant. Ce peut être fait en tapant FUNCTION E. Un argument numérique lui donne le numéro de la ligne où commencer, à défaut elle repartira de 40 lignes en arrière.

8.4 Définitions pour les entrées/sorties

% Variable

La dernière D-line calculée par MACSYMA (qu'elle soit ou non affichée). Voir aussi %%.

%% Variable

La valeur du dernier calcul exécuté pendant une interruption (MACSYMA-BREAK). Peut aussi être utilisé dans la nième instruction des instructions composées pour référencer la valeur de la (n-1)ème instruction. E.g. F(N):=(INTEGRATE(X^N,X),SUBST(3,X,%%)-SUBST(2,X,%%)); est essentiellement équivalente à F(N):=BLOCK([%%], %%:INTEGRATE(X^N,X), SUBST(3,X,%%)-SUBST(2,X,%%)); Ceci fonctionnera aussi pour communiquer entre la (n-1)ème et la nième instruction (non atomique) d'un BLOCK.

%EDISPFLAG Variable

défaut: [FALSE] - si TRUE, MACSYMA affiche %E en exposant négatif comme un quotient, i.e. %E^-X comme 1/%E^X.

%TH (*i*) Fonction

est le ième calcul précédent. C'est à dire, si l'expression suivante à calculer est D(j), %TH est D(j-i). Utile dans les fichiers BATCH ou pour faire référence à un groupe d'expressions D. Par exemple, si SUM est initialisée à 0 alors FOR I:1 THRU 10 DO SUM:SUM+%TH(I) donnera à SUM la somme des dix dernières expressions D.

"?" Symbole spécial

- Comme préfixe d'une fonction ou d'un nom de variable, signifie que la fonction ou la variable est un jeton LISP, pas un jeton MACSYMA. Deux points d'interrogation accolés, ??, (videra? will flush) la ligne de commande courante de MACSYMA.

ABSBOXCHAR Variable

défaut: [!] est le caractère utilisé pour placer les signes de valeur absolue autour des expressions qui sont hautes de plus d'une ligne.

APPENDFILE (*nom-fichier1, nom-fichier2, DSK, répertoire*) Fonction

est comme WRITEFILE(DSK,répertoire) mais ajoute au fichier dont le nom est spécifié par les deux premiers arguments. Un CLOSEFILE subséquent supprimera le fichier original et renommera le fichier modifié.

BACKUP () Fonction

Pour sauvegarder et voir ce que vous avez fait, voyez PLAYBACK.

BATCH (*spécification-de-fichier*) Fonction

lit et évalue des lignes de commande MACSYMA contenues dans un fichier - Une facilité pour exécuter des lignes de commande stockées dans un fichier disque plutôt qu'en mode ligne usuel. Cette facilité a plusieurs utilisations, par exemple fournir un réservoir de lignes de commande de travail, donner des démonstrations sans erreur, ou aider à organiser ses idées dans des situations complexes de résolution de problème où des modifications peuvent être faites par l'intermédiaire d'un éditeur de texte. Un fichier batch consiste en un ensemble de lignes de commande MACSYMA, chacune se terminant par un ; ou un $, qui peuvent être par la suite séparées par des espaces, des retours chariot, des sauts de page, ainsi de suite. La fonction BATCH lit les lignes de commande du fichier une par une, en les renvoyant sur la console de l'utilisateur, et en les exécutant à leur tour. Le contrôle est retourné à la console de l'utilisateur seulement lorsque de sérieuses erreurs se produisent ou lorsque la fin du fichier est rencontrée. Bien sûr, l'utilisateur peut quitter le traitement du fichier en tapant contrôle-G à tout moment. Les fichiers BATCH peuvent être créés avec un éditeur de texte ou avec la commande STRINGOUT. Faites DESCRIBE(STRINGOUT) pour les détails. DESCRIBE(FILE); et DESCRIBE(FILES); ont des informations supplémentaires sur la façon dont l'argument du fichier est interprété, et sur les fichiers en général.

BATCHKILL Variable

défaut: [FALSE] - si TRUE l'effet de tous les fichiers BATCH précédents est annulé car un KILL(ALL) et un RESET() seront automatiquement exécutés lorsque le suivant sera lu. Si BATCHKILL est lié à un autre atome alors un KILL de la valeur de BATCHKILL sera exécuté.

BATCHLOAD (*spécification-de-fichier*) Fonction
Sauve silencieusement dans le fichier sans sortie sur le terminal ni étiquette.

BATCON (*argument*) Fonction
continue la sauvegarde interrompue dans un fichier.

BATCOUNT Variable
défaut: [0] peut être fixé au numéro de la dernière expression sauvée depuis un fichier. Ainsi BATCON(BATCOUNT-1) recommencera le BATCH depuis l'expression (before the last BATCHed in from before???).

BOTHCASES Variable
défaut: [TRUE] alors MAXIMA reconnaît le texte aussi bien en minuscules qu'en majuscules. Notez, cependant, que les noms de toutes les variables ou fonctions spéciales de MAXIMA sont en majuscules. Le défaut est maintenant TRUE car il rend le code plus lisible, permettant aux utilisateurs d'avoir des noms comme SeriesSolve.

À cause de cela nous mettons en majuscules toutes les variables et fonctions système, et les utilisateurs peuvent les entrer comme ils veulent (en majuscules ou minuscules). Mais toutes les autres variables et fonctions sont sensibles à la casse. Lorsque vous affichez votre programme avec par exemple grind(fundef(f)), vous verrez que les symboles tels que 'IF', 'SIN',... apparaissent tous en majuscules alors que les symboles non système apparaissent dans la casse que vous avez utilisée.

Voici la syntaxe utilisée: si le symbole est rencontré pour la première fois, si la version en majuscules est dans le package et a une liste non triviale de fonction ou propriété, alors le symbole majuscule est utilisé, et il est enregistré comme devant être ainsi utilisé par la suite. Si un symbole est déjà dans le package alors il est juste utilisé.

En fait cela signifie que la plupart des anciens programmes continueront à fonctionner, et que les nouveaux peuvent écrire sIn, Sin, SIN, sin etc et que ce sera toujours interprété comme étant SIN. Cependant s'ils écrivent MySin ce sera différent de MYSIN, car MYSIN n'est pas une fonction ou une variable système.

```
SeriesSolve(f,x):=
   if (f = sin) ...
```

qui est lu comme

```
SeriesSolve(f,x):=
   IF (f = SIN) ...
```

CHANGE_FILEDEFAULTS Variable
défaut: [TRUE] sur les systèmes PDP10, et FALSE ailleurs. Contrôle si l'utilisateur faisant un LOADFILE ou un BATCH a changé les valeurs par défaut de son fichier avec LOADFILE ou BATCH. Le paramètre TRUE est pour ceux qui aiment le style par défaut DDT de fichier. L'option FALSE est pour ceux qui préfèrent les conventions d'autres systèmes d'exploitation, comme le style LISP, ou qui écrivent des packages qui font des LOADFILE ou BATCH ne devant pas interférer avec les valeurs par défaut des fichiers utilisateur.

CLOSEFILE (*nom-fichier1, nom-fichier2*) Fonction

ferme un fichier ouvert avec WRITEFILE et lui donne le nom nom-fichier1 nom-fichier2 (sur une machine Lisp on ne doit dire que CLOSEFILE();). Ainsi pour sauver un fichier contenant l'affichage de toutes les entrées et sorties d'une partie d'une session avec MACSYMA l'utilisateur lance un WRITEFILE, traite avec MACSYMA, puis lance un CLOSEFILE. L'utilisateur peut aussi lancer la fonction PLAYBACK après un WRITEFILE pour sauver l'affichage de transactions précédentes. Notez que ce qui est sauvé de cette façon est une copie de l'affichage des expressions, pas les expressions elles-mêmes. Pour sauver l'expression actuelle sous sa forme interne la fonction SAVE peut être utilisée. L'expression peut alors être récupérée avec la fonction LOADFILE. Pour sauver une expression sous forme linéaire pouvant être plus tard utilisée dans un fichier BATCH, la fonction STRINGOUT est utilisée.

COLLAPSE (*expression*) Fonction

"comprime" son argument en faisant partager les mêmes cellules à ses sous-expressions communes (i.e. égales), en économisant ainsi de l'espace (COLLAPSE est une sous-routine utilisée par la commande OPTIMIZE). Ainsi, appeler COLLAPSE peut être utile avant d'utiliser FASSAVE ou après chargement dans un fichier SAVE. Vous pouvez comprimer ensemble plusieurs expressions avec COLLAPSE([expr1,...,exprN])$. De même vous pouvez comprimer les éléments d'un tableau A en faisant COLLAPSE(LISTARRAY('A))$.

CONCAT (*arg1, arg2, ...*) Fonction

évalue ses arguments et retourne la concaténation de leurs valeurs, résultant en un nom ou une chaîne entre guillemets dont le type est donné par celui du premier argument. Ainsi si X est lié à 1 et D est libre alors CONCAT(X,2)="12" et CONCAT(D,X+1)=D2.

SCONCAT (*arg1, arg2, ...*) Fonction

évalue ses arguments et les concatène en une chaîne. Contrairement à CONCAT, les arguments n'ont pas à être des atomes. Le résultat est une chaîne de Common Lisp.

```
(C5) sconcat("xx[",3,"]:",expand((x+y)^3));
(D5)                    xx[3]:Y^3+3*X*Y^2+3*X^2*Y+X^3
```

La chaîne résultante pourra être utilisé conjointement avec print.

CURSORDISP Variable

défaut: [TRUE] - si TRUE, provoque l'afffichage selon une suite logique des expressions. Ne fonctionne que sur une console qui peut suivre le mouvement du curseur. Si FALSE, les expressions sont simplement affichées ligne par ligne. CURSORDISP est FALSE lorsqu'un WRITEFILE est activé.

DIREC Variable

- La valeur de cette variable est le répertoire par défaut du fichier pour SAVE, STORE, FASSAVE, et STRINGOUT. Il est initialisé au nom de login de l'utilisateur, s'il a un répertoire sur disque, et à l'un des répertoires useri autrement. DIREC détermine sur quel répertoire du disque les fichiers seront écrits.

DISP (*expr1,expr2, ...*) Fonction

est comme DISPLAY mais seule la valeur des arguments sont affichés plutôt que les équations. Utile pour des arguments compliqués qui n'ont pas de noms ou si c'est la valeur de l'argument qui est intéressante, pas son nom.

DISPCON (*tenseur1,tenseur2,...*) Fonction

affiche les propriétés de contraction des tenseuri comme elles ont été données à DEFCON. DISPCON(ALL) affiche toutes les propriétés de contraction qui ont été définies.

DISPLAY (*expr1, expr2, ...*) Fonction

affiche les équations dont le membre gauche est une expri non évaluée, et dont le membre droit est la valeur de l'expression centrée sur la ligne. Cette fonction est utile dans les blocs et les instructions FOR afin d'avoir l'affichage des résultats intermédiaires. Les arguments de DISPLAY sont en général des atomes, des variables indicées, ou des appels de fonction. Voir la fonction DISP.

```
(C1) DISPLAY(B[1,2]);
                                      2
                           B     = X - X
                            1, 2
(D1)                              DONE
```

DISPLAY2D Variable

défaut: [TRUE] - si FALSE provoquera en affichage standard une chaîne à une dimension plutôt que celui d'une forme à deux dimensions. Peut être bénéfique pour les utilisateurs voulant conserver du papier sur les consoles d'impression.

DISPLAY_FORMAT_INTERNAL Variable

défaut: [FALSE] - si TRUE provoque l'affichage des expressions sans les transformer de façon que leur représentation mathématique interne soit cachée. L'affichage correspond alors à ce que retourne la commande INPART plutôt que la commande PART. Exemples:

```
Utilisateur     PART       INPART
a-b;            A - B     A + (- 1) B
                A            - 1
a/b;            -         A B
                B
                           1/2
sqrt(x);        SQRT(X)    X
                4 X        4
X*4/3;          ---        - X
                3         3
```

DISPTERMS (*expr*) Fonction

affiche son argument en parties l'une sous l'autre. C'est à dire que l'opérateur de 'expr' est affiché en premier, puis chaque terme d'une somme, ou facteur d'un produit, ou

partie d'une expression plus générale, est affiché séparément. Utile si expr est trop grande pour être affichée autrement. Par exemple si P1, P2, ... sont de très grandes expressions alors le programme d'affichage peut manquer d'espace de stockage en essayant d'afficher P1+P2+... d'un seul coup. Cependant, DISPTERMS(P1+P2+...) affichera P1, puis sous elle P2, etc. Si DISPTERMS n'est pas utilisée, et si une expression exponentielle est trop grande pour être affichée comme A**B elle apparaîtra comme EXPT(A,B) (ou comme NCEXPT(A,B) dans le cas A^^B).

DSKALL Variable

défaut: [] - si TRUE provoquera l'écriture périodique sur disque des valeurs, fonctions, tableaux et règles en plus des expressions étiquetées. TRUE est la valeur par défaut alors que si DSKALL est FALSE seules les expressions étiquetées seront écrites.

ERROR_SIZE Variable

défaut: [20 pour un terminal d'affichage, 10 pour les autres]. contrôle la taille des messages d'erreur. Par exemple, soit U:(C^D^E+B+A)/(COS(X-1)+1); . U a une taille d'erreur de 24. Donc si ERROR_SIZE a une valeur < 24 alors

```
(C1) ERROR("The function", FOO,"doesn't like", U,"as input.");
prints as:
The function FOO doesn't like ERREXP1 as input.
If ERROR_SIZE>24 then as:
                                  E
                                 D
                                C   + B + A
The function FOO doesn't like -------------- as input.
                              COS(X - 1) + 1
```

Les expressions plus grandes que ERROR_SIZE sont remplacées par des symboles, et ces symboles sont (liés ?set to?) aux expressions. Les symboles sont pris dans une liste établie par l'utilisateur

```
ERROR_SYMS:[ERREXP1,ERREXP2,ERREXP3]
```

. La valeur par défaut de ce commutateur dépend de l'expérience de l'utilisateur. Si vous trouvez cette valeur trop grande ou trop petite pour vous, envoyez un mèl à MACSYMA.

ERROR_SYMS Variable

défaut: [ERREXP1,ERREXP2,ERREXP3] - Dans les messages d'erreur, les expressions plus grandes que ERREUR_SIZE sont remplacées par des symboles, et ces symboles sont (liés ?set to?) aux expressions. Ils sont pris dans une liste ERREUR_SYMS, et sont initialement ERREXP1, ERREXP2, ERREXP3, etc. Après l'affichage d'un message d'erreur, p. ex. "The function FOO doesn't like ERREXP1 as input.", l'utilisateur peut taper ERREXP1; pour voir l'expression. ERREUR_SYMS peut être lié par l'utilisateur à un ensemble différent de symboles, s'il le désire.

EXPT (*A,B*) Fonction

si une expression exponentielle est trop grande pour être affichée comme A^B elle le sera sous la forme EXPT(A,B) (ou NCEXPT(A,B) pour A^^B).

EXPTDISPFLAG Variable

défaut: [TRUE] - si TRUE, MACSYMA affiche les expressions aux exposants négatifs en utilisant les quotients, p. ex. X**(-1) devient 1/X.

FASSAVE (*args*) Fonction

est semblable à SAVE mais produit un fichier FASL dans lequel le partage des sous-expressions qui sont partagées dans le noyau est préservé dans le fichier créé. Par conséquent, les expressions qui ont des sous-expressions communes consommeront moins d'espace lorsqu'elles seront rechargées à partir d'un fichier créé par FASSAVE plutôt que par SAVE. Les fichiers créés par FASSAVE sont rechargés avec LOADFILE, exactement comme les fichiers créés avec SAVE. FASSAVE retourne une liste de la forme [<nom de fichier>,<taille du fichier en blocs>,...] où ... sont les objets sauvées. Des avertissements sont affichés dans le cas de gros fichiers. FASSAVE peut être utilisée pendant que WRITEFILE s'exécute.

FILEDEFAULTS () Fonction

retourne le nom de fichier courant par défaut, dans le format utilisé par l'implantation Macsyma spécifique (voir DESCRIBE(FILE) pour connaître ce format). C'est la spécification de fichier utilisée par LOADFILE, BATCH, et nombre d'autres commandes d'accès aux fichiers.

FILEDEFAULTS('fich) - fixe les valeurs par défauts des fichiers à "fich". L'argument de FILEDEFAULTS est évalué car il prévu que la commande sera surtout utilisée dans des programmes. Le "fich" doit être un fichier réel, afin qu'on puisse utiliser cette fonction, p. ex. si le but réel est seulement de définir le champ "périphérique" sur quelque chose, et qu'on ne se préoccupe pas du paramétrage des autres champs.

FILENAME Variable

défaut: [] - La valeur de cette variable est le premier nom des fichiers qui sont générés par le processus automatique de stockage sur disque. La valeur par défaut est constituée des trois premiers caractères du nom de login de l'utilisateur concaténés avec le plus petit entier non utilisé, p. ex. ECR1.

FILENAME_MERGE ("*nom-fichier1*","*nom-fichier2*",...); Fonction

fusionne ensemble les nom-fichiers, ce qui signifie que la fonction retourne "nom-fichier1" sauf que les composants manquants proviennent des composants correspondants du "nom-fichier2", et s'ils manquent ici aussi, alors de "nom-fichier3".

FILENUM Variable

défaut: [0] - Le nom par défaut du second fichier pour les fichiers créés par SAVE, STRINGOUT, ou FASSAVE si aucun nom n'est spécifié par l'utilisateur. C'est un entier, et il est incrémenté de un chaque fois qu'un nouveau fichier est écrit.

FILE_SEARCH Variable

- C'est une liste de fichiers nommant les répertoires à rechercher par LOAD et nombre d'autres fonctions. La valeur par défaut est une liste SHARE des divers répertoires utilisé par Macsyma.

FILE_SEARCH("nom-fichier"); recherche dans ces répertoires et périphériques spécifiés par les variables FILE_SEARCH_LISP, FILE_SEARCH_MAXIMA et FILE_SEARCH_DEMO, et retourne le nom du premier fichier trouvé. Cette fonction est invoquée par la fonction LOAD, (which is why LOAD("FFT") finds and loads share/fft.mac??). Vous pouvez ajouter un chemin à la liste appropriée. Notez que le format des chemins permet de spécifier des extensions et des chemins multiples.

```
"/home/wfs/###.{o,lisp,mac,mc}"
"/home/{wfs,joe}/###.{o,lisp,mac,mc}"
```

Les '###' sont remplacés par le nom réel du fichier passé. FILE_SEARCH vérifie d'abord si ce nom existe, avant de le substituer dans les diverses configurations.

FILE_STRING_PRINT Variable

défaut: [FALSE] sur MC, [TRUE] ailleurs. Si TRUE, les noms-fichier sont sortis comme des chaînes; si FALSE, comme des listes. Par exemple, lorsqu'un fichier hors du noyau est chargé dans MACSYMA (p. ex. le package LIMIT), le message apparaît sur MC dans un format de liste comme:

LIMIT FASL DSK MACSYM being loaded

et en format chaîne comme:

DSK:MACSYM;LIMIT FASL being loaded

Le format chaîne est comme les spécifications de fichier de haut niveau (DDT).

FILE_TYPE ("*nom-fichier*") Fonction

retourne FASL, LISP, ou MACSYMA, selon le type du fichier. FASL signifie un fichier Lisp compilé, qui a en principe une extension de .VAS (in NIL?).

GRIND (*arg*) Fonction

affiche arg dans un format plus lisible que la commande STRING. Il retourne une D-ligne comme valeur.

Le commutateur GRIND, par défaut: [FALSE], si TRUE fera utiliser le mode "grind" par les commandes STRING, STRINGOUT, et PLAYBACK au lieu du mode "chaîne". Pour PLAYBACK, le mode "grind" peut aussi être activé (pour traiter les lignes d'entrée) en spécifiant l'option GRIND.

IBASE Variable

défaut: [10] - la base pour l'entrée des nombres.

INCHAR Variable

défaut: [C] - le préfixe alphabétique des noms des expressions entrées par l'utilisateur.

LDISP (*expr1,expr2,...*) Fonction

est comme DISP mais génère aussi des étiquettes intermédiaires.

LDISPLAY (*expr1,expr2,...*) Fonction

est comme DISPLAY mais génère aussi des étiquettes intermédiaires.

LINECHAR Variable

défaut: [E] - le préfixe alphabétique des noms des expressions intermédiaires affichées.

LINEDISP Variable

défaut: [TRUE] - Permet l'utilisation de lignes graphiques dans le dessin des équations dans les systèmes qui les supportent (p. ex. la Machine Lisp). Peut être désactivé en passant LINEDISP à FALSE. Il est automatiquement désactivé pendant un WRITEFILE.

LINEL Variable

défaut: [] - le nombre de caractères qui sont affichés sur une ligne. Il est initialement fixé par MACSYMA à la longueur de la ligne du type de terminal utilisé (tant qu'il est connu) mais peut être modifié à tout moment par l'utilisateur. Celui-ci peut aussi avoir à le refixer dans DDT avec :TCTYP.

LOAD ("*nom-fichier*") Fonction

prend un argument, un nom de fichier représenté comme une "chaîne" (i.e. entre guillemets), ou une liste (p. ex. entre crochets), puis localise et charge le fichier indiqué. Si aucun répertoire n'est spécifié, il recherche les répertoires SHAREi et tous les autres répertoires listés dans la variable FILE_SEARCH et charge le fichier indiqué.

LOAD("EIGEN") chargera le package eigen sans que l'utilisateur ait besoin de connaître les détails du package, s'il a été compilé, traduit, sauvé, ou "fassauvé", i.e. LOAD fonctionnera sur tout fichier LOADFILE et BATCH.

Note: LOAD utilisera BATCHLOAD s'il trouve que le fichier est un fichier BATCH (ce qui signifie qu'il le fera en "silence" sans sortie sur le terminal ni étiquette).

D'autres commandes MACSYMA de chargement de fichiers sont: LOADFILE, RESTORE, BATCH, et DEMO. Faites DESCRIBE(commande); pour les détails. LOADFILE et RESTORE travaillent sur des fichiers écrits avec SAVE; BATCH et DEMO pour ceux écrits avec STRINGOUT ou créés comme listes de commandes avec un éditeur de texte. Si LOAD ne trouve pas le fichier, vérifiez la valeur de FILE_SEARCH pour vous assurez qu'elle contient un masque approprié.

```
(C4) load("eigen");
MACSYMA BUG: Unknown file type NIL

Error: macsyma error
Error signalled by MEVAL1.
Broken at $LOAD.  Type :H for Help.
MAXIMA>>:q

By examining the file system we find the file is actually in
/public/maxima/share/eigen.mc.   So we add that to the file_search
path.   This can be done at start up (see init.lsp) or,
else it can be done and then the system resaved
once it has been customized for local directories and pathnames.
At lisp level we would do
```

```
(in-package "MAXIMA")
(setq $file_search ($append (list '(mlist)
        "/tmp/foo.mac" "/tmp/foo.mc") $file_search))
and at maxima level:

(C5) file_search:append(["/public/maxima/share/foo.mc"],
        file_search)$
(C6) load("eigen");

batching /usr/public/maxima/share/eigen.mc
(D6)                    #/public/maxima/share/eigen.mc
(C7) eigenvaleurs(matrix([a,b],[c,d]));
                2                     2
       - SQRT(D  - 2 A D + 4 B C + A ) + D + A
(D7) [[-------------------------------------,
                          2

                                 2                     2
                          SQRT(D  - 2 A D + 4 B C + A ) + D + A
                          -------------------------------------], [1, 1]]
                                            2
```

LOADFILE (*nom-fichier*) Fonction

charge le fichier désigné par ses arguments. Cette fonction peut être utilisée pour rappeler des quantités qui ont été stockées par une session MACSYMA précédente avec les fonctions SAVE ou STORE. Spécifiez le nom du chemin comme avec votre système d'exploitation. Pour Unix ce serait "/home/wfs/foo.mc" par exemple.

LOADPRINT Variable

défaut: [TRUE] - dirige l'affichage des messages accompagnant le chargement des fichiers. Les options suivantes sont disponibles:

TRUE signifie toujours afficher le message; 'LOADFILE signifie afficher seulement lorsque la commande LOADFILE est utilisée; 'AUTOLOAD signifie afficher seulement lorsqu'un fichier est automatiquement chargé (p. ex. le fichier d'intégration SIN FASL); FALSE signifie ne jamais afficher le message de chargement.

NOSTRING (*arg*) Fonction

affiche toutes les lignes entrées en rejouant plutôt qu'en les chaînant. Si arg est GRIND alors l'affichage sera dans un format plus lisible. On peut inclure un nombre quelconque d'options comme dans PLAYBACK([5,10],20,TIME,SLOW).

OBASE Variable

défaut: [10] la base d'affichage des nombres.

OUTCHAR Variable

défaut: [D] - le préfixe alphabétique des noms des expressions en sortie.

PACKAGEFILE Variable

défaut:[FALSE] - Les concepteurs de packages qui utilisent SAVE, FASSAVE, ou TRANSLATE pour créer des packages (des fichiers) pour ceux qui peuvent vouloir fixer PACKAGEFILE:TRUE$ pour empêcher l'ajout d'information aux listes d'information de MACSYMA (p. ex. VALUES, FUNCTIONS) sauf si nécessaire lorsque le fichier est chargé. De cette façon, le contenu du package ne restera pas dans le chemin de l'utilisateur lorsqu'il ajoutera ses propres données. Notez que cela ne résoud pas le problème des possibles conflits de noms. Notez aussi que le drapeau affecte simplement ce qui est la sortie du fichier package. Fixer le drapeau à TRUE est aussi utile pour créer les fichiers d'initialisation de MACSYMA.

PARSEWINDOW Variable

défaut:[10] - le nombre maximum de "tokens lexicaux" qui sont affichés de chaque côté du point-erreur lorsqu'une erreur de syntaxe (d'analyse) se produit. Cette option est spécialement utile sur les terminaux lents. La mettre à -1 provoque l'affichage de la chaîne entrée tout entière lorsqu'une erreur se produit.

PFEFORMAT Variable

défaut: [FALSE] - si TRUE provoque l'affichage des nombres rationnels sous forme linéaire et les dénominateurs qui sont des entiers comme multiplicateurs de nombres rationnels.

PRINT (*exp1, exp2, ...*) Fonction

évalue et affiche ses arguments l'un après l'autre "sur une ligne" en commençant le plus à gauche. Si le expi n'est pas lié ou est précédé d'un guillemet unique ou est entouré de guillemets alors il est affiché littéralement. Par exemple, PRINT("THE VALUE OF X IS ",X). La valeur retournée par PRINT est la valeur de son dernier argument. Aucune ligne intermediaire n'est créée (pour "imprimer" des fichiers, voir la fonction PRINTFILE).

SPRINT (*exp1, exp2, ...*) Fonction

évalue et affiche ses arguments l'un après l'autre "sur une ligne" en commençant le plus à gauche. Les nombres sont affichés avec le (signe '-' à droite près du nombre '-' right next to the number?), sans se préoccuper de la longueur de la ligne.

TCL_OUTPUT (*LIST INDEX* &optional-skip) Fonction

affiche une liste TCL basée sur l'extraction par LIST de (the INDEX slot?). Ici skip vaut 2 par défaut, signifiant qu'un élément sur deux sera affiché si l'argument est sous forme d'une liste de nombres, plutôt qu'une liste de listes. Par exemple:

```
TCL_OUTPUT([x1,y1,x2,y2,x3,y3],1) --> {x1 x2 x3 }
TCL_OUTPUT([x1,y1,x2,y2,x3,y3],2) --> {y1 y2 y3 }
TCL_OUTPUT([1,2,3,4,5,6],1,3) --> {1 4}
TCL_OUTPUT([1,2,3,4,5,6],2,3) --> {2 5}
```

READ (*chaîne1, ...*) Fonction

affiche ses arguments, puis lit et évalue une expression. Par exemple: A:READ("ENTER THE NUMBER OF VALUES").

READONLY (*chaîne1,...*) Fonction

affiche ses arguments, puis lit une expression (qui contrairement à READ n'est pas évaluée).

REVEAL (*exp,profondeur*) Fonction

affichera exp à la profondeur entière spécifiée en indiquant la longueur de chaque partie. Les sommes seront affichées comme Sum(n) et les produits comme Product(n) où n est le nombre de sous-parties de la somme ou du produit. Les exponentielles seront affichées comme Expt.

```
(C1) INTEGRATE(1/(X^3+2),X)$
(C2) REVEAL(%,2);
(D2)                    Negterm + Quotient + Quotient
(C3) REVEAL(D1,3);
                                          ATAN          LOG
(D3)                 - Quotient + ---------- + ----------
                                  Product(2)   Product(2)
```

RMXCHAR Variable

défaut: []] - Le caractère utilisé pour afficher le délimiteur (droit) d'une matrice (voir aussi LMXCHAR).

SAVE (*nom-fichier,arg1, arg2,...,argi*) Fonction

sauve sur disque les quantités décrites par ses arguments et les conserve aussi dans le noyau. Les arg sont les expressions à sauvegarder. ALL est le plus simple, mais notez que sauver TOUT sauvera le contenu tout entier de votre MACSYMA, qui dans le cas d'un gros calcul peut donner un gros fichier. VALUES, FUNCTIONS, ou tout autre item des INFOLISTS (faites DESCRIBE(INFOLISTS); pour la voir) peut être sauvé avec SAVE, comme le peuvent les fonctions et les variables par leur nom. Les lignes C et D peuvent aussi être sauvées, mais il vaut mieux leur donner des noms explicites, ce qui peut se faire sur la ligne de commande, p. ex. SAVE(RES1=D15);. Les fichiers sauvés par SAVE seront rechargés avec LOADFILE. SAVE retourne le nom du chemin où les items ont été sauvés.

SAVEDEF Variable

défaut: [TRUE] - si TRUE conservera la version de MACSYMA d'une fonction utilisateur lorsque la fonction sera transférée par TRANSLATE. Permet l'affichage de la définition par DISPFUN et son édition. Si SAVEDEF est FALSE, les noms des fonctions transférées sont ôtés de la liste des FUNCTIONS.

SHOW (*exp*) Fonction

affiche exp avec les objets indicés qu'elle contient montrés avec des indices inférieurs covariants et des indices supérieurs contravariants. Les indices dérivés seront affichés en indices inférieurs, séparés des indices covariants par une virgule.

SHOWRATVARS (*exp*) Fonction

retourne une liste des RATVARS (variables CRE) de exp.

STARDISP Variable

défaut: [FALSE] - si TRUE provoque l'affichage explicite de la multiplication avec une * entre opérandes.

STRING (*expr*) Fonction

convertit expr en notation linéaire de MACSYMA (semblable à celle du FORTRAN) exactement comme si elle avait été entrée et placée dans le tampon pour une possible édition (auquel cas expr est en général Ci). L'expression obtenue avec STRING ne pourra être utilisée dans un calcul.

STRINGOUT (*args*) Fonction

sortira une expression dans un fichier sous forme linéaire. De tels fichiers sont alors utilisés par les commandes BATCH ou DEMO.

STRINGOUT(spécification-de-fichier, A1, A2, ...) place dans un fichier donné par la spécification-de-fichier ([nom-fichier1,nom-fichier2,DSK, répertoire]) les valeurs données par A1,A2,.. dans un format MACSYMA lisible. La spécification-de-fichier peut être omise, dans ce cas les valeurs par défaut seront utilisées. Les Ai sont en général des étiquettes C ou peuvent être (INPUT meaning the valeur of all C labels?). Une autre option est de faire des ai des FUNCTIONS qui provoqueront la sortie sur fichier de toutes les définitions de fonctions utilisateur (i.e. toutes celles récupérées par DISPFUN(ALL)). De même les ai peuvent être des VALUES, et toutes les variables auxquelles l'utilisateur a affecté des valeurs seront sorties. Les ai peuvent aussi être une liste [m,n] ce qui signifie chaîner toutes les étiquettes de m à n inclusivement. Cette fonction peut être utilisée pour créer un fichier d'instructions FORTRAN en faisant un peu de simple édition sur les expressions à sortir.

Si le commutateur GRIND est mis à TRUE, alors STRINGOUT utilisera le format GRIND au lieu du format STRING.

Note: un STRINGOUT peut être fait pendant l'exécution d'un WRITEFILE.

TEX (*expr*) Fonction
TEX(expr,nom-fichier) Fonction
TEX(étiquette,nom-fichier) Fonction

Dans le cas d'une étiquette, un numéro gauche d'une équation sera produit. Pour un nom de fichier, la sortie sera ajoutée au fichier.

```
(C1) integrate(1/(1+x^3),x);

                                          2 x - 1
                              2         ATAN(-------)
                           LOG(x  - x + 1)   SQRT(3)    LOG(x + 1)
(D1)                     - --------------- + ---------- + ----------
                                  6          SQRT(3)          3
(C2) tex(d1);
```

```
$$-{{\log \left(x^{2}-x+1\right)}\over{6}}+{{\arctan {{2\>x-1}\over{
 \sqrt{3}}}}\over{\sqrt{3}}}+{{\log \left(x+1\right)}\over{3}}\leqno{
 \tt (D1)}$$
(D2)                                   (D1)
(C6) tex(integrate(sin(x),x));

$$-\cos x$$
(D6)                                   FALSE
(C7) tex(d1,"/tmp/jo.tex");

(D7)                                   (D1)
```

où la dernière expression sera ajoutée au fichier '`/tmp/jo.tex`'

SYSTEM(commande) Fonction

Exécute COMMAND en sous-processus. La commande sera passée au shell par défaut pour exécution. SYSTEM n'est pas connu de tous les systèmes d'exploitation, mais existe généralement dans l'environnement unix.

Si hist est une liste de fréquences que vous voulez tracer comme bargraphe avec xgraph d'unix,

```
(C1) (with_stdout("_hist.out",
           for i:1 thru length(hist) do (
             print(i,hist[i]))),
       system("xgraph -bar -brw .7 -nl < _hist.out"));
```

Afin que le tracé se fasse en arrière-plan (retournant le contrôle à maxima) et pour supprimer le fichier temporaire à la fin, faites :

```
system("(xgraph -bar -brw .7 -nl < _hist.out;  rm -f _hist.out)&")
```

TTYOFF Variable

défaut: [FALSE] - si TRUE arrête l'affichage de la sortie sur la console.

WITH_STDOUT(fichier,stmt1,stmt2,...) macro

Ouvre le fichier puis évalue stmt1, stmt2, ... Tout affichage sur la sortie standard va dans le fichier au lieu du terminal. Retourne toujours FALSE. Notez que display2d doit être "false", sinon l'affichage fera des choses comme "- 3" au lieu de "-3".

```
mygnuplot(f,var,range,nombre_ticks):=
 block([numer:true,display2d:false],
 with_stdout("/tmp/gnu",
   for x:range[1] thru range[2] step
                (range[2]-range[1])/nombre_ticks
       do (print(x,at(f,var=x)))),
 system("echo \"set data style lines; set title '",
        f,"' ;plot '/tmp/gnu'
;pause 10 \" | gnuplot"));

(C8) with_stdout("/home/wfs/joe",
      n:10,
```

```
      for i:8 thru n
        do(print("factorial(",i,") gives ",i!)));
(D8)                                  FALSE
(C9) system("cat /home/wfs/joe");
factorial( 8 ) gives  40320
factorial( 9 ) gives  362880
factorial( 10 ) gives  3628800
(D9)                                    0
```

WRITEFILE (*DSK*, *répertoire*) Fonction

ouvre un fichier en écriture. Sur une Machine Lisp on utilise WRITEFILE("nom-fichier"). Toute interaction entre l'utilisateur et MACSYMA est alors enregistrée dans ce fichier, exactement comme elle l'est sur la console. Un tel fichier est une transcription de la session, et n'est pas rechargeable ou "batchable" à nouveau dans MACSYMA (voyez aussi CLOSEFILE.)

9 Calculs en virgule flottante

9.1 Définitions pour virgule flottante

BFFAC (*exp,n*) Fonction

version BFLOAT de la fonction factorielle (Gamma décalée). Le second argument est le nombre de chiffres à retenir et renvoyer, c'est une bonne idée d'en demander deux de plus. Cette fonction est disponible en faisant LOAD(BFFAC); .

ALGEPSILON Variable

La valeur par défaut est 10^-8. La valeur de ALGEPSILON est utilisée par ALGSYS.

BFLOAT (*X*) Fonction

convertit tous les nombres et les fonctions de nombres en nombres "bigfloat". Fixer FPPREC[16] à N, donne une précision "bigfloat" de N chiffres. Si FLOAT2BF[FALSE] est FALSE un message d'avertissement est affiché lorsqu'un nombre en virgule flottante est converti en un nombre "bigfloat" (puisque ceci peut mener à une perte de précision).

BFLOATP (*exp*) Fonction

est TRUE si exp est un nombre "bigfloat", est sinon FALSE.

BFPSI (*n,z,fpprec*) Fonction

donne des polygammas d'argument réel arg et d'ordre entier. Pour les digammas, BFPSI0(z,fpprec) est plus direct. Notez que -BFPSI0(1,fpprec) fournit %GAMMA en BFLOAT. Pour utiliser ceci faire LOAD(BFFAC);

BFTORAT Variable

défaut: [FALSE] - contrôle la conversion de bfloats en nombres rationnels. Si

```
BFTORAT:FALSE
```

RATEPSILON sera utilisé pour contrôler la conversion (ce qui donne des nombres rationnels relativement petits). Si

```
BFTORAT:TRUE
```

le nombre rationnel généré représentera avec précision le bfloat.

BFTRUNC Variable

défaut: [TRUE] - empêchera l'affichage des zéros terminaux des nombres bigfloat. Ainsi, si BFTRUNC:FALSE, BFLOAT(1); l'affiche 1.000000000000000B0. Sinon il est affiché 1.0B0.

CBFAC (*z,fpprec*) Fonction

un factoriel pour bfloats complexes. Peut être utilisé en faisant LOAD(BFAC);. Pour plus de détails voir share2/bfac.usg.

FLOAT (*exp*) Fonction

convertit les entiers, les nombres rationnels et les bigfloats de exp en nombres en virgule flottante. (C'est aussi un EVFLAG It is also an EVFLAG?), FLOAT provoque la conversion des nombres rationnels non entiers et des nombres "bigfloat" en nombres en virgule flottante.

FLOAT2BF Variable

défaut: [FALSE] - un message d'avertissement est affiché lorsqu'un nombre en virgule flottante est converti en un nombre "bigfloat" (car cela peut mener à une perte de précision).

FLOATDEFUNK Fonction

est un utilitaire créant des fonctions en virgule flottante depuis une expression mathématique. Prend l'expression entrée et la FLOAT, puis l'OPTIMIZE, enfin insère des déclarations MODE_DECLARE pour toutes les variables. C'est LA façon d'utiliser ROMBERG, PLOT2, INTERPOLATE, etc. p. ex. EXP:some-hairy-macsyma-expression;

```
FLOATDEFUNK('F,['X],EXP);
```

va définir la fonction F(X) pour vous. Faites PRINTFILE(MCOMPI,DOC,MAXDOC); pour les détails.

FLOATNUMP (*exp*) Fonction

est TRUE si exp est un nombre en virgule flottante, sinon FALSE.

FPPREC Variable

défaut: [16] - Floating Point PRECision. Peut être fixé à une valeur entière représentant la précision voulue.

FPPRINTPREC Variable

défaut: [0] - le nombre de chiffres à afficher pour un nombre "bigfloat", rendant possible le calcul avec un grand nombre de chiffres de précision, mais la réponse est affichée avec un plus petit nombre de chiffres. Si FPPRINTPREC est 0 (le défaut), ou >= FPPREC, alors la valeur de FPPREC contrôle le nombre de chiffres de l'affichage. Cependant, si FPPRINTPREC a une valeur entre 2 et FPPREC-1, il contrôle le nombre de chiffres utilisés (le nombre minimal de chiffres utilisés est 2, un à gauche de la virgule et un à droite; la valeur 1 pour FPPRINTPREC est incorrecte).

?ROUND (*x*,&diviseur-optionnel) Fonction

arrondit le nombre X en virgule flottante à l'entier le plus proche. L'argument doit être un flottant système régulier, pas un bigfloat. Le ? débutant le nom indique que c'est une fonction Common Lisp normale.

```
(C3) ?round(-2.8);
(D3)                                  - 3
```

?TRUNCATE (*x*,&diviseur-optionnel) Fonction

tronque le nombre en virgule flottante X vers 0, pour en faire un entier. L'argument doit être un flottant système régulier, pas un bigfloat. Le ? débutant le nom indique que c'est une fonction Common Lisp normale.

```
(C4) ?truncate(-2.8);
(D4)                                - 2
(C5) ?truncate(2.4);
(D5)                                 2
(C6) ?truncate(2.8);
(D6)                                 2
```

ZUNDERFLOW Variable

défaut: [TRUE] - si FALSE, une erreur sera signalée si un "dépassement négatif" (underflow) en virgule flottante se produit. Actuellement dans NIL Macsyma, toutes les erreurs de dépassement en virgule flottante, et de division par zéro sont signalées, et ce commutateur est ignoré.

10 Contextes

10.1 Définitions pour les contextes

ACTIVATE (*cont1, cont2, ...*) Fonction

provoque l'activation des contextes conti spécifiés. Les faits de ces contextes sont utilisés pour faire des déductions et récupérer des informations. Ils ne sont pas listés lorsque FACTS(); est exécutée. La variable ACTIVECONTEXTS est la liste des contextes qui ont été activés par la fonction ACTIVATE.

ACTIVECONTEXTS Variable

défaut: [] - est une liste des contextes qui ont été activés par la fonction ACTIVATE, par opposition à leur activation dûe au fait qu'ils sont sous-contextes du contexte courant.

ASSUME (*pred1, pred2, ...*) Fonction

vérifie d'abord la redondance et la consistence avec la base de données courante des prédicats spécifiés. S'ils sont consistents et non redondants, ils sont ajoutés à la base de données; s'ils sont inconsistents ou redondants, aucune action n'est entreprise. ASSUME retourne une liste dont les entrées sont les prédicats ajoutés à la base de données et les atomes REDUNDANT ou INCONSISTENT si applicables.

ASSUMESCALAR Variable

défaut: [TRUE] - aide à déterminer si les expressions `exp` pour lesquelles

```
NONSCALARP(exp) is FALSE
```

sont supposées se comporter comme des scalaires pour certaines transformations. Soit `exp` représentant toute non liste/non matrice, et `[1,2,3]` une liste ou matrice quelconque.

```
exp.[1,2,3]; ==>
 [exp,2*exp,3*exp]
```

si ASSUMESCALAR est TRUE ou SCALARP(exp) est TRUE ou CONSTANTP(exp) est TRUE. Si ASSUMESCALAR est TRUE, de telles expressions se comporteront comme des scalaires seulement pour les opérateurs commutatifs, mais pas pour le `"."`. Si ASSUMESCALAR est FALSE, de telles expressions se comporteront comme des non scalaires. Si ASSUMESCALAR est ALL, de telles expressions se comporteront comme des scalaires pour tous les opérateurs listés ci-dessus.

ASSUME_POS Variable

défaut:[FALSE] - En utilisant INTEGRATE, etc. on introduit souvent des paramètres qui sont réels et positifs ou bien les calculs peuvent souvent être construits de façon que ce soit vrai. Le commutateur ASSUME_POS (FALSE par défaut) est tel que s'il est mis à TRUE, MACSYMA va supposer que nos paramètres sont positifs. L'intention est ici de couper court au nombre de questions que MACSYMA a besoin de poser.

Il est clair que l'information ASSUME ou toute information contextuelle présente aura la précédence. L'utilisateur peut contrôler ce qui est considéré comme étant un paramètre dans ce but. Les paramètres par défaut sont ceux qui satisfont SYMBOLP(x) ou SUBVARP(x). L'utilisateur peut changer ceci en positionnant l'option ASSUME_POS_PRED [par défaut FALSE] sur le nom d'une fonction prédicat à un argument. P. ex. si vous voulez que seuls les symboles soient des paramètres, vous pouvez faire ASSUME_POS:TRUE$ ASSUME_POS_PRED:'SYMBOLP$ SIGN(A); -> POS, SIGN(A[1]); -> PNZ.

ASSUME_POS_PRED Variable

défaut:[FALSE] - peut avoir un argument pour contrôler ce qui sera considéré comme étant un paramètre pour les "suppositions" que fera INTEGRATE ... voyez ASSUME et ASSUME_POS .

CONTEXT Variable

défaut: INITIAL. Chaque fois qu'un utilisateur présume un nouveau fait, il est placé dans le contexte nommé comme valeur actuelle de la variable CONTEXT. De même, FORGET fait référence à la valeur actuelle de CONTEXT. Pour changer de contexte, liez simplement CONTEXT au le contexte desiré. Si le contexte spécifié n'existe pas il sera créé par un appel invisible à NEWCONTEXT. Le contexte spécifié par la valeur de CONTEXT est automatiquement activé. Faites DESCRIBE(CONTEXTS); pour une description générale du mécanisme de CONTEXT.

CONTEXTS Variable

défaut: [INITIAL,GLOBAL] - est une liste de contextes qui existent actuellement, y compris le contexte actif courant. Le mécanisme de contexte rend possible pour un utilisateur de lier ensemble et de nommer une portion sélectionnée de sa base de données, appelée un contexte. Une fois cela fait, l'utilisateur peut obtenir de MACSYMA de supposer ou oublier un grand nombre de faits simplement en activant ou désactivant leur contexte. Tout atome symbolique peut être un contexte, et les faits contenus dans ce contexte seront conservés jusqu'à ce que l'utilisateur les détruise, individualement avec FORGET ou tous avec KILL qui détruit le contexte auquel ils appartiennent.

Les contextes sont placés selon une hiérarchie formelle, dont la racine est toujours le contexte GLOBAL, qui contient des informations sur MACSYMA dont ont besoin certaines fonctions. Dans un contexte donné, tous les faits de ce contexte sont "actifs" (ce qui signifie qu'ils sont utilisés en déductions et récupération) comme le sont tous les faits de tout contexte inférieur à ce contexte. Lorsqu'un nouveau MACSYMA est lancdé, l'utilisateur est dans un contexte dit INITIAL, qui a GLOBAL comme sous-contexte.

Les fonctions traitant des contextes sont: FACTS, NEWCONTEXT, SUPCONTEXT, KILLCONTEXT, ACTIVATE, DEACTIVATE, ASSUME, et FORGET.

DEACTIVATE (*cont1, cont2, ...*) Fonction

désactive les contextes conti spécifiés.

FACTS (*item*) Fonction

Si 'item' est le nom d'un contexte alors FACTS retourne une liste des faits de ce contexte. Si aucun argument n'est donné, la fonction donne la liste du contexte courant. Si 'item' n'est pas le nom d'un contexte alors elle retourne une liste de faits connus concernant 'item' dans le contexte courant. Les faits qui sont actifs, mais dans un contexte différent, ne sont pas dans la liste.

FEATURES declaration

MACSYMA a des propriétés prédéfinies qui sont manipulées par la base de données. Elles sont appelées FEATURES. On peut faire DECLARE(N,INTEGER), etc. On peut aussi déclarer ses propres FEATURES en faisant par exemple DECLARE(INCREASING, FEATURE); qui permettra alors de déclarer DECLARE(F, INCREASING);. On peut ensuite vérifier si F est INCREASING avec le prédicat FEATUREP par FEATUREP(F, INCREASING). Il y a une infoliste FEATURES qui est une liste de FEATURES connues. Actuellement, ce sont: INTEGER, NONINTEGER, EVEN, ODD, RATIONAL, IRRATIONAL, REAL, IMAGINARY, COMPLEX, ANALYTIC, INCREASING, DECREASING, ODDFUN, EVENFUN, POSFUN, COMMUTATIVE, LASSOCIATIVE, RASSOCIATIVE, SYMMETRIC, et ANTISYMMETRIC.

Note: les "features" système peuvent être vérifiées avec STATUS(FEATURE, ...) ; voir DESCRIBE(STATUS); ou DESCRIBE(FEATURE); pour les détails.

FORGET (*pred1, pred2, ...*) Fonction

supprime les relations etablies par ASSUME. Les prédicats peuvent être des expressions équivalentes (mais pas nécessairement identiques) à celles présumées précédemment par ASSUME. FORGET(liste) en est aussi une forme correcte.

KILLCONTEXT (*contexte1,contexte2,...,contexten*) Fonction

supprime les contextes spécifiés. Si l'un d'eux est le contexte courant, le nouveau contexte actif deviendra le premier sous-contexte disponible du contexte courant n'ayant pas été supprimé. Si le premier contexte disponible non supprimé est GLOBAL alors INITIAL est utilisé à la place. Si le contexte INITIAL est supprimé, un nouvel INITIAL est créé, qui ne contient pas de fait.

KILLCONTEXT ne permet pas à l'utilisateur de supprimer un contexte qui est actuellement actif, soit parce qu'il est un sous-contexte du context courant, ou par utilisation de la fonction ACTIVATE.

NEWCONTEXT (*nom*) Fonction

crée un nouveau contexte (vide), appelé "nom", qui a GLOBAL comme seul sous-contexte. Le contexte nouvellement créé deviendra le contexte actif courant.

SUPCONTEXT (*nom,contexte*) Fonction

créera un nouveau contexte (appelé nom) dont le sous-contexte est contexte. Si contexte n'est pas spécifié, le contexte courant sera présumé. S'il est spécifié, contexte doit exister.

11 Polynômes

11.1 Introduction aux polynômes

Les polynômes sont stockés dans maxima soit sous Forme Générale soit sous forme d'expressions rationnelles canoniques (Canonical Rational Expressions, CRE). Cette dernière est une forme standard, et est utilisée en interne par des opérations comme factor, ratsimp, etc.

Les CRE constituent une sorte de représentation qui convient spécialement pour le développement des polynômes et des fonctions rationnelles (ainsi qu'aux polynômes et fonctions rationnelles partiellement factorisés lorsque RATFAC[FALSE] est TRUE). Dans cette forme CRE un ordre des variables (de la plus à la moins principale) est supposé pour chaque expression. Les polynômes sont représentés récursivement par une liste consistant en la variable principale suivie d'une suite de paires d'expressions, une pour chaque terme du polynôme. Le premier membre de chaque paire est l'exposant de la variable principale dans ce terme et le second membre est le coefficient de ce terme qui peut être un nombre ou un polynôme d'une autre variable représenté à nouveau sous cette forme. Ainsi la partie principale de la forme CRE de 3*X^2-1 est (X 2 3 0 -1) et celle de 2*X*Y+X-3 est (Y 1 (X 1 2) 0 (X 1 1 0 -3)) en supposant que Y est la variable principale, et (X 1 (Y 1 2 0 1) 0 -3) si X est la variable principale. La "principalité" (main-ness?) est en général déterminée par ordre alphabétique inverse. Les "variables" d'une expression CRE ne sont pas nécessairement atomiques. En fait, toute sous-expression dont l'opérateur principal n'est pas + - * / ou ^ avec une puissance entière sera considérée comme une "variable" de l'expression (sous forme CRE) où il se trouve. Par exemple les variables CRE de l'expression X+SIN(X+1)+2*SQRT(X)+1 sont X, SQRT(X), et SIN(X+1). Si l'utilisateur ne spécifie pas d'ordre des variables à l'aide de la fonction RATVARS, MACSYMA choisira l'ordre alphabétique. En général, CRE représente des expressions rationnelles, c'est à dire, des rapports de polynômes, où numérateur et dénominateur n'ont pas de facteurs communs, et où le dénominateur est positif. La forme interne est essentiellement une paire de polynômes (le numérateur et le dénominateur) précédée de la liste ordonnée des variables. Si une expression est à afficher sous forme CRE ou si elle contient des sous-expressions sous forme CRE, le symbole /R/ suivra l'étiquette de la ligne.

Voyez la fonction RAT de conversion d'une expression sous forme CRE. Une forme CRE étendue est utilisée pour la représentation des séries de Taylor. La notion d'expression rationnelle est étendue à celle des exposants des variables pouvant être des nombres rationnels positifs ou négatifs plutôt que seulement des entiers positifs et les coefficients peuvent eux-mêmes être des expressions rationnelles comme décrit ci-dessus plutôt que juste des polynômes. Ceux-ci sont représentés en interne par une forme récursive de polynôme qui est semblable à une forme CRE et en est une généralisation, mais comporte des informations supplémentaires tel que le degré d'une troncature. Comme avec la forme CRE, le symbole /T/ suit l'étiquette de la ligne de telles expressions.

11.2 Polynômes: définitions

ALGEBRAIC Variable

défaut: [FALSE] - doit être TRUE pour que la simplification d'entiers algébriques ait lieu.

BERLEFACT Variable

défaut: [TRUE] - si FALSE alors l'algorithme de factorisation de Kronecker sera utilisé, sinon ce sera celui de Berlekamp, qui l'est par défaut.

BEZOUT (*p1, p2, var*) Fonction

une alternative à la commande RESULTANT. Elle retourne une matrice. Le DETERMINANT de cette matric est le résultat désiré.

BOTHCOEF (*exp, var*) Fonction

retourne une liste dont le premier membre est le coefficient de var dans exp (comme trouvé par RATCOEF si exp est sous forme CRE, autrement par COEFF) et dont le second membre est la partie restante de exp. C'est à dire, [A,B] où exp=A*var+B.

```
(C1) ISLINEAR(EXP,VAR):=BLOCK([C],
        C:BOTHCOEF(RAT(EXP,VAR),VAR),
        IS(FREEOF(VAR,C) ET C[1]#0))$
(C2) ISLINEAR((R**2-(X-R)**2)/X,X);
(D2)                                TRUE
```

COEFF (*exp, v, n*) Fonction

obtient le coefficient de v**n dans exp. n peut être omis s'il est 1. v peut être un atome, ou une sous-expression complète de exp, p. ex., X, SIN(X), A[I+1], X+Y, etc (dans le dernier cas l'expression (X+Y) devra se trouver dans exp). Il peut être quelquefois nécessaire de développer ou factoriser exp afin de rendre v^n explicite. Ce n'est pas fait automatiquement par COEFF.

```
(C1) COEFF(2*A*TAN(X)+TAN(X)+B=5*TAN(X)+3,TAN(X));
(D1)                           2 A + 1 = 5
(C2) COEFF(Y+X*%E**X+1,X,0);
(D2)                              Y + 1
```

COMBINE (*exp*) Fonction

simplifie la somme exp en plaçant dans un unique terme ceux ayant le même dénominateur.

CONTENT (*p1, var1, ..., varn*) Fonction

retourne une liste dont le premier élément est le plus grand commun diviseur des coefficients des termes du polynôme p1 dans la variable varn (c'est le contenu CONTENT) et dont le second élément est le polynôme p1 divisé par le contenu.

```
(C1) CONTENT(2*X*Y+4*X**2*Y**2,Y);
(D1)            [2*X, 2*X*Y**2+Y].
```

DENOM (*exp*) Fonction

retourne le dénominateur de l'expression rationnelle exp.

DIVIDE (*p1, p2, var1, ..., varn*) Fonction

calcule le quotient et le reste du polynôme p1 divisé par le polynôme p2, dans la variable principale varn du polynôme. Les autres variables sont comme dans la fonction RATVARS. Le résultat est une liste dont le premier élément est le quotient et le second élément le reste.

```
(C1) DIVIDE(X+Y,X-Y,X);
(D1)                          [1, 2 Y]
(C2) DIVIDE(X+Y,X-Y);
(D2)                       [ - 1, 2 X]
```

(Notez que Y est la variable principale dans C2)

ELIMINATE ([*eq1,eq2,...,eqn*],[*v1,v2,...,vk*]) Fonction

élimine les variables des équations (ou des expressions supposées égales à zéro) en prenant les résultants successifs. Elle retourne une liste de n-k expressions dont les k variables v1,...,vk pnt été éliminées. D'abord v1 est éliminé, laissant n-1 expressions, puis v2, etc. Si k=n alors une unique expression dans une liste est retournée indépendante des variables v1,...,vk. Dans ce cas SOLVE est appelée pour résoudre le dernier résultant de la dernière variable. Exemple:

```
(C1) EXP1:2*X^2+Y*X+Z;
                                      2
(D1)                      Z + X Y + 2 X
(C2) EXP2:3*X+5*Y-Z-1;
(D2)                   - Z + 5 Y + 3 X - 1
(C3) EXP3:Z^2+X-Y^2+5;
                         2    2
(D3)                    Z  - Y  + X + 5
(C4) ELIMINATE([EXP3,EXP2,EXP1],[Y,Z]);
             8         7         6          5          4
(D3) [7425 X  - 1170 X  + 1299 X  + 12076 X  + 22887 X
                                3         2
                         - 5154 X  - 1291 X  + 7688 X + 15376]
```

EZGCD (*p1, p2, ...*) Fonction

donne une liste dont le premier élément est le pgcd des polynômes p1,p2,... et dont les éléments restants sont les polynômes divisés par le pgcd. Utilise toujours l'algorithme EZGCD.

FACEXPAND Variable

défaut: [TRUE] - contrôle si les facteurs irréductibles retournés par FACTOR sont sous forme développée (le défaut) ou récursive (CRE normal).

FACTCOMB (*exp*) Fonction

essaie de combiner les coefficients des factorielles de exp avec les factorielles elles-mêmes en convertissant, par exemple, (N+1)*N! en (N+1)!.

Si SUMSPLITFACT[TRUE] est mis à FALSE, MINFACTORIAL sera appliqué après un FACTCOMB.

```
(C1) (N+1)^B*N!^B;
                                          B   B
(D1)                              (N + 1)  N!
(C2) FACTCOMB(%);
```

FACTOR (*exp*) Fonction

factorise l'expression exp, contenant un nombre quelconque de variables ou de fonctions, en facteurs irréductibles sur les entiers.

FACTOR(exp, p) factorise exp sur le champ des entiers avec un élément contigü dont le polynôme minimal est p.

FACTORFLAG[FALSE] si FALSE supprime la factorisation des entiers facteurs d'expression rationnelles.

DONTFACTOR peut être défini comme une liste de variables par rapport auxquelles la factorisation ne doit pas se produire (elle est vide au départ). La factorisation n'aura pas lieu non plus par rapport à toutes les variables qui sont moins importantes (en utilisant le classement des variables défini par une forme CRE) que celles de la liste DONTFACTOR.

SAVEFACTORS[FALSE] si TRUE sauvegarde à l'aide de certaines fonctions les facteurs d'une expression qui est un produit de facteurs, afin d'accélérer les factorisations ultérieurs d'expressions contenant certains de ces mêmes facteurs.

BERLEFACT[TRUE] si FALSE fera utiliser l'algorithme de factorisation de Kronecker, sinon celui de Berlekamp, qui est l'algorithme par défaut.

INTFACLIM[1000] est le plus grand diviseur qui sera essayé pour factoriser un entier "bignum". Si mis à FALSE (c'est le cas lorsque l'utilisateur appelle FACTOR explicitement), ou si l'entier est un "fixnum" (i.e. est contenu dans un seul mot machine), la factorisation complète de l'entier sera tentée. Le paramétrage utilisateur de INTFACLIM est utilisé pour les appels internes de FACTOR. Ainsi, INTFACLIM peut être redéfini pour empêcher MACSYMA de prendre un temps excessivement long pour factoriser de grands entiers.

NEWFAC[FALSE] peut être mis à TRUE pour utiliser les nouvelles routines de factorisation.

Faites EXAMPLE(FACTOR); pour voir des exemples.

FACTORFLAG Variable

défaut: [FALSE] - si FALSE supprime la factorisation des facteurs entiers des expressions rationnelles.

FACTOROUT (*exp,var1,var2,...*) Fonction

réarrange la somme exp en une somme de termes de la forme f(var1,var2,...)*g où g est un produit d'expressions ne contenant pas de vari et où f est factorisé.

FACTORSUM (*exp*) Fonction

essaie de grouper les termes des facteurs de exp qui sont des sommes en groupes de termes tels que leur somme est "factorisable". Il peut retrouver le résultat de EXPAND((X+Y)^2+(Z+W)^2) mais il ne peut pas retrouver EXPAND((X+1)^2+(X+Y)^2) car les termes ont des variables en commun.

```
(C1) (X+1)*((U+V)^2+A*(W+Z)^2),EXPAND;
      2      2                                2      2
(D1) A X Z  + A Z  + 2 A W X Z + 2 A W Z + A W  X + V  X
                     2        2    2           2
        + 2 U V X + U  X + A W  + V  + 2 U V + U
(C2) FACTORSUM(%);
                                    2          2
(D2)                (X + 1) (A (Z + W)  + (V + U) )
```

FASTTIMES (*p1, p2*) Fonction

multiplie les polynômes p1 et p2 en utilisant un algorithme spécial pour la multiplication de polynômes. Ils devront être multivariables, denses, et à peu près de la même taille. La multiplication classique est d'ordre N*M où N et M sont les degrés. FASTTIMES est d'ordre MAX(N,M)**1.585.

FULLRATSIMP (*exp*) Fonction

Lorsque des expressions non rationnelles sont impliquées, un seul appel à RATSIMP suivi comme il est usuel par une simplification non rationnelle ("générale") peut ne pas être suffisant pour renvoyer un résultat simplifié. Quelquefois, plus d'un tel appel peut être nécessaire. La commande FULLRATSIMP rend ce processus plus pratique. FULLRATSIMP applique RATSIMP sans arrêt, suivie d'une simplification en une expression non rationnelle jusqu'à ce qu'aucune modification ne se produise. Par exemple, considérez l'expression EXP: (X^(A/2)+1)^2*(X^(A/2)-1)^2/(X^A-1). RATSIMP(EXP); donne (X^(2*A)-2*X^A+1)/(X^A-1) . FULLRATSIMP(EXP); donne X^A-1. Le problème peut être reconnu en faisant RAT(EXP); qui donne ((X^(A/2))^4-2*(X^(A/2))^2+1)/(X^A-1).

FULLRATSIMP(exp,var1,...,varn) prend un ou plusieurs arguments, comme RATSIMP et RAT.

FULLRATSUBST (*a,b,c*) Fonction

est comme RATSUBST sauf qu'elle s'appelle elle-même récursivement sur son résultat jusqu'à ce que ce résultat ne change plus. Cette fonction est utile lorsque l'expression

de remplacement et l'expression remplacée ont en commun une ou plusieurs variables. FULLRATSUBST accepte aussi ses arguments dans le format de LRATSUBST. C'est à dire que le premier argument peut être une unique équation de substitution ou une liste de telles équations, alors que le second argument est l'expression à traiter. Il y a une démonstration disponible avec DEMO("lrats.dem"); .

GCD (*p1, p2, var1, ...*) Fonction

calcule le plus grand diviseur commun de p1 et p2. Le drapeau GCD[SPMOD] détermine quel algorithme sera employé. Mettre GCD à EZ, EEZ, SUBRES, RED, ou SPMOD sélectionne respectivement l'algorithme EZGCD, New EEZ GCD, sous-résultant PRS, réduit, ou modulaire. Si GCD:FALSE alors GCD(p1,p2,var) retournera toujours 1 pour toute var. Beaucoup de fonctions (p. ex. RATSIMP, FACTOR, etc.) calculent implicitement le pgcd. Pour les polynômes homogènes il est recommandé d'utiliser GCD:SUBRES. Pour prendre le pgcd lorsqu'une expression algébrique est présente, p. ex. GCD(X^2-2*SQRT(2)*X+2,X-SQRT(2)); , ALGEBRAIC doit être TRUE et GCD ne pas être EZ. SUBRES est un nouvel algorithme, et ceux qui ont utilisé le paramètre RED devront probablement le changer en SUBRES. Le drapeau GCD, par défaut: [SPMOD], si FALSE empêchera aussi de prendre le pgcd lorsque les expressions sont converties au format CRE. Cela accélèrera quelquefois le calcul si le pgcd n'est pas requis.

GCFACTOR (*n*) Fonction

factorise l'entier gaussien n sur les gaussiens, i.e. les nombres de la forme a + b i où a et b sont des entiers rationnels (i.e. des entiers ordinaires). Les facteurs sont normalisés en faisant a et b non négatifs.

GFACTOR (*exp*) Fonction

factorise le polynôme exp sur les entiers gaussiens (i. e. pourvus de SQRT(-1) = %I). C'est comme FACTOR(exp,A**2+1) où A est %I.

```
(C1)  GFACTOR(X**4-1);
(D1)         (X - 1) (X + 1) (X + %I) (X - %I)
```

GFACTORSUM (*exp*) Fonction

est semblable à FACTORSUM mais applique GFACTOR au lieu de FACTOR.

HIPOW (*exp, v*) Fonction

le plus grand exposant explicite de v dans exp. Il peut être parfois nécessaire de développer exp car ce n'est pas fait automatiquement par HIPOW. Ainsi HIPOW(Y**3*X**2+X*Y**4,X) est 2.

INTFACLIM Variable

par défaut: [1000] - est le plus grand diviseur qui sera essayé lors de la factorisation d'un entier "bignum". Si mis à FALSE (c'est le cas lorsque l'utilisateur appelle FACTOR explicitement), ou si l'entier est un "fixnum" (i.e. est contenu dans un

seul mot machine), la factorisation complète de l'entier sera tentée. Le paramétrage utilisateur de INTFACLIM est utilisé pour les appels internes à FACTOR. Ainsi, INTFACLIM peut être ré-initialisé pour empêcher MACSYMA de prendre un temps anormalement long lors de la factorisation de grands entiers.

KEEPFLOAT Variable

par défaut: [FALSE] - si mis à TRUE empêchera les nombres en virgule flottante d'être rationalisés lorsque les expressions qui les contiennent sont converties au format CRE.

LRATSUBST (*list,exp*) Fonction

est semblable à SUBST(liste_d'équations,exp) sauf qu'elle utilise RATSUBST au lieu de SUBST. Le premier argument de LRATSUBST doit être une équation ou une liste d'équations au format identique à celui accepté par SUBST (voir DESCRIBE(SUBST);). Les substitutions sont faites dans l'ordre donné par la liste d'équations, c.-à-d., de gauche à droite. Faire DEMO("lrats.dem"); pour en avoir une démonstration.

MODULUS Variable

par défaut: [FALSE] - si sa valeur est un premier positif p, toute l'arithmétique des routines de fonctions rationnelles sera faite modulo p. C'est à dire que tous les entiers seront réduits à moins de p/2 en valeur absolue (si p=2 alors tous les entiers sont réduits à 1 ou 0). C'est le système dit à modulo "équilibré", p. ex. N MOD 5 = -2, -1, 0, 1, ou 2.

Avertissement: si EXP est déjà sous forme CRE lorsque vous ré-initialisez MODULUS, vous pouvez alors avoir besoin de ré-rationnaliser EXP, p. ex. EXP:RAT(RATDISREP(EXP)), afin d'obtenir des résultats corrects (si MODULUS est un entier positif non premier, ce paramètre sera accepté, mais un avertissement sera donné).

NEWFAC Variable

par défaut: [FALSE] - si TRUE alors FACTOR utilisera les nouvelles routines de factorisation.

NUM (*exp*) Fonction

renvoie le numérateur, exp1, de l'expression rationnelle exp = exp1/exp2.

QUOTIENT (*p1, p2, var1, ...*) Fonction

calcule le quotient du polynôme p1 divisé par le polynôme p2.

RAT (*exp, v1, ..., vn*) Fonction

convertit exp au format CRE en développant et combinant tous les termes sur un dénominateur commun et en supprimant le plus grand diviseur commun du numérateur et du dénominateur et en convertissant de plus les nombres en virgule flottante en nombres rationnels avec une tolérance de RATEPSILON[2.0E-8]. Les variables sont ordonnées selon les v1,...,vn comme dans RATVARS, si elles sont spécifiées. RAT ne simplifie pas en général les fonctions autres que + , - , * , / , et l'exponentiation

à une puissance entière alors que RATSIMP traite ces cas. Notez que les atomes (nombres et noms) au format CRE ne sont pas les mêmes que sous la forme générale. Ainsi RAT(X)- X résulte en RAT(0) qui a une représentation interne différente de 0. RATFAC[FALSE] si TRUE invoque une forme partiellement factorisée des expressions rationnelles CRE. Durant les opérations rationnelles l'expression est maintenue aussi complètement factorisée que possible sans faire appel au package factor. Ceci devrait toujours économiser de l'espace et peut gagner du temps lors de certains calculs. Les numérateur et dénominateur sont encore rendus premiers relatifs (e.g. RAT((X^2 -1)^4/(X+1)^2); donne (X-1)^4*(X+1)^2), mais les facteurs dans chaque partie peuvent ne pas être premiers relatifs.

RATPRINT[TRUE] si FALSE supprime l'affichage du message informant l'utilisateur de la conversion des nombres en virgule flottante en nombres rationnels.

KEEPFLOAT[FALSE] si TRUE empêche les nombres en virgule flottante d'être convertis en nombres rationnels (voir aussi les fonctions RATEXPAND et RATSIMP).

```
(C1) ((X-2*Y)**4/(X**2-4*Y**2)**2+1)*(Y+A)*(2*Y+X)
       /(4*Y**2+X**2);
                                              4
                                     (X - 2 Y)
                (Y + A) (2 Y + X) (------------ + 1)
                                       2      2 2
                                     (X  - 4 Y )
(D1)            -----------------------------------
                                        2    2
                                   4 Y  + X
(C2) RAT(%,Y,A,X);
                                    2 A + 2 Y
(D2)/R/                             ---------
                                    X + 2 Y
```

RATALGDENOM Variable

par défaut: [TRUE] - si TRUE permet la rationnalisation des dénominateurs radicaux wrt. Il faut utiliser la forme CRE dans le mode algébrique.

RATCOEF (*exp, v, n*) Fonction

retourne le coefficient, C, de l'expression v**n de exp. n peut être omis s'il vaut 1. C sera indépendant des variables en v (sauf peut-être dans un sens non rationnel). Si aucun coefficient de ce type n'existe, zéro sera retourné. RATCOEF développe et simplifie rationnellement son premier argument et peut ainsi donner des réponses différentes de celles de COEFF qui est purement syntaxique. Par exemple RATCOEF((X+1)/Y+X,X) retourne (Y+1)/Y alors que COEFF retourne 1. RATCOEF(exp,v,0) voit exp comme une somme, donne une somme des termes qui ne contiennent pas v. Par conséquent, si v se trouve avec des puissances négatives, RATCOEF ne devra pas être utilisée. Comme exp est rationnellement simplifiée avant d'être examinée, les coefficients peuvent ne pas apparaître entièrement comme on s'y attendait.

```
(C1) S:A*X+B*X+5$
(C2) RATCOEF(S,A+B);
(D2)                X
```

RATDENOM (*exp*) Fonction

obtient le dénominateur de l'expression rationnelle exp. Si exp est sous forme générale alors la fonction DENOM devra être utilisée à la place, à moins que l'on veuille obtenir un résultat CRE.

RATDENOMDIVIDE Variable

par défaut: [TRUE] - si FALSE arrêtera le (découpage splitting out ??) des termes du numérateur des expressions RATEXPANDées qui pourrait se produire.

RATDIFF (*exp, var*) Fonction

différencie l'expression rationnelle exp (qui doit être un rapport de polynômes ou un polynôme de la variable var) par rapport à var. Pour les expressions rationnelles ceci est bien plus rapide que DIFF. Le résultat est laissé sous forme CRE. Cependant RATDIFF ne devrait pas être utilisée sur des formes CRE factorisées ; utilisez plutôt DIFF pour de telles expressions.

```
(C1) (4*X**3+10*X-11)/(X**5+5);
                                        3
                                   4 X  + 10 X - 11
(D1)                               ----------------
                                         5
                                        X
(C2) MODULUS:3$
(C3) MOD(D1);
                                    2
                                   X  + X - 1
(D3)                        --------------------
                             4    3    2
                            X  + X  + X  + X + 1
(C4) RATDIFF(D1,X);
                            5    4    3
                           X  - X  - X  + X - 1
(D4)                 -----------------------------
                      8    7    5    4    3
                     X  - X  + X  - X  + X  - X + 1
```

RATDISREP (*exp*) Fonction

change son argument sous forme CRE en forme générale. C'est parfois utile si l'on veut arrêter la "contagion", ou utiliser des fonctions rationnelles dans des contextes non rationnels. La plupart des fonctions CRE travailleront sur des expressions CRE ou non CRE, mais les réponses peuvent prendre différentes formes. Si un argument

non CRE est passé à RATDISREP, il retourne son argument inchangé. Voyez aussi TOTALDISREP.

RATEPSILON Variable

défaut: [2.0E-8] - la tolérance utilisée lors de la conversion en nombres rationnels des nombres en virgule flottante.

RATEXPAND (*exp*) Fonction

développe exp en multipliant les produits de sommes et les sommes exponentielles, en combinant les fractions sur un dénominateur commun, en supprimant le plus grand commun diviseur du numérateur et du dénominateur, puis en découpant le numérateur (si c'est une somme) en ses termes respectifs divisés par le dénominateur. Ceci s'effectue en convertissant exp sous forme CRE puis à nouveau sous forme générale.

Le commutateur RATEXPAND, par défaut: [FALSE], si TRUE provoquera le développement complet des expressions CRE lors de leur reconversion sous forme générale ou affichée, alors que si FALSE elles seront mises sous forme récursive (voir RATSIMP).

RATDENOMDIVIDE[TRUE] - si FALSE arrêtera le découpage des termes du numérateur des expressions RATEXPANDées qui pourrait se produire.

KEEPFLOAT[FALSE] si mis à TRUE empêchera les nombres en virgule flottante d'être rationalisés lorsque les expressions qui les contiennent sont converties en forme CRE.

```
(C1) RATEXPAND((2*X-3*Y)**3);
                3         2       2        3
(D1)      - 27 Y  + 54 X Y  - 36 X  Y + 8 X
(C2) (X-1)/(X+1)**2+1/(X-1);
                              X - 1       1
(D2)                         -------- + -----
                                    2   X - 1
                             (X + 1)
(C3) EXPAND(D2);
                         X              1           1
(D3)                -------------- - ------------ + -----
                     2              2             X - 1
                    X  + 2 X + 1   X  + 2 X + 1
(C4) RATEXPAND(D2);
                             2
                          2 X                2
(D4)                 ---------------- + ---------------
                      3    2            3    2
                     X  + X  - X - 1   X  + X  - X - 1
```

RATFAC Variable

par défaut: [FALSE] - si TRUE invoque une forme partiellement factorisée des expression CRE rationnelles. Durant les opérations rationnelles l'expression est maintenue

aussi complètement factorisée que possible sans faire appel au package factor. Ceci devrait toujours économiser de l'espace et peut gagner du temps lors de certains calculs. Les numérateur et dénominateur sont encore rendus premiers relatifs (e.g. RAT((X^2 -1)^4/(X+1)^2); donne (X-1)^4*(X+1)^2), mais les facteurs dans chaque partie peuvent ne pas être premiers relatifs.

Dans le package CTENSR (Component Tensor Manipulation), si RATFAC est TRUE, il provoque la factorisation automatique des tenseurs de Ricci, d'Einstein, de Riemann, et de Weyl et de la Courbure scalaire. ** Ce ne devrait être ainsi positionné que dans les cas où on sait que les composants tensoriels ont peu de termes **.

Note: lesprocédés RATFAC et RATWEIGHT sont incompatibles et ne doivent pas être utilisés en même temps.

RATNUMER (*exp*) Fonction

obtient le numérateur de l'expression rationnelle exp. Si exp est sous forme générale la fonction NUM devra être utilisée à la place, à moins que l'on veuille obtenir un résultat CRE.

RATNUMP (*exp*) Fonction

est TRUE si exp est un nombre rationnel (ou entier), sinon FALSE.

RATP (*exp*) Fonction

est TRUE si exp est sous forme CRE ou CRE étendue, sinon FALSE.

RATPRINT Variable

par défaut: [TRUE] - si FALSE supprime l'affichage du message informant l'utilisateur de la conversion des nombres en virgule flottante en nombres rationnels.

RATSIMP (*exp*) Fonction

semblable à RATEXPAND, simplifie rationnellement l'expression exp et toutes ses sous-expressions y compris les arguments des fonctions non rationnelles. Le résultat est retourné comme quotient de deux polynômes sous forme récursive, i.e. les coefficients de la variable principale sont des polynômes en les autres variables. Les variables peuvent, comme dans RATEXPAND, inclurent des fonctions non rationnelles (e.g. SIN(X**2+1)) mais avec RATSIMP, les arguments de ces fonctions non rationnelles sont rationnellement simplifiés. Notez que RATSIMP est affecté par certaines des variables qui affectent RATEXPAND.

RATSIMP(exp,v1,v2,...,vn) - permet la simplification rationnelle avec la spécification de variable ordonnée comme avec RATVARS.

RATSIMPEXPONS[FALSE] - si TRUE provoque la simplification rationnelle automatique (par RATSIMP) des exposants des expressions pendant la simplification.

```
(C1) SIN(X/(X^2+X))=%E^((LOG(X)+1)**2-LOG(X)**2);
                                                2          2
                       X                (LOG(X) + 1)  - LOG (X)
(D1)          SIN(------) = %E
                     2
```

```
                  X  + X
(C2) RATSIMP(%);
                                      1          2
(D2)                          SIN(-----) = %E X
                                    X + 1
(C3) ((X-1)**(3/2)-(X+1)*SQRT(X-1))/SQRT((X-1)*(X+1));
                        3/2
                (X - 1)    - SQRT(X - 1) (X + 1)
(D3)            --------------------------------
                    SQRT(X - 1) SQRT(X + 1)
(C4) RATSIMP(%);
                                    2
(D4)                        - -----------
                              SQRT(X + 1)
(C5)  X**(A+1/A),RATSIMPEXPONS:TRUE;
                     2
                    A  + 1
                    ------
                      A
(D5)               X
```

RATSIMPEXPONS Variable

par défaut: [FALSE] - si TRUE provoque la simplification rationnelle automatique (par RATSIMP) des exposants des expressions pendant la simplification.

RATSUBST (*a, b, c*) Fonction

remplace b par a dans c. b peut être une somme, un produit, une puissance, etc. RATSUBST a quelques connaisances du sens des expressions alors que SUBST fait une substitution purement syntaxique. Ainsi SUBST(A,X+Y,X+Y+Z) retourne X+Y+Z alors que RATSUBST retournera Z+A.

RADSUBSTFLAG[FALSE] - si TRUE permet à RATSUBST de faire des substitutions telle que U pour SQRT(X) en X. Faites EXAMPLE(RATSUBST); pour des exemples.

RATVARS (*var1, var2, ..., varn*) Fonction

place ses n arguments dans une liste où la variable varn la plus à droite sera la variable principale des futures expression rationnelles dans lesquelles elle se trouve, et les autres variables suivront en séquence. Si une variable manque dans la liste RATVARS, il lui sera donné une priorité inférieure à celle de la variable la plus à gauche var1. Les arguments de RATVARS peuvent être soit des variables soit des fonctions non rationnelles (e.g. SIN(X)). La variable RATVARS est une liste des arguments qui ont été donnés à cette fonction.

RATWEIGHT (*v1, w1, ..., vn, wn*) Fonction

affecte un poids de wi à la variable vi. Ceci provoque le remplacement d'un terme par 0 si son poids dépasse la valeur de la variable RATWTLVL [FALSE par défaut ce qui sig-

nifie pas de troncature]. Le poids d'un terme est la somme des produits du poids d'une variable dans le terme par sa puissance. Ainsi le poids de 3*v1**2*v2 est 2*w1+w2. Cette troncature se produit seulement pour la multiplication ou l'exponentiation des expressions sous forme CRE.

```
(C5) RATWEIGHT(A,1,B,1);
(D5)                          [[B, 1], [A, 1]]
(C6) EXP1:RAT(A+B+1)$
(C7) %**2;
                       2                    2
(D7)/R/               B  + (2 A + 2) B + A  + 2 A + 1
(C8) RATWTLVL:1$
(C9) EXP1**2;
(D9)/R/                        2 B + 2 A + 1
```

Note: les procédés RATFAC et RATWEIGHT sont incompatibles et ne doivent pas être utilisés en même temps.

RATWEIGHTS Variable

- une liste d'affectations de poids (définie par la fonction RATWEIGHT), RATWEIGHTS; ou RATWEIGHT(); vous montrera la liste.

```
KILL(...,RATWEIGHTS)
```

et

```
SAVE(...,RATWEIGHTS);
```

fonctionnent toutes les deux.

RATWEYL Variable

par défaut: [] - l'un des commutateurs contrôlant la simplification des composants du tenseur conformal de Weyl; si TRUE, alors les composants seront rationnellement simplifiés; si FACRAT est TRUE alors les résultats seront aussi factorisés.

RATWTLVL Variable

par défaut: [FALSE] - utilisé avec la fonction RATWEIGHT pour contrôler la troncature d'expressions rationnelles (forme CRE). Pour la valeur par défaut FALSE, aucune troncature ne se produit.

REMAINDER (*p1, p2, var1, ...*) Fonction

calcule le reste de la division du polynôme p1 par le polynôme p2.

RESULTANT (*p1, p2, var*) Fonction

calcule le résultant de deux polynômes p1 et p2, par élimination de la variable var. Le résultant est un déterminant des coefficients de var en p1 et p2 qui égale zéro si et seulement si p1 et p2 n'ont pas de facteurs non constants en commun. Si p1 ou p2 peut être factorisé, il peut être préférable d'appeler FACTOR avant RESULTANT.

RESULTANT[SUBRES] - contrôle quel algorithme sera utilisé pour calculer le résultant. SUBRES pour le sous-résultant prs [le défaut], MOD pour l'algorithme du

résultant modulaire , et RED pour prs réduit. Pour la plupart des problèmes SUBRES devrait être le meilleur. Sur certains problèmes à grand degré univariate ou bivariate MOD peut être mieux. Une autre alternative est la commande BEZOUT qui prend les mêmes arguments que RESULTANT et retourne une matrice. Le DETERMINANT de cette matrice est le résultant recherché.

SAVEFACTORS Variable

par défaut: [FALSE] - si TRUE les facteurs d'une expression qui est un produit de facteurs sont conservés par certaines fonctions afin d'accélérer les factorisations ultérieures des expressions contenant certains de ces mêmes facteurs.

SQFR (*exp*) Fonction

est semblable à FACTOR sauf que les facteurs polynômiaux sont "square-free", c.-à-d. qu'ils n'ont que des facteurs de degré un. Cet algorithme, qui est aussi utilisé par la première étape de FACTOR, utilise le fait qu'un polynôme a en commun avec sa nième dérivée tous ses facteurs de degré > n. Ainsi en prenant les pgcds du polynôme des dérivées par rapport à chaque variable du polynôme, tous les facteurs de degrée > 1 peuvent être trouvés.

```
(C1) SQFR(4*X**4+4*X**3-3*X**2-4*X-1);
                                 2              2
(D1)                        (X  - 1) (2 X + 1)
```

TELLRAT (*poly*) Fonction

ajoute à l'anneau des entiers algébriques connu de MACSYMA, l'élément qui est solution du polynôme à coefficients entiers. MACSYMA connaît initialement %I et toutes les racines des entiers. TELLRAT(X); signifie substituer 0 à X dans les fonctions rationnelles. Il y a une commande UNTELLRAT qui prend les noyaux et enlève les propriétés TELLRAT. Lorsque TELLRAT est appliqué à un polynôme multivariable, p. ex. TELLRAT(X^2-Y^2);, il y a une ambiguïté, à savoir faut-il substituer Y^2 à X^2 ou vice versa. Le système choisira un ordre particulier, mais si l'utilisateur veut spécifier lequel, alors par exemple TELLRAT(Y^2=X^2); propose de remplacer Y^2 by X^2.

TELLRAT et UNTELLRAT peuvent prendre tous deux un nombre quelconque d'arguments, et TELLRAT(); retourne une liste des substitutions courantes.

Note: lorsque vous appliquez TELLRAT à des polynômes réductibles, prenez garde de ne pas rationaliser un dénominateur avec un diviseur zéro. P. ex., TELLRAT(W^3-1)$ ALGEBRAIC:TRUE$ RAT(1/(W^2-W)); affichera "quotient par zéro". Cette erreur peut être évitée en faisant RATALGDENOM:FALSE$.

ALGEBRAIC[FALSE] doit être mis à TRUE afin d'effectuer la simplification d'entiers algébriques. Faites EXAMPLE(TELLRAT); pour les exemples.

TOTALDISREP (*exp*) Fonction

convertit toutes les sous-expressions de exp de CRE en forme générale. Si exp est elle-même sous forme CRE alors ceci est identique à RATDISREP mais dans le cas contraire RATDISREP retournerait exp inchangée alors que TOTALDISREP ("totally

disrep" it??). Utile pour ("ratdisrepping"??) des expressions, p.ex., des équations, des listes, des matrices, etc. qui ont certaines sous-expressions au format CRE.

UNTELLRAT (x) Fonction

prend les noyaux et enlève les propriétés TELLRAT.

12 Constantes

12.1 Définitions pour les constantes

E Variable
- la base des logarithmes naturels, e, est représentée dans MACSYMA par %E.

FALSE Variable
- la constante booléenne, faux (NIL en LISP).

MINF Variable
- le réel moins infini.

PI Variable
- "pi" est représenté dans MACSYMA par %PI.

TRUE Variable
- la constante booléenne, vrai (T en LISP).

13 Logarithmes

13.1 Définitions pour les logarithmes

LOG (*X*) Fonction

le logarithme naturel.

LOGEXPAND[TRUE] - change LOG(A^B) en B*LOG(A). Si mis à ALL, LOG(A*B) sera aussi simplifié en LOG(A)+LOG(B). Si mis à SUPER, LOG(A/B) sera aussi simplifié en LOG(A)-LOG(B) pour les nombres rationnels a/b, a#1. (LOG(1/B), pour B entier, est toujours simplifié). Si mis à FALSE, aucune de ces simplifications n'aura lieu.

LOGSIMP[TRUE] - si FALSE aucune simplification de %E en une puissance contenant LOG n'est faite.

LOGNUMER[FALSE] - si TRUE les arguments négatifs en virgule flottante de LOG seront toujours convertis en leur valeur absolue avant de prendre le log. Si NUMER est aussi TRUE, les arguments entiers négatifs de LOG seront aussi convertis en leur valeur absolue.

LOGNEGINT[FALSE] - si TRUE implante la règle LOG(-n) -> LOG(n)+%i*%pi pour n entier positif.

%E_TO_NUMLOG[FALSE] - si TRUE, pour "r" un nombre rationnel, et "x" une expression, %E^(r*LOG(x)) sera simplifiée en x^r . Il faut noter que la commande RADCAN fait aussi cette transformation, et des transformations plus compliquées de cette sorte également. La commande LOGCONTRACT "contracte" les expressions contenant LOG.

LOGABS Variable

par défaut: [FALSE] - lorsque des logs sont générés lors d'intégration indéfinie, p. ex. INTEGRATE(1/X,X), la réponse est donnée en termes de LOG(ABS(...)) si LOGABS est TRUE, mais en termes de LOG(...) si LOGABS est FALSE. Pour l'intégration définie, le paramètre LOGABS:TRUE est utilisé, car ici "l'évaluation" de l'intégrale indéfinie aux points limites est souvent nécessaire.

LOGARC Variable

par défaut: [FALSE] - si TRUE provoque la conversion des fonctions circulaires inverses et hyperboliques en leur forme logarithmique.

LOGARC(exp) provoque cette conversion pour une expression particulière sans définir le commutateur ou avoir à ré-évaluer l'expression avec EV.

LOGCONCOEFFP Variable

par défaut:[FALSE] - contrôle quels coefficients seront contractés avec LOGCONTRACT. Peut être mis au nom d'une fonction prédicat d'un argument. Par exemple, si vous aimez générer des SQRT, vous pouvez faire LOGCONCOEFFP:'LOGCONFUN$ LOGCONFUN(M):=FEATUREP(M,INTEGER) OR RATNUMP(M)$. Alors LOGCONTRACT(1/2*LOG(X)); donnera LOG(SQRT(X)).

LOGCONTRACT (*exp*) Fonction

parcourt récursivement une exp, en transformant les sous-expressions de la forme a1*LOG(b1) + a2*LOG(b2) + c en LOG(RATSIMP(b1^a1 * b2^a2)) + c

```
(C1) 2*(A*LOG(X) + 2*A*LOG(Y))$
(C2) LOGCONTRACT(%);
                                              2  4
(D3)                                   A LOG(X  Y )
```

Si vous faites DECLARE(N,INTEGER); alors LOGCONTRACT(2*A*N*LOG(X)); donne A*LOG(X^(2*N)). Les coefficients "contractés" de cette manière sont ceux tels qu'ici le 2 et le N qui satisfont FEATUREP(coeff,INTEGER). L'utilisateur peut contrôler quels coefficients seront contractés en mettant l'option LOGCONCOEFFP[FALSE] au nom d'une fonction prédicat d'un argument. Par exemple, si vous aimez générer des SQRT, vous pouvez faire LOGCONCOEFFP:'LOGCONFUN$ LOGCONFUN(M):=FEATUREP(M,INTEGER) OR RATNUMP(M)$. Alors LOGCONTRACT(1/2*LOG(X)); donnera LOG(SQRT(X)).

LOGEXPAND Variable

par défaut: [TRUE] - transforme LOG(A^B) en B*LOG(A). Si mis à ALL, LOG(A*B) sera aussi simplifié en LOG(A)+LOG(B). Si mis à SUPER, LOG(A/B) sera aussi simplifié en LOG(A)-LOG(B) pour les nombres rationnels a/b, a#1 (LOG(1/B), pour B entier, est toujours simplifié). Si mis à FALSE, aucune de ces simplifications n'aura lieu.

LOGNEGINT Variable

par défaut: [FALSE] - si TRUE implante la règle LOG(-n) -> LOG(n)+%i*%pi pour n entier positif.

LOGNUMER Variable

par défaut: [FALSE] - si TRUE les arguments négatifs en virgule flottante de LOG seront toujours convertis en leur valeur absolue avant de prendre le log. Si NUMER est aussi TRUE, les arguments entiers négatifs de LOG seront aussi convertis en leur valeur absolue.

LOGSIMP Variable

par défaut: [TRUE] - si FALSE aucune simplification de %E en une puissance contenant LOG n'est effectuée.

PLOG (*X*) Fonction

la branche principale du logarithme naturel à valeur complexe avec -%PI < CARG(X) <= +%PI .

POLARFORM (*exp*) Fonction

retourne R*%E^(%I*THETA) où R et THETA sont des réels purs.

14 Trigonométrie

14.1 Introduction à la trigonométrie

MACSYMA contient beaucoup de définitions de fonctions trigonométriques. Toutes les identités trigonométriques ne sont pas programmées, mais l'utilisateur peut en ajouter beaucoup à l'aide des possibilités (pattern matching ?? de filtrage) du système. Les fonctions trigonométriques définies dans MACSYMA sont : ACOS, ACOSH, ACOT, ACOTH, ACSC, ACSCH, ASEC, ASECH, ASIN, ASINH, ATAN, ATANH, COS, COSH, COT, COTH, CSC, CSCH, SEC, SECH, SIN, SINH, TAN, et TANH. Nombre de commandes sont spécialement prévues pour traiter ces fonctions, voyez TRIGEXPAND, TRIGREDUCE, et le commutateur TRIGSIGN. Deux packages SHARE étendent les règles de simplification construites dans MACSYMA, NTRIG et ATRIG1. Faites DESCRIBE(cmd) pour les détails.

14.2 Définitions pour la trigonométrie

ACOS Fonction
- Arc cosinus

ACOSH Fonction
- Arc cos hyperbolique

ACOT Fonction
- Arc cotangente

ACOTH Fonction
- Arc cotangente hyperbolique

ACSC Fonction
- Arc cosécante

ACSCH Fonction
- Arc cosécante hyperbolique

ASEC Fonction
- Arc sécante

ASECH Fonction
- Arc sécante hyperbolique

ASIN Fonction
- Arc sinus

ASINH Fonction
- Arc sinus hyperbolique

ATAN Fonction
- Arc tangente

ATAN2 (*Y*,*X*) Fonction
renvoie la valeur de ATAN(Y/X) dans l'intervalle -%PI à %PI.

ATANH Fonction
- Arc tangente hyperbolique

ATRIG1 Fonction
- SHARE1;ATRIG1 FASL contient plusieurs règles additionnelles de simplification pour les fonctions trigonométriques inverses. Avec les règles déjà connues de Macsyma, les angles suivants sont entièrement implantés : 0, %PI/6, %PI/4, %PI/3, et %PI/2. Les angles correspondants des trois autres quadrants sont aussi disponibles. Faites LOAD(ATRIG1); pour les utiliser.

COS Fonction
- Cosinus

COSH Fonction
- Cosinus hyperbolique

COT Fonction
- Cotangente

COTH Fonction
- Cotangente hyperbolique

CSC Fonction
- Cosécante

CSCH Fonction
- Cosécante hyperbolique

HALFANGLES Variable
par défaut: [FALSE] - si TRUE les demis-angles sont simplifiés.

SEC Fonction
- Sécante

SECH Fonction
- Sécante hyperbolique

SIN Fonction
- Sinus

SINH Fonction

- Sinus hyperbolique

TAN Fonction

- Tangente

TANH Fonction

- Tangente hyperbolique

TRIGEXPAND (*exp*) Fonction

développe les fonctions trigonométriques et hyperboliques des sommes des angles et des angles multiples se trouvant dans exp. Les meilleurs résultats sont obtenus si exp est développée. Pour améliorer le contrôle utilisateur de la simplification, cette fonction développe un seul niveau à la fois, en développant les sommes d'angles et d'angles multiples. Pour avoir le développement complet immédiat des sinus et cosinus, mettez le commutateur TRIGEXPAND:TRUE.

TRIGEXPAND par défaut: [FALSE] - si TRUE provoque le développement de toutes les expressions contenant des SIN et des COS se produisant subséquemment.

HALFANGLES[FALSE] - si TRUE provoque la simplification des demis-angles.

TRIGEXPANDPLUS[TRUE] - contrôle la règle de la "somme" pour TRIGEXPAND, le développement des sommes (p. ex. SIN(X+Y)) n'aura lieu que si TRIGEXPANDPLUS est TRUE.

TRIGEXPANDTIMES[TRUE] - contrôle la règle du "produit" pour TRIGEXPAND, le développement de produits (p. ex. SIN(2*X)) n'aura lieu que si TRIGEXPANDTIMES est TRUE.

```
(C1) X+SIN(3*X)/SIN(X),TRIGEXPAND=TRUE,EXPAND;
                                2           2
(D1)                     - SIN (X) + 3 COS (X) + X
(C2) TRIGEXPAND(SIN(10*X+Y));
(D2)               COS(10 X) SIN(Y) + SIN(10 X) COS(Y)
```

TRIGEXPANDPLUS Variable

par défaut: [TRUE] - contrôle la règle de la "somme" pour TRIGEXPAND. Ainsi, lorsque la commande TRIGEXPAND est utilisée ou que le commutateur TRIGEXPAND est mis à TRUE, le développement de sommes (p. ex. SIN(X+Y)) n'aura lieu que si TRIGEXPANDPLUS est TRUE.

TRIGEXPANDTIMES Variable

par défaut: [TRUE] - contrôle la règle du "produit" pour TRIGEXPAND. Ainsi, lorsque la commande TRIGEXPAND est utilisée ou que le commutateur TRIGEXPAND est mis à TRUE, le développement de produits (p. ex. SIN(2*X)) n'aura lieu que si TRIGEXPANDPLUS est TRUE.

TRIGINVERSES Variable

par défaut: [ALL] - contrôle la simplification de la composition des fonctions trigonométriques et hyperboliques avec leurs fonctions inverses: si ALL, par exemple les deux fonctions ATAN(TAN(X)) et TAN(ATAN(X)) se simplifient en X. Si TRUE, la simplification de l'arcfunction(function(x)) est évitée. Si FALSE, les deux simplifications arcfun(fun(x)) et fun(arcfun(x)) sont évitées.

TRIGREDUCE (*exp, var*) Fonction

combine les produits et puissances des SIN et COS trigonométriques et hyperboliques de var en ceux des multiples de var. Elle tente aussi d'éliminer ces fonctions lorsqu'elles se trouvent en dénominateurs. Si var est omise toutes les variables dans exp sont utilisées. Voir aussi la fonction POISSIMP (6.6).

```
(C4) TRIGREDUCE(-SIN(X)^2+3*COS(X)^2+X);
(D4)                          2 COS(2 X) + X + 1
Les routines de simplification trigonométrique utiliseront les
informations declarées dans certains cas simples.  Les déclarations à propos de
(C5) DECLARE(J, INTEGER, E, EVEN, O, ODD)$
(C6) SIN(X + (E + 1/2)*%PI)$
(D6)                          COS(X)
(C7) SIN(X + (O + 1/2) %PI);
(D7)                         - COS(X)
```

TRIGSIGN Variable

par défaut: [TRUE] - si TRUE permet la simplification des arguments négatifs des fonctions trigonométriques. P. ex., SIN(-X) deviendra -SIN(X) seulement si TRIGSIGN est TRUE.

TRIGSIMP (*expr*) Fonction

emploie les identités sin(x)^2 + cos(x)^2 = 1 et cosh(x)^2 - sinh(x)^2 = 1 pour simplifier les expressions contenant tan, sec, etc. en sin, cos, sinh, cosh de sorte qu'une simplification ultérieure puisse être obtenue en utilisant TRIGREDUCE sur le résultat. Quelques exemples se trouvent dans DEMO("trgsmp.dem"); . Voir aussi la fonction TRIGSUM.

TRIGRAT (*trigexp*) Fonction

donne une forme cannonique simplifiée quasilinéaire d'une expression trigonométrique; trigexp est une fraction rationnelle de plusieurs sin, cos ou tan, leurs arguments sont des formes linéaires de certaines variables (ou noyaux) et %pi/n (n entier) à coefficients entiers. Le résultat est une fraction simplifiée avec numérateur et dénominateur linéaires en sin et cos. Ainsi TRIGRAT linéarise toujours lorsque c'est possible (écrit par D. Lazard).

```
(c1) trigrat(sin(3*a)/sin(a+%pi/3));

(d1)                    sqrt(3) sin(2 a) + cos(2 a) - 1
```

Voici un autre exemple (pour lequel la fonction a été créée); voir [Davenport, Siret, Tournier, Calcul Formel, Masson (ou en anglais, Addison-Wesley), section 1.5.5, théorème de Morley). Les temps sont ceux d'un VAX 780.

```
(c4)   c:%pi/3-a-b;

                                              %pi
(d4)                                - b - a + ---
                                               3

(c5)   bc:sin(a)*sin(3*c)/sin(a+b);

                                sin(a) sin(3 b + 3 a)
(d5)                            ---------------------
                                     sin(b + a)

(c6)   ba:bc,c=a,a=c$

(c7)   ac2:ba^2+bc^2-2*bc*ba*cos(b);

        2       2
     sin (a) sin (3 b + 3 a)
(d7) -----------------------
                2
             sin (b + a)

                                                %pi
  2 sin(a) sin(3 a) cos(b) sin(b + a - ---) sin(3 b + 3 a)
                                                 3
- ------------------------------------------------------
                            %pi
                  sin(a - ---) sin(b + a)
                             3

     2         2         %pi
  sin (3 a) sin (b + a - ---)
                         3
+ --------------------------
             2     %pi
          sin (a - ---)
                    3

(c9)   trigrat(ac2);
Totaltime= 65866 msec.  GCtime= 7716 msec.

(d9)
- (sqrt(3) sin(4 b + 4 a) - cos(4 b + 4 a)

- 2 sqrt(3) sin(4 b + 2 a)
```

```
+ 2 cos(4 b + 2 a) - 2 sqrt(3) sin(2 b + 4 a) + 2 cos(2 b + 4 a)
+ 4 sqrt(3) sin(2 b + 2 a) - 8 cos(2 b + 2 a) - 4 cos(2 b - 2 a)
+ sqrt(3) sin(4 b) - cos(4 b) - 2 sqrt(3) sin(2 b) + 10 cos(2 b)
+ sqrt(3) sin(4 a) - cos(4 a) - 2 sqrt(3) sin(2 a) + 10 cos(2 a)
   - 9)/4
```

15 Fonctions spéciales

15.1 Introduction aux fonctions spéciales

[index](expr) - *Bessel Funct 1st Kind* (*in SPECINT*) %J

[index](expr) %K

Bessel Funct 2nd Kind (in SPECINT)

Constant, in ODE2

15.2 GAMALG

- Un programme algébrique de matrice gamma de Dirac qui garde la trace et manipule les matrices gamma en n dimensions. Il peut être chargé dans MACSYMA par LOADFILE("gam");. Un manuel préliminaire est contenu dans le fichier SHARE;GAM USAGE et peut être imprimé par PRINTFILE(GAM,USAGE,SHARE);.

15.3 SPECINT

Le package HYPGEO des fonctions hypergéométriques spéciales est encore en développement. Actuellement il trouvera la transformée de Laplace ou plutôt l'intégrale de 0 à INF de certaines fonctions spéciales ou de leurs combinaisons. Le facteur, EXP(-P*var) doit être explicitement déclaré. La syntaxe est la suivante :

SPECINT(EXP(-P*var)*expr,var);

où var est la variable d'intégration et expr peut être une expression quelconque contenant des fonctions spéciales (à vos risques). La notation des fonctions spéciales est :

```
%J[index](expr)          Fonction de Bessel du 1er ordre
%K[index](expr)            "     "     "    du 2ème ordre
%I[     ](    )          Bessel modifiée
%HE[     ](  )          Polynôme de Hermite
%P[  ]( )               Fonction de Legendre
%Q[  ]( )               Legendre du second ordre
HSTRUVE[ ]( )           Struve H Fonction
LSTRUVE[ ]( )             "    L Fonction
%F[ ]([],[],expr)       Fonction hypergéométrique
GAMMA()
GAMMAGREEK()
GAMMAINCOMPLETE()
SLOMMEL
%M[]()                  Fonction de Whittaker du 1er ordre
%W[]()                     "       "    "     du 2ème  "
```

Pour une meilleure compréhension de ce qu'elle peut faites DEMO(HYPGEO,DEMO,SHARE1);

.

15.4 Définitions pour les fonctions spéciales

AIRY (*X*) Fonction

retourne la fonction Airy Ai d'argument réel X. Le fichier SHARE1;AIRY FASL contient des routines d'évaluation des fonctions de Airy Ai(X), Bi(X), et de leurs dérivatées dAi(X), dBi(X). Ai et Bi satisfont l'équation de AIRY diff(y(x),x,2)-x*y(x)=0. Lisez SHARE1;AIRY USAGE pour les détails.

ASYMP Fonction

Une version préliminaire d'un programme pour trouver le comportement asymptotique des diagrammes de Feynman a été installée dans le répertoire SHARE1;. Pour plus d'information, voyez le fichier SHARE1;ASYMP USAGE. (Pour les fonctions d'analyse asymptotique, voir ASYMPA).

ASYMPA Fonction

Analyse asymptotique - Le fichier SHARE1;ASYMPA > contient les fonctions de simplification d'analyse asymptotique, y compris les fonctions big-O et little-o qui sont largement utilisées en analyse de complexité et en analyse numérique. Faites BATCH("asympa.mc");. (Pour le comportement asymptotique des diagrammes de Feynman, voir ASYMP).

BESSEL (*Z*,*A*) Fonction

retourne la fonction J de Bessel pour le complexe Z et le réel A > 0.0 . Un tableau BESSELARRAY est aussi créé de telle sorte que BESSELARRAY[I] = J[I+A- ENTIER(A)](Z).

BETA (*X*, *Y*) Fonction

identique à GAMMA(X)*GAMMA(Y)/GAMMA(X+Y).

GAMMA (*X*) Fonction

la fonction gamma. GAMMA(I)=(I-1)! pour I un entier positif. Pour la constante de Euler-Mascheroni, voir %GAMMA. Voir aussi la fonction MAKEGAMMA. La variable GAMMALIM[1000000] contrôle la simplification de la fonction gamma.

GAMMALIM Variable

par défaut: [1000000] - contrôle la simplification de la fonction gamma pour les arguments en nombres entiers et rationnels. Si la valeur absolue de l'argument n'est pas plus grand que GAMMALIM, alors la simplification sera effectuée. Notez que le commutateur FACTLIM contrôle aussi la simplification du résultat de GAMMA d'un argument entier.

INTOPOIS (*A*) Fonction

convertit A en codage de Poisson.

MAKEFACT (*exp*) Fonction

transforme les occurrences des fonctions binôme,gamma, et beta de exp en factorielles.

MAKEGAMMA (*exp*) Fonction

transforme les occurrences des fonctions binôme,gamma, et beta de exp en fonctions gamma.

NUMFACTOR (*exp*) Fonction

donne le facteur numérique multipliant l'expression exp qui devra être un terme unique. Si le pgcd de tous les termes d'une somme est cherché la fonction CONTENT peut être utilisée.

```
(C1) GAMMA(7/2);
(D1)                15 SQRT(%PI)
                    ------------
                         8
(C2) NUMFACTOR(%);
                     15
(D2)                 --
                      8
```

OUTOFPOIS (*A*) Fonction

convertit A en codage de Poisson en représentation générale. Si A n'est pas sous forme de Poisson, la conversion se fera, i.e. elle ressemblera au résultat de OUTOFPOIS(INTOPOIS(A)). Cette fonction est ainsi un simplificateur sous forme cannonique de sommes de puissances de SIN et COS d'un type particulier.

POISDIFF (*A, B*) Fonction

différentie A par rapport à B. B ne doit apparaître que comme arguments trigonométriques ou seulement dans les coefficients.

POISEXPT (*A, B*) Fonction

(B entier positif) est fonctionnellement identique à INTOPOIS(A**B).

POISINT (*A, B*) Fonction

intègre dans un sens restraint identique à POISDIFF. Les termes non périodiques de B sont ignorés si B est dans les arguments trigonométiques.

POISLIM Variable

par défaut: [5] - détermine le domaine des coefficients dans les arguments des fonctions trigonométriques. La valeur initiale de 5 correspond à l'intervalle [-2^(5-1)+1,2^(5-1)], ou [-15,16], mais il peut être mis à [-2^(n-1)+1, 2^(n-1)].

POISMAP (*series, sinfn, cosfn*) Fonction

va appliquer les fonctions sinfn sur les termes sinus et cosfn sur les termes cosinus de la série de Poisson donnée. sinfn et cosfn sont des fonctions de deux arguments qui sont respectivement un coefficient et une partie trigonométrique d'un terme de series.

POISPLUS (*A, B*) Fonction

est fonctionnellement identique à INTOPOIS(A+B).

POISSIMP (*A*) Fonction

convertit A en représentation générale en une série de Poisson.

POISSON special symbol

Le symbol /P/ suit la ligne étiquette des expressions en séries de Poisson.

POISSUBST (*A, B, C*) Fonction

substitue A à B dans C. C est une suite de Poisson. (1) Si B est une variable U, V, W, X, Y, ou Z alors A doit être une expression linéaire de ces variables (p. ex. 6*U+4*V). (2) Si B est autre que ces variables, alors A doit aussi être indépendant de ces variables, et de plus indépendant de sinus ou cosinus.

POISSUBST(A, B, C, D, N) est un type spécial de substitution qui opère sur A et B comme en (1) ci-dessus, mais où D est une suite de Poisson, développe COS(D) et SIN(D) d'ordre N afin de fournir le résultat de la substitution de A+D pour B dans C. L'idée est que D est un développement en termes d'un petit paramètre. Par exemple, POISSUBST(U,V,COS(V),E,3) donne COS(U)*(1-E^2/2) - SIN(U)*(E-E^3/6).

POISTIMES (*A, B*) Fonction

est fonctionnellement identique à INTOPOIS(A*B).

POISTRIM () Fonction

est un nom de fonction réservé qui (si l'utilisateur l'a défini) est appliquée lors d'une multiplication de Poisson. C'est une fonction prédicat de 6 arguments qui sont les coefficients des U, V,..., Z d'un terme. Les termes pour lesquels POISTRIM est TRUE (pour les coefficients de ce terme) sont éliminés pendant la multiplication.

PRINTPOIS (*A*) Fonction

affiche une suite de Poisson dans un format lisible. Comme avec OUTOFPOIS, elle convertit d'abord A en un codage de Poisson, si nécessaire.

PSI (*X*) Fonction

déivée de LOG(GAMMA(X)). Actuellement, MACSYMA n'a pas de possibilités d'évaluation numérique pour PSI. Pour avoir des informations sur la notation PSI[N](X), voyez POLYGAMMA.

16 Polynômes orthogonaux

16.1 Introduction aux polynômes orthogonaux

Le package specfun, situé dans le répertoire share, contient du code Maxima pour l'évaluation de tous les polynômes orthogonaux donnés dans la liste du chapitre 22 de l'Abramowitz et Stegun. Ils comprennent les polynômes de Tchebychev, de Laguerre, de Hermite, de Jacobi, de Legendre, et les polynômes ultrasphériques (Gegenbauer). De plus, specfun contient du code pour les fonctions sphériques de Bessel, de Hankel, et sphériques harmoniques.

La table suivante donne la liste de chacune des fonctions de specfun, son nom dans Maxima, les restrictions sur ses arguments (m et n doivent être des entiers), et une référence à l'algorithme qu'utilise specfun pour l'évaluer. Avec quelques exceptions, specfun respecte les conventions de Abramowitz et Stegun. Avant d'utiliser specfun, vérifiez que les conventions de specfun sont conformes à votre attente.

A&S se réfère à Abramowitz et Stegun, *Handbook of Mathematical Fonctions* (10th printing, December 1972), G&R à Gradshteyn et Ryzhik, *Table of Integrals, Series, and Products* (1980 édition corrigée et augmentée), et Merzbacher à *Quantum Mechanics* (2ed, 1970).

Fonction	*Nom dans Maxima*	*Restrictions*	*Référence(s)*
Tchebychev T	chebyshev_t(n, x)	$n> -1$	A&S 22.5.31
Tchebychev U	chebyshev_u(n, x)	$n> -1$	A&S 22.5.32
Laguerre généralisé	gen_laguerre(n,a,x)	$n> -1$	A&S page 789
Laguerre	laguerre(n,x)	$n> -1$	A&S 22.5.67
Hermite	hermite(n,x)	$n> -1$	A&S 22.4.40, 22.5.41
Jacobi	jacobi_p(n,a,b,x)	$n> -1, a, b> -1$	A&S page 789
Legendre P associé	assoc_legendre_p(n,m,x)	$n> -1$	A&S 22.5.37, 8.6.6, 8.2.5
Legendre Q associé	assoc_legendre_q(n,m,x)	$n> -1, m> -1$	G & R 8.706
Legendre P	legendre_p(n,m,x)	$n> -1$	A&S 22.5.35
Legendre Q	legendre_q(n,m,x)	$n> -1$	A&S 8.6.19
Hankel 1er sphérique	spherical_hankel1(n, x)	$n> -1$	A&S 10.1.36
Hankel 2ème sphérique	spherical_hankel2(n, x)	$n> -1$	A&S 10.1.17
Bessel J sphérique	spherical_bessel_j(n,x)	$n> -1$	A&S 10.1.8, 10.1.15
Bessel Y sphérique	spherical_bessel_y(n,x)	$n> -1$	A&S 10.1.9, 10.1.15
harmonique sphérique	spherical_harmonic(n,m,x,y)	$n> -1, \|m\|<=n$	Merzbacher 9.64
ultrasphérique (Gegenbauer)	ultraspherical(n,a,x)	$n> -1$	A&S 22.5.27

Le package specfun est essentiellement conçu pour le calcul symbolique. On espère qu'il donne aussi des résultats précis en virgule flottante; cependant, on ne prétend pas que les algorithmes conviennent bien à l'évaluation numérique. Certains efforts ont cependant été faits pour fournir de bonnes performances numériques. Lorsque tous les arguments, sauf

l'ordre, sont des réels (des flottants, mais pas des "bfloats" ???), beaucoup des fonctions de specfun appellent une version déclarée "flottante" de la fonction de Jacobi, ce qui accélère beaucoup l'évaluation en virgule flottante des polynômes orthogonaux.

specfun traite la plupart des erreurs du domaine en retournant une fonction non évaluée. Aucune tentative n'a été faite pour définir des règles de simplification (basées sur des relations de récursion) pour des fonctions non évaluées. Les utilisateurs devront être conscients qu'il est possible qu'une expression impliquant des sommes de fonctions spéciales non évaluées puisse disparaître, et que Maxima est incapable de la réduire à zéro. Soyez prudent.

Pour accéder aux fonctions de specfun, vous devez d'abord charger specfun.o. Ou vous pouvez ajouter des instructions d'auto-chargement à votre fichier init.lsp (situé dans votre répertoire de travail). Pour auto-charger, par exemple, la fonction hermite, ajoutez

```
(defprop |$hermite| #"specfun.o" autoload)
(add2lnc '|$hermite| $props)
```

à votre fichier init.lsp. Voici un exemple d'utilisation de specfun :

```
(c1) load("specfun.o")$
(c2) [hermite(0,x),hermite(1,x),hermite(2,x)];
(d2) [1,2*x,-2*(1-2*x^2)]
(c3) diff(hermite(n,x),x);
(d3) 2*n*hermite(n-1,x)
```

Lors de l'utilisation d'une version compilée de specfun, soyez particulièrement attentif à donner le nombre correct d'arguments à la fonction; l'appeler avec trop peu d'arguments peut générer un message d'erreur fatal. Par exemple

```
(c1) load("specfun")$
/* chebyshev_t requiert deux arguments. */
(c2) chebyshev_t(8);
Erreur: capture d'une erreur fatale [la mémoire peut être endommagée]
Les liens rapides sont activés : faire (si::use-fast-links nil) pour débogage
Erreur signalée par MMAPCAR.
Interrompue à SIMPLIFY.  Tapez :H pour de l'aide.
```

Le code maxima traduit en Lisp traite de telles erreurs plus élégamment. Si specfun.LISP est installé sur votre machine, le même calcul renvoie un message d'erreur plus clair. Par exemple

```
(c1) load("specfun.LISP")$
(c2) chebyshev_t(8);
Erreur: attend 2 args mais en reçoit 1
Les liens rapides sont activés: faire (si::use-fast-links nil) pour débogage
Erreur signalé par MACSYMA-TOP-LEVEL.
Interrompue à |$CHEBYSHEV_T|.  Tapez :H pour de l'aide.
```

Généralement, le code compilé s'exécute plus rapidement que le code traduit ; cependant le code traduit peut être meilleur pour le développement de programme.

Pour certaines fonctions, lorsque l'ordre est symbolique mais a été déclaré comme étant un entier, specfun retourne une représentation en série (cette représentation n'est utilisée par specfun dans aucun calcul). Vous pouvez utiliser cette particularité pour trouver les valeurs symboliques pour des valeurs spéciales de polynômes orthogonaux. Un exemple:

```
(c1) load("specfun")$
(c2) legendre_p(n,1);
(d2)        legendre_p(n, 1)
/* Déclare n comme étant entier ; alors legendre_p(n,1) s'évalue à 1. */
(c3) declare(n,integer)$
(c4) legendre_p(n,1);
(d4)        1
(c5) ultraspherical(n,3/2,1);
(d4)         (n+1)*gamma (n+3) / (2*gamma (n+2))
```

Bien que cet exemple ne le montre pas, deux termes de la somme sont ajoutés en dehors de la sommation. Enlever ces deux termes évite les erreurs associées aux termes 0^0 dans une somme qui devrait s'évaluer à 1, mais s'évalue à 0 dans une sommation Maxima. Comme les indices de la somme vont de 1 à $n-1$, l'indice le plus bas serait supérieur au plus haut lorsque $n=0$; mettre sumhack à vrai corrige cela. Par exemple:

```
(c1) load("specfun.o")$
(c2) declare(n,integer)$
(c3) e : legendre_p(n,x)$
(c4) ev(e,sum,n=0);
Limite inférieur de SUM: 1
est plus grande que la limite supérieure: - 1
 -- une erreur.  Terminé.  Pour déboguer essayez DEBUGMODE(TRUE);)
(c5) ev(e,sum,n=0),sumhack : true;
(d5)         1
```

La plupart des fonctions de specfun possèdent une propriété (gradef ???) ; les dérivées par rapport à l'ordre ou à d'autres paramètres de fonction ne sont pas évaluées.

Le package specfun et sa documentation ont été écrits par Barton Willis de l'Université du Nebraska à Kearney. Il est distribué sous les termes de la General Public License (GPL). Envoyez les rapports de bogues et les commentaires concernant ce package à willisb@unk.edu. Dans votre rapport, mettez la version de Maxima et de specfun. Celle de specfun peut être trouvée avec :

```
(c2) get('specfun,'version);
(d2)          110
```

16.2 Définitions pour les polynômes orthogonaux

ASSOC_LEGENDRE_P (n, m, x) Fonction

[package specfun] retourne la fonction associée de Legendre du premier ordre pour les entiers $n> -1$ et $m> -1$. Lorsque $|m|>n$ et $n> =0$, nous avons $assoc\_legendre\_p(n,m,x)=0$.

Référence: A&S 22.5.37 page 779, A&S 8.6.6 (seconde équation) page 334, et A&S 8.2.5 page 333.

Pour avoir accès à cette fonction, load("specfun").

Voir [ASSOC_LEGENDRE_Q], page 114, [LEGENDRE_P], page 115, et [LEGENDRE_Q], page 115.

ASSOC_LEGENDRE_Q (*n*, *m*, *x*) Fonction

[package specfun] retourne la fonction associée de Legendre du second ordre pour les entiers $n> -1$ et $m> -1$.

Référence: Gradshteyn et Ryzhik 8.706 page 1000.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi ASSOC_LEGENDRE_P, LEGENDRE_P, et LEGENDRE_Q.

CHEBYSHEV_T (*n*, *x*) Fonction

[package specfun] retourne la fonction de Tchebychev du premier ordre pour les entiers $n> -1$.

Référence: A&S 22.5.31 page 778 et A&S 6.1.22 page 256.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi CHEBYSHEV_U.

CHEBYSHEV_U (*n*, *x*) Fonction

[package specfun] retourne la fonction de Tchebychev du second ordre pour les entiers $n> -1$.

Référence: A&S, 22.8.3 page 783 et A&S 6.1.22 page 256.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi CHEBYSHEV_T.

GEN_LAGUERRE (*n*, *a*, *x*) Fonction

[package specfun] retourne le polynôme généralisé de Laguerre pour les entiers $n> -1$.

Pour avoir accès à cette fonction, load("specfun").

Référence: table de la page 789 dans A&S.

HERMITE (*n*,*x*) Fonction

[package specfun] retourne le polynôme de Hermite pour les entiers $n> -1$.

Pour avoir accès à cette fonction, load("specfun").

Référence: A&S 22.5.40 et 22.5.41, page 779.

JACOBI_P (*n*, *a*, *b*, *x*) Fonction

[package specfun] retourne le polynôme de Jacobi pour les entiers $n> -1$ et a et b symboliques ou $a> -1$ et $b> -1$. (Les polynômes de Jacobi sont en fait définis pour tous a et b ; cependant le polynôme de Jacobi de poids $(1-x)\hat{}a(1+x)\hat{}b$ n'est pas intégrable pour $a<=-1$ ou $b<=-1$).

Lorsque a, b, et x sont des flottants (mais pas des bfloats) specfun appelle une version spéciale de *jacobi_p*. Pour des valeurs numériques, cette version est plus rapide que l'autre version. Beaucoup des fonctions de specfun sont calculées comme cas spécial des polynômes de Jacobi ; elles apprécient aussi l'accroissement de vitesse procuré par la version spéciale de *jacobi*.

Si n a été déclaré comme étant un entier, $jacobi\_p(n, a, b, x)$ retourne une représentation en sommation de la fonction de Jacobi. Comme Maxima simplifie $0\hat{}0$ en 0 dans une somme, deux termes de la somme sont ajoutés en dehors de la summation.

Pour avoir accès à cette fonction, load("specfun").

Référence: table de la page 789 dans A&S.

LAGUERRE (*n*, *x*) Fonction

[package specfun] retourne le polynôme de Laguerre pour les entiers $n > -1$.

Référence: A&S 22.5.16, page 778 et A&S page 789.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi GEN_LAGUERRE.

LEGENDRE_P (*n*, *x*) Fonction

[package specfun] retourne le polynôme de Legendre du premier ordre pour les entiers $n > -1$.

Référence: A&S 22.5.35 page 779.

Pour avoir accès à cette fonction, load("specfun").

Voir [LEGENDRE_Q], page 115.

LEGENDRE_Q (*n*, *x*) Fonction

[package specfun] retourne le polynôme de Legendre du premier ordre pour les entiers $n > -1$.

Référence: A&S 8.6.19 page 334.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi LEGENDRE_P.

SPHERICAL_BESSEL_J (*n*, *x*) Fonction

[package specfun] retourne la fonction sphérique de Bessel du premier ordre pour les entiers $n > -1$.

Référence: A&S 10.1.8 page 437 et A&S 10.1.15 page 439.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi SPHERICAL_HANKEL1, SPHERICAL_HANKEL2, et SPHERICAL_BESSEL_Y.

SPHERICAL_BESSEL_Y (*n*, *x*) Fonction

[package specfun] retourne la fonction sphérique de Bessel du second ordre pour les entiers $n > -1$.

Référence: A&S 10.1.9 page 437 et 10.1.15 page 439.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi SPHERICAL_HANKEL1, SPHERICAL_HANKEL2, and SPHERICAL_BESSEL_Y.

SPHERICAL_HANKEL1 (*n*,*x*) Fonction

[package specfun] retourne la fonction sphérique de Hankel du premier ordre pour les entiers $n> -1$.

Référence: A&S 10.1.36 page 439.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi SPHERICAL_HANKEL2, SPHERICAL_BESSEL_J, and SPHERICAL_BESSEL_Y.

SPHERICAL_HANKEL2 (*n*,*x*) Fonction

[package specfun] retourne la fonction sphérique de Hankel du second ordre pour les entiers $n> -1$.

Référence: A&S 10.1.17 page 439.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi SPHERICAL_HANKEL1, SPHERICAL_BESSEL_J, and SPHERICAL_BESSEL_Y.

SPHERICAL_HARMONIC (*n*, *m*, *x*, *y*) Fonction

[package specfun] retourne la fonction sphérique harmonique pour les entiers $n> -1$ et $|m|<=n$.

Référence: Merzbacher 9.64.

Pour avoir accès à cette fonction, load("specfun").

Voir aussi ASSOC_LEGENDRE_P

ULTRASPHERICAL (*n*,*a*,*x*) Fonction

[package specfun] retourne les polynômes ultrasphériques pour les entiers $n>-1$. Les polynômes ultrasphériques sont aussi connus sous le nom de polynômes de Gegenbauer.

Référence: A&S 22.5.27

Pour avoir accès à cette fonction, load("specfun").

Voir aussi JACOBI_P.

17 Limites

17.1 Définitions pour les limites

LHOSPITALLIM Variable

par défaut: [4] - le nombre maximal de fois que la règle de L'Hospital est utilisée par LIMIT. Ceci empêche un bouclage infini dans des cas comme LIMIT(COT(X)/CSC(X),X,0).

LIMIT (*exp, var, val, dir*) Fonction

trouve la limite de exp lorsque la variable réelle var approche de la valeur val depuis la direction dir. dir peut avoir la valeur PLUS pour une limite depuis le haut, MINUS pour une limite depuis le bas, ou peut être omise (impliquant qu'une limite des deux côtés doit être calculée). Pour la méthode voyez Wang, P., "Evaluation of Definite Integrals by Symbolic Manipulation" - Ph.D. Thesis - MAC TR-92 October 1971. LIMIT utilise les symboles spéciaux suivants : INF (infini positif) et MINF (infini négatif). Sur la sortie elle peut utiliser aussi UND (non définie), IND (non définie mais bornée) et INFINITY (infini complexe).

LHOSPITALLIM[4] est le nombre maximal de fois que la règle de L'Hospital est utilisée par LIMIT. Ceci empêche un bouclage infini dans des cas comme LIMIT(COT(X)/CSC(X),X,0).

TLIMSWITCH[FALSE] si TRUE le package limit utilisera si possible les séries de Taylor.

LIMSUBST[FALSE] empêche LIMIT de tenter des substitutions sur des formes inconnues. C'est pour éviter des bogues comme LIMIT(F(N)/F(N+1),N,INF); donnant 1. Mettre LIMSUBST à TRUE permet de telles substitutions. Comme LIMIT est souvent appelée pour simplifier des expressions constantes, par exemple, INF-1, LIMIT peut être utilisée dans des cas avec seulement un argument, p. ex. LIMIT(INF-1); . Faites EXAMPLE(LIMIT); pour les exemples.

TLIMIT (*exp,var,val,dir*) Fonction

est exactement la fonction LIMIT avec TLIMSWITCH mise à TRUE.

TLIMSWITCH Variable

par défaut: [FALSE] - si TRUE le package limit utilisera si possible les séries de Taylor

18 Différentiation

18.1 Définitions pour la différentiation

ANTID (*G,X,U(X)*) Fonction

Une routine pour évaluer les intégrales des expressions impliquant une fonction arbitraire non spécifiée et ses dérivées. Elle peut être utilisée par LOAD(ANTID); , après quoi par la fonction ANTIDIFF. P. ex. ANTIDIFF(G,X,U(X)); où G est l'expression impliquant U(X) (U(X) quelconque) et ses dérivées, dont l'intégrale par rapport à X est recherchée.

Les fonctions NONZEROANDFREEOF et LINEAR sont aussi définies, ainsi que ANTID. ANTID est comme ANTIDIFF sauf qu'elle retourne une liste en deux parties, la première est la partie intégrée de l'expression et la seconde le reste non intégrable.

ANTIDIFF - Fonction

Voir ANTID.

ATOMGRAD property

La propriété de gradient atomique d'une expression. Peut être définie par GRADEF.

ATVALUE (*forme, liste, valeur*) Fonction

permet à l'utilisateur d'affecter à forme la valeur des bornes aux points spécifiés par liste.

```
(C1) ATVALUE(F(X,Y),[X=0,Y=1],A**2)$
  La forme doit être une fonction, f(v1,v2,...), ou une dérivée,
```

DIFF(f(v1,v2,...),vi,ni,vj,nj,...) dans laquelle les arguments fonctionnels apparaîssent explicitement (ni est l'ordre de différentiation par rapport à vi). La liste des équations détermine la "frontière" à laquelle la valeur est donnée; liste peut être une liste d'équations, comme ci-dessus, ou une unique équation, vi = expr. Les symboles @1, @2,... seront utilisés pour représenter les variables fonctionnelles v1,v2,... lorsque les "atvalues" sont affichées.

PRINTPROPS([f1, f2,...], ATVALUE) afficheront les atvalues des fonctions f1,f2,... comme spécifié précédemment pour l'utilisation de la fonction ATVALUE. Si la liste contient seulement un élément alors cet élément peut être donné sans être une liste. Si un premier argument ALL est donné alors les "atvalues" pour toutes les fonctions qui les contiennent seront affichées. Faites EXAMPLE(ATVALUE); pour un exemple.

CARTAN Fonction

Le calcul intégral extérieur des formes différentielles est un outil de base de la géométrie différentielle développée par Elie Cartan et a d'importantes applications dans la théorie des équations différentielles partielles. L'implantation actuelle est dûe à F.B. Estabrook et H.D. Wahlquist. Le programme s'explique de lui-même et on y accède en faisant batch("cartan"); qui en donne une description avec des exemples.

DELTA (*t*) Fonction

Fonction delta de Dirac. Actuellement seule LAPLACE reconnait la fonction DELTA:

```
(C1) LAPLACE(DELTA(T-A)*SIN(B*T),T,S);
Is A positive, negative or zero?
POS;
                                              - A S
(D1)                          SIN(A B) %E
```

DEPENDENCIES Variable

par défaut: [] - la liste des atomes qui ont des dépendances fonctionnelles (définies par les fonctions DEPENDS ou GRADEF). La commande DEPENDENCIES a été remplacée par la commande DEPENDS. Faites DESCRIBE(DEPENDS);

DEPENDS (*funlist1*,*varlist1*,*funlist2*,*varlist2*,...) Fonction

déclare les dépendances fonctionnellss des variables utilisées par DIFF.

```
DEPENDS([F,G],[X,Y],[R,S],[U,V,W],U,T)
```

informe DIFF que F et G dépendent de X et Y, que R et S dépendent de U, V, W, et que U dépend de T. Les arguments de DEPENDS sont évalués. Les variables de chaque funlist sont déclarées dépendre de toutes les variables de la varlist suivante. Une funlist peut contenir le nom d'une variable atomique ou d'un tableau. Dans ce dernier cas, il est supposé que tous les éléments du tableau dépendent de toutes les variables de la varlist qui suit. Initialement, DIFF(F,X) est 0; après DEPENDS(F,X), les différentiations futures de F par rapport à X donneront dF/dX ou Y (si DERIVABBREV:TRUE).

```
(C1) DEPENDS([F,G],[X,Y],[R,S],[U,V,W],U,T);
(D1)           [F(X, Y), G(X, Y), R(U, V, W), S(U, V, W), U(T)]
(C2) DEPENDENCIES;
(D2)           [F(X, Y), G(X, Y), R(U, V, W), S(U, V, W), U(T)]
(C3) DIFF(R.S,U);
                                  dR           dS
(D3)                              -- . S + R . --
                                  dU           dU
```

Comme MACSYMA connaît la règle de la chaîne pour les dérivées symboliques, elle profite des dépendances données comme suit :

```
(C4) DIFF(R.S,T);
                             dR dU             dS dU
(D4)                        (-- --) . S + R . (-- --)
                             dU dT             dU dT
Si nous donnons
(C5) DERIVABBREV:TRUE;
(D5)                                  TRUE
alors en ré-exécutant la commande C4, nous obtenons
(C6) ''C4;
                                 (R  U ) . S + R . (S  U )
(D6)                               U  T              U  T
```

Pour éliminer une dépendance précédemment déclarée, la commande REMOVE peut être utilisée. Par exemple, pour dire que R ne dépend plus de U comme déclaré dans C1, l'utilisateur peut taper

```
REMOVE(R,DEPENDENCY)
```

Ceci éliminera toutes les dépendances pouvant avoir été déclarées pour R.

```
(C7) REMOVE(R,DEPENDENCY);
(D7)                                    DONE
(C8) ''C4;
(D8)                                R . (S  U )
                                     U    T
```

AVERTISSEMENT: DIFF est la seule commande MACSYMA qui utilise les informations de DEPENDENCIES. Il faut donner explicitement leurs dépendances aux arguments de INTEGRATE, LAPLACE, etc. dans la commande, p. ex., INTEGRATE(F(X),X).

DERIVABBREV Variable

par défaut: [FALSE] - si TRUE fera afficher les dérivées sous forme d'indices.

DERIVDEGREE (*exp, dv, iv*) Fonction

trouve le plus haut degré de la dérivée de la variable dépendante dv par rapport à la variable indépendante iv se trouvant dans exp.

```
(C1) 'DIFF(Y,X,2)+'DIFF(Y,Z,3)*2+'DIFF(Y,X)*X**2$
(C2) DERIVDEGREE(%,Y,X);
(D2)                            2
```

DERIVLIST (*var1,...,vark*) Fonction

provoque seulement les différentiations par rapport aux variables indiquées, sans la commande EV.

DERIVSUBST Variable

par défaut: [FALSE] - contrôle les substitutions non syntaxiques, comme

```
SUBST(X,'DIFF(Y,T),'DIFF(Y,T,2));
```

Si DERIVSUBST est mise à TRUE, cela donne 'DIFF(X,T).

DIFF symbole spécial

[drapeau] pour ev provoque l'exécution de toutes les différentiations indiquées dans exp.

DIFF (*exp, v1, n1, v2, n2, ...*) Fonction

DIFF différencie exp par rapport à chacun des vi, ni fois. Si seulement la première dérivée par rapport à une variable est cherchée, la forme DIFF(exp,v) peut être utilisée. Si la forme nominale de la fonction est requise (comme, par exemple, lors de

l'écriture d'une équation différentielle), 'DIFF devra être utilisée et l'affichage se fera en deux dimensions.

DERIVABBREV[FALSE] si TRUE provoquera l'affichage des dérivées en indices.

DIFF(exp) donne la "différentielle totale", c.-à-d. la somme des dérivées de exp par rapport à chacune de ses variables multipliée par la fonction DEL de la variable. Aucune simplification ultérieure de DEL n'est proposée.

```
(C1) DIFF(EXP(F(X)),X,2);
                              2
                        F(X)  d              F(X)  d          2
(D1)                   %E     (--- F(X)) + %E     (-- (F(X)))
                               2                   dX
                              dX
(C2) DERIVABBREV:TRUE$
(C3) 'INTEGRATE(F(X,Y),Y,G(X),H(X));
                                  H(X)
                                 /
                                 [
(D3)                             I     F(X, Y) dY
                                 ]
                                 /
                                  G(X)
(C4) DIFF(%,X);
              H(X)
             /
             [
(D4)         I     F(X, Y)  dY + F(X, H(X)) H(X)  - F(X, G(X)) G(X)
             ]            X                     X                   X
             /
              G(X)
```

Les modifications suivantes ont été incorporées au package tensor : 1) les dérivées de tous les objets indicés dans exp auront les variables vi ajoutées comme arguments additionnels. Alors tous les indices des dérivées seront triés. 2) les vi peuvent être des entiers de 1 jusqu'à la valeur de la variable DIMENSION [par défaut : 4]. Ceci provoque la différentiation wrt jusqu'au vème membre de la liste COORDINATES qui devra être une liste des noms des coordonnées, p. ex., [x,y,z,t]. Si COORDINATES est liée à une variable atomique, la variable indicée par vi sera utilisée comme variable de différentiation. Ceci permet l'utilisation d'un tableau de noms de coordonnées ou de noms d'indices comme X[1], X[2],.... Si aucune valeur n'a été donnée à COORDINATES, alors les variables seront traitées comme dans 1) ci-dessus.

DSCALAR (*fonction*) Fonction

applique le scalaire d'Alembertien à la fonction scalar.

```
(C41) DEPENDENCIES(FIELD(R));
(D41)                            [FIELD(R)]
(C42) DSCALAR(FIELD);
(D43)
```

```
   -M
 %E  ((FIELD  N - FIELD  M + 2 FIELD   ) R + 4 FIELD )
            R  R       R  R         R R             R

- ------------------------------------------------------
                                2 R
```

EXPRESS (*expression*) Fonction

Le résultat utilise la forme nominale des dérivées provenant du développement des opérateurs différentiels vectoriels. Pour forcer l'évaluation de ces dérivées, la fonction interne EV peut être utilisée avec le drapeau DIFF, après utilisation de la fonction interne DEPENDS, pour établir toutes les nouvelles dépendances implicites.

GENDIFF Fonction

Quelquefois DIFF(E,X,N) peut être réduite bien que N soit symbolique.

```
batch("gendif.mc")$
```

et vous pouvez essayer, par exemple,

```
DIFF(%E^(A*X),X,Q)
```

en utilisant GENDIFF plutôt que DIFF. Les items non évaluables sont mis entre guillemets. Certains items sont en termes de "GENFACT"(, which see ???).

GRADEF (*f*(*x1, ..., xn*)*, g1, ..., gn*) Fonction

définit les dérivées de la fonction f par rapport à ses n arguments, i. e., df/dxi = gi, etc. Si moins de n gradients, disons i, sont donnés, alors ils se réfèrent aux i premiers arguments de f. Les xi sont simplement des variables factices dans les entêtes de la définition de la fonction et sont utilisées pour indiquer le ième argument de f. Tous les arguments pour GRADEF sauf le premier sont évalués de sorte que si g est une fonction définie elle est invoquée et le résultat est utilisé. Les gradients sont nécessaires lorsque, par exemple, une fonction n'est pas connues explicitement mais ses premières dérivées le sont et on veut trouver ses dérivées d'ordres supérieurs. GRADEF peut aussi être utilisée pour redéfinir les dérivées des fonctions prédéfinies de MACSYMA (p. ex. GRADEF(SIN(X),SQRT(1-SIN(X)**2))). Il n'est pas permis d'utiliser GRADEF sur des fonctions indicées.

GRADEFS est une liste de fonctions auxquelles ont été attribuées des gradients avec la commande GRADEF (i.e. GRADEF(f(x1, ..., xn), g1, ..., gn)).

PRINTPROPS([f1,f2,...],GRADEF) peut être utilisée pour afficher les (gradefs ???) des fonctions f1,f2,..

GRADEF(a,v,exp) peut être utilisée pour déclarer que exp est la dérivée de la variable atomique a par rapport à v. Ceci fait automatiquement un DEPENDS(a,v).

PRINTPROPS([a1,a2,...],ATOMGRAD) peut être utilisé pour afficher les propriétés du gradient atomique de a1,a2,...

GRADEFS Variable

par défaut: [] - une liste des fonctions auxquelles ont été attribuées des gradients avec la commande GRADEF (i.e. GRADEF(f(x1, ..., xn), g1, ..., gn).)

LAPLACE (*exp, ovar, lvar*) Fonction

prend la transformée de Laplace de exp par rapport à la variable ovar et le paramètre transformé lvar. Exp ne peut impliquer que les fonctions EXP, LOG, SIN, COS, SINH, COSH, et ERF. Elle peut aussi être une équation différentielle à coefficients linéaires constants auquel cas la ATVALUE de la variable dépendante sera utilisée. Ils peuvent être fournis soit avant soit après la transformation. Comme les conditions initiales doivent être spécifiées au zéro, si des conditions aux limites sont imposées ailleurs on peut les imposer sur la solution générale et éliminer les constantes en résolvant la solution générale pour elles et en substituant ensuite leurs valeurs. Exp peut aussi impliquer des intégrales de convolution. Des relations fonctionnelles doivent être explicitement représentées afin que LAPLACE fonctionne correctement. C'est à dire que si F dépend de X et Y elle doit être écrite F(X,Y) partout où F est présente, comme dans LAPLACE('DIFF(F(X,Y),X),X,S). LAPLACE n'est pas affectée par les DEPENDENCIES définies par la commande DEPENDS.

```
(C1) LAPLACE(%E**(2*T+A)*SIN(T)*T,T,S);
                          A
                       2 %E  (S - 2)
(D1)                   ---------------
                                 2     2
                       ((S - 2)  + 1)
```

UNDIFF (*exp*) Fonction

retourne une expression équivalente à exp mais où toutes les dérivées des objets indicés sont remplacées par la forme nominale de la fonction DIFF avec des arguments qui auraient contenus ces objets si la différentiation avait été effectuée. C'est utile si on veut remplacer un objet indicé différencié par une définition de fonction puis exécuter la différentiation en faisant EV(...,DIFF).

19 Intégration

19.1 Introduction à l'intégration

MACSYMA possède plusieurs routines de traitement de l'intégration. La commande INTEGRATE utilise la plupart d'entre elles. Existe aussi le package ANTID, qui traite une fonction non spécifiée (ainsi bien sûr que ses dérivées). Pour les calculs numériques, il y a la fonction de ROMBERG, et la version IMSL de Romberg, DCADRE. Il y a aussi un intégrateur adaptatif qui utilise la règle de la quadrature de Newton-Cotes à 8 panneaux, appelée QUANC8. Les fonctions hypergeométriques sont aussi traitées, faites DESCRIBE(SPECINT); pour les détails.

D'une façon générale, MACSYMA ne traite que les intégrales qui le sont en termes de "fonctions élémentaires" (fonctions rationnelles, trigonométriques, logarithmiques, exponentelles, radicaux, etc.) avec quelques extensions (fonction d'erreur, dilogarithme). Elle ne traite pas les intégrales de fonctions inconnues telles que g(x) et h(x).

19.2 Définitions pour l'intégration

CHANGEVAR (*exp,f*(*x,y*),*y,x*) Fonction

exécute le changement de variable donné par f(x,y) = 0 dans toutes les intégrales se trouvant dans exp avec intégration par rapport à x; y est la nouvelle variable.

```
(C1) 'INTEGRATE(%E**SQRT(A*Y),Y,0,4);
                      4
                     /
                     [    SQRT(A) SQRT(Y)
(D1)                 I (%E                ) dY
                     ]
                     /
                      0
(C2) CHANGEVAR(D1,Y-Z^2/A,Z,Y);
                                  2 SQRT(A)
                                 /
                                 [              Z
                               2 I          Z %E  dZ
                                 ]
                                 /
                                  0
(D4)                             ----------------------
                                            A
```

CHANGEVAR peut aussi être utilisée pour changer les indices d'une somme ou d'un produit. Cependant, il faut bien voir que si un changement est fait dans une somme ou un produit, ce changement doit être un décalage, i.e. I=J+ ..., pas une fonction de plus haut degré. P. ex.

```
(C3) SUM(A[I]*X^(I-2),I,0,INF);
                                 INF
```

```
                                 ====
                                 \          I - 2
(D3)                              >     A  X
                                 /       I
                                 ====
                                 I = 0
(C4) CHANGEVAR(%,I-2-N,N,I);
                                INF
                                ====
                                \                N
(D4)                             >       A       X
                                /         N + 2
                                ====
                                N = - 2
```

DBLINT (*'F,'R,'S,a,b*) Fonction

une routine d'intégration double qui a été écrite en macsyma de haut niveau puis traduite et compilée en code machine.

Utilisez LOAD(DBLINT); pour avoir accès à ce package. Elle utilise la règle de Simpson dans les deux directions x et y pour calculer /B /S(X) | | | | F(X,Y) DY DX . | | /A /R(X) La fonction F(X,Y) doit être une fonction traduite ou compilée de deux variables, et R(X) et S(X) doivent être chacune une fonction traduite ou compilée d'une seule variable, tandis que a et b doivent être des nombres en virgule flottante. La routine a deux variables globales qui déterminent le nombre de divisions des intervalles x et y: DBLINT_X et DBLINT_Y sont toutes deux initialisées à 10, et peuvent être changées indépendamment en d'autres valeurs entières (il y a 2*DBLINT_X+1 points calculés dans la direction x, et 2*DBLINT_Y+1 dans la direction y). La routine subdivise l'axe des X puis pour chaque valeur de X elle calcule d'abord R(X) et S(X); ensuite l'axe des Y entre R(X) et S(X) est subdivisé et l'intégrale le long de l'axe des Y est exécutée selon la règle de Simpson; ensuite l'intégrale le long de l'axe des X est calculée par la règle de Simpson avec les Y-intégrales comme valeurs de la fonction. Cette procédure peut être numériquement instable pour une grande variété de raisons, mais elle est raisonnablement rapide : évitez de l'utiliser sur des fonctions rapidement oscillantes ou ayant des singularités (pôles ou points de rebroussement dans la région). Les intégrales Y dépendent de la distance entre R(X) et S(X), ainsi si S(X)-R(X) varie rapidement avec X, il peut se produire des erreurs substantielles provenant de la troncature des pas différents des diverses intégrales Y. On peut augmenter DBLINT_X et DBLINT_Y en un effort pour améliorer la couverture de la région, au dépens du temps de calcul. Les valeurs de la fonction ne sont pas conservées, donc si la fonction consomme beaucoup de temps, vous devrez attendre pour le re-calcul si vous changez quelque chose (désolé). Les fonctions F, R, et S doivent être traduites ou compilées avant d'appeler DBLINT. Cela donnera dans bien des cas une amélioration d'un ordre de grandeur de la vitesse par rapport au code interprété ! Demandez à LPH (ou à GJC) pour l'utilisation de ces aides numériques. Le fichier SHARE1;DBLINT DEMO peut être lancé en mode batch ou démo pour illustrer l'utilisation à l'aide

d'un problème d'exemple ; le fichier SHARE1;DBLNT DEMO1 est une extension de DEMO qui utilise aussi les autres aides numériques, FLOATDEFUNK et QUANC8. Envoyez toutes les notes de bogues et les questions à LPH.

DEFINT (*exp, var, low, high*) Fonction

DEFinite INTegration, idem INTEGRATE(exp,var,low,high). Elle utilise des méthodes symboliques, si vous voulez utiliser une méthode numérique essayez ROMBERG(exp,var,low,high).

ERF (*X*) Fonction

la fonction d'erreur, dont la dérivée est 2*EXP(-X^2)/SQRT(%PI).

ERFFLAG Variable

par défaut: [TRUE] - si FALSE empêche RISCH d'introduire la fonction ERF dans la réponse s'il n'y en avait pas au départ dans l'intégrande.

ERRINTSCE Variable

par défaut: [TRUE] - si un appel à la routine INTSCE n'est pas de la forme

```
EXP(A*X+B)*COS(C*X)^N*SIN(C*X)
```

alors le programme régulier d'intégration sera appelé si le commutateur ERRINTSCE[TRUE] est TRUE. S'il est FALSE alors INTSCE ne sera pas définie.

ILT (*exp, lvar, ovar*) Fonction

prend la transformée inverse de Laplace de exp par rapport à lvar et au paramètre ovar. exp doit être un rapport de polynômes dont le dénominateur n'a que des facteurs linéaires ou quadratiques. En utilisant ensemble LAPLACE et ILT avec les fonctions SOLVE ou LINSOLVE l'utilisateur peut résoudre une unique équation différentielle ou intégrale de convolution, ou un ensemble d'entre elles.

```
(C1) 'INTEGRATE(SINH(A*X)*F(T-X),X,0,T)+B*F(T)=T**2;
                T
               /
               [                                      2
(D1)           I (SINH(A X) F(T - X)) dX + B F(T) = T
               ]
               /
               0
(C2) LAPLACE(%,T,S);
              A LAPLACE(F(T), T, S)
(D2)          ---------------------
                       2    2
                      S  - A
                                                 2
                   + B LAPLACE(F(T), T, S) = --
                                                  3
                                                 S
(C3) LINSOLVE([%],['LAPLACE(F(T),T,S)]);
SOLUTION
```

```
                                                 2      2
                                              2 S  - 2 A
(E3)       LAPLACE(F(T), T, S) = --------------------
                                                5          2     3
                                             B S  + (A - A  B) S
(D3)                           [E3]
(C4) ILT(E3,S,T);
IS  A B (A B - 1)  POSITIVE, NEGATIVE, OR ZERO?
POS;
                                                  2
                                SQRT(A) SQRT(A B  - B) T
                          2 COSH(-----------------------)
                                                B
(D4)  F(T) =  - --------------------------------
                                           A
                        2
                       A T                2
                    + ------- + ------------------
                      A B - 1    3  2      2
                                A  B  - 2 A  B + A
```

INTEGRATE (*exp, var*) Fonction

intègre exp par rapport à var ou retourne une expression intégrale (la forme nominale) si elle ne peut accomplir l'intégration (voir la note 1 ci-dessous). Trois étapes sont en gros utilisées :

- (1) INTEGRATE regarde si l'intégrande est de la forme F(G(X))*DIFF(G(X),X) en vérifiant que la dérivée d'une sous-expression (i.e. G(X) dans le cas ci-dessus) divise l'intégrande. Si c'est le cas elle cherche F dans une table des intégrales et remplace G(X) par X dans l'intégrale de F. Ceci peut amener l'utilisation de gradients en prenant la dérivée (si une fonction inconnue apparaît dans l'intégrande elle doit être éliminée dès cette étape sinon INTEGRATE retournera la forme nominale de l'intégrande).
- (2) INTEGRATE essaie de comparer l'intégrande à une forme pour laquelle une méthode spécifique peut être utilisée, p. ex. des substitutions trigonométriques.
- (3) Si les deux premières étapes échouent elle utilise l'algorithme de Risch.

  Les relations fonctionnelles doivent être explicitement représentées afin qu'INTEGRATE fonctionne proprement.

  INTEGRATE n'est pas affectée par les DEPENDENCIES définies par la commande DEPENDS.

  INTEGRATE(exp, var, low, high) calcule l'intégrale définie de exp par rapport à var de low vers high, ou retourne la forme nominale si elle ne peut accomplir l'intégration. Les limites ne devront pas contenir var. Plusieurs méthodes sont utilisées, y compris la substitution directe dans l'intégrale indéfinie et l'intégration par parties. Des intégrales (impropres ??? Improper) peuvent utiliser les noms INF pour l'infini positif et MINF pour l'infini négatif. Si une

"forme" intégrale est désirée pour la manipulation (par exemple, une intégrale qui ne peut être calculée tant que certains nombres n'ont pas remplacés certains paramètres), la forme nominale 'INTEGRATE peut être utilisée et elle sera affichée avec le signe de l'intégrale (voir la note 1 ci-dessous).

La fonction LDEFINT utilise LIMIT pour évaluer l'intégrale aux limites inférieure et supérieure.

Quelquefois pendant l'intégration il peut être demandé à l'utilisateur le signe d'une expression. Les réponses convenables sont POS;, ZERO;, ou NEG;.

```
(C1) INTEGRATE(SIN(X)**3,X);
                     3
                  COS (X)
(D1)              ------- - COS(X)
                     3
(C2) INTEGRATE(X**A/(X+1)**(5/2),X,0,INF);
IS  A + 1  POSITIVE, NEGATIVE, OR ZERO?
POS;
IS  2 A - 3  POSITIVE, NEGATIVE, OR ZERO?
NEG;
                              3
(D2)             BETA(A + 1, - - A)
                              2
(C3) GRADEF(Q(X),SIN(X**2));
(D3)                                Q(X)
(C4) DIFF(LOG(Q(R(X))),X);
                             d              2
                            (-- R(X)) SIN(R (X))
                             dX
(D4)                        --------------------
                                  Q(R(X))
(C5) INTEGRATE(%,X);
(D5)                           LOG(Q(R(X)))
```

Note 1 : le fait que MACSYMA ne calcule pas certaines intégrales n'implique pas toujours que l'intégrale n'existe pas sous forme fermée. Dans l'exemple ci-dessous l'appel à l'intégration retourne la forme nominale alors que l'intégrale peut être trouvée assez facilement. Par exemple, on peut calculer les racines de `X^3+X+1 = 0` pour ré-écrire l'intégrande sous la forme

```
1/((X-A)*(X-B)*(X-C))
```

où A, B et C sont les racines. MACSYMA intégrera cette forme équivalente bien que l'intégrale soit assez compliquée.

```
(C6) INTEGRATE(1/(X^3+X+1),X);
                            /
                            [     1
(D6)                        I ---------- dX
                            ]  3
                            / X  + X + 1
```

INTEGRATION_CONSTANT_COUNTER Variable

un compteur qui est mis à jour chaque fois qu'une constante d'intégration (appelée par MACSYMA, p. ex., "INTEGRATIONCONSTANT1") est introduite dans une expression par intégration indéfinie d'une équation.

INTEGRATE_USE_ROOTSOF Variable

par défaut: [false] - si non fausse alors le dénominateur d'une fonction rationnelle ne peut être factorisée, nous donnons l'intégrale sous une forme qui est une somme sur les racines du dénominateur:

```
(C4) integrate(1/(1+x+x^5),x);

      /  2
      [ x  - 4 x + 5
      I ------------ dx                              2 x + 1
      ]  3    2                2             5 ATAN(-------)
      / x  - x  + 1        LOG(x  + x + 1)          SQRT(3)
(D4)  ----------------- - --------------- + ---------------
               7                 14              7 SQRT(3)
```

mais nous mettons maintenant le drapeau à vrai et la première partie de l'intégrale va subir une autre simplification.

```
(C5) INTEGRATE_USE_ROOTSOF:true;

(D5)                           TRUE
(C6) integrate(1/(1+x+x^5),x);

      ====         2
      \       (%R1  - 4 %R1 + 5) LOG(x - %R1)
       >      ------------------------------
      /                     2
      ====             3 %R1  - 2 %R1
                       3    2
      %R1 in ROOTSOF(x  - x  + 1)
(D6) ------------------------------------------------------------
               7

                                                      2 x + 1
                                           2       5 ATAN(-------)
                                      LOG(x  + x + 1)    SQRT(3)
                                    - --------------- + ---------
                                           14             7 SQRT(3)
```

Notez qu'il est possible que nous voulions une approximation des racines dans le plan complexe, et donc que nous fournissions la fonction factorisée, puisque nous pourrons alors grouper les racines et leurs conjuguées complexes, afin d'obtenir une meilleure réponse.

INTSCE (*expr,var*) Fonction

INTSCE LISP contient une routine, écrite par Richard Bogen, pour l'intégration de produits de sinus,cosinus et exponentielles de la forme

```
EXP(A*X+B)*COS(C*X)^N*SIN(C*X)^M
```

L'appel est INTSCE(expr,var) où expr peut être une expression quelconque, mais si elle n'est pas de la forme ci-dessus le programme régulier d'intégration sera invoqué si le commutateur ERRINTSCE[TRUE] est TRUE. S'il est FALSE INTSCE ne sera pas défini.

LDEFINT (*exp,var,ll,ul*) Fonction

donne l'intégrale définie de exp en utilisant LIMIT pour évaluer l'intégrale indéfinie de exp par rapport à var à la limite supérieure ul et à la limite inférieure ll.

POTENTIAL (*givengradient*) Fonction

Le calcul utilise la variable globale

```
POTENTIALZEROLOC[0]
```

qui doit être NONLIST ou de la forme

```
[indeterminatej=expressionj, indeterminatek=expressionk, ...]
```

la première étant équivalente à l'expression non liste pour tous les membres droits de la dernière. Les membres droits indiqués sont utilisés comme limite inférieure de l'intégration. Le succès de l'intégrations peut dépendre de leurs valeurs et de leur ordre. POTENTIALZEROLOC vaut initialement 0.

QQ Fonction

Le fichier SHARE1;QQ FASL (qui peut être chargé par LOAD("QQ");) contient une fonction QUANC8 qui peut prendre soit 3 soit 4 arguments. La version à 3 arguments calcule l'intégrale de la fonction spécifiée en premier argument sur l'intervalle de lo à hi comme dans QUANC8('nom fonction,lo,hi); . Le nom de la fonction doit être "apostrophé". La version à 4 arguments calculera l'intégrale de la fonction ou de l'expression (premier argument) par rapport à la variable (second argument) sur l'intervalle de lo à hi comme dans QUANC8(<f(x) ou expression en x>,x,lo,hi).

La méthode est celle de la quadrature polynômiale du 8ème ordre de Newton-Cotes, et la routine est adaptive. Elle passe ainsi son temps à diviser l'intervalle si nécessaire pour atteindre les conditions d'erreur spécifiées par les variables globales QUANC8_RELERR (valeur par défaut = 1.0e-4) et QUANC8_ABSERR (valeur par défaut = 1.0e-8) qui donne le test d'erreur relative : |intégrale(fonction)-valeur calculée|< quanc8_relerr*|intégrale(fonction)| et le test de l'erreur absolue: |intégrale(fonction)-valeur calculée|<quanc8_abserr.

Faites PRINTFILE(QQ,USAGE,SHARE1) pour les détails.

QUANC8 (*'nom fonction,lo,hi*) Fonction

Un intégrateur adaptif disponible dans SHARE1;QQ FASL. Les fichiers DEMO et USAGE sont fournis. La méthode consiste à utiliser la règle de la quadrature à 8 panneaux de Newton-Cotes, d'où le nom QUANC8 de la fonction, disponible en versions à 3 ou 4 arguments. Les tests d'erreur absolue et relative sont effectués. Pour l'utiliser faites LOAD("QQ"); . Pour plus de détails faites DESCRIBE(QQ); .

RESIDUE (*exp, var, val*) Fonction

calcule le résidu dans le plan complexe de l'expression exp lorsque la variable var prend la valeur val. Le résidu est le coefficient de (var-val)**(-1) dans la suite de Laurent pour exp.

```
(C1) RESIDUE(S/(S**2+A**2),S,A*%I);
                          1
(D1)                      -
                          2
(C2) RESIDUE(SIN(A*X)/X**4,X,0);
                          3
                         A
(D2)                   - --
                         6
```

RISCH (*exp, var*) Fonction

intègre exp par rapport à var en utilisant le cas transcendant de l'algorithme de Risch (le cas algébrique de cet algorithme n'a pas été implanté). Ceci traite actuellement les cas des exponentielles et logarithmes imbriqués que la partie principale d'INTEGRATE ne peut pas faire. INTEGRATE appliquera automatiquement RISCH si ces cas sont donnés.

ERFFLAG[TRUE] - si FALSE empêche RISCH d'introduire la fonction ERF dans la réponse s'il n'y en a pas déjà dans l'intégrande.

```
(C1) RISCH(X^2*ERF(X),X);
             2     2
           - X     X             3           2
         %E     (%E   SQRT(%PI) X  ERF(X) + X  + 1)
(D1)     ---------------------------------------
                        3 SQRT(%PI)
(C2) DIFF(%,X),RATSIMP;
                                  2
(D2)                             X  ERF(X)
```

ROMBERG (*exp,var,ll,ul*) Fonction

ou ROMBERG(exp,ll,ul) - Intégration de Romberg. Vous n'avez pas besoin de charger un fichier pour utiliser ROMBERG, elle est autochargée. Il y a deux façons d'utiliser cette fonction. La première est inefficace, comme la version d'intégrale définie de INTEGRATE : ROMBERG(<intégrande>,<variable d'intégration>,<limite inférieure>, <limite supérieure>);

```
Exemples:
        ROMBERG(SIN(Y),Y,1,%PI);
                TIME= 39 MSEC.          1.5403023
        F(X):=1/(X^5+X+1);
        ROMBERG(F(X),X,1.5,0);
```

```
                    TIME= 162 MSEC.          - 0.75293843
```

La seconde, efficace, est utilisée comme suit :

```
ROMBERG(<nom fonction>,<limite inférieure>,<limite supérieure>);
Exemple:
F(X):=(MODE_DECLARE([FONCTION(F),X],FLOAT),1/(X^5+X+1));
TRANSLATE(F);
ROMBERG(F,1.5,0);
        TIME= 13 MSEC.           - 0.75293843
```

Le premier argument doit être une fonction traduite (par TRANSLATE) ou compilée (si elle est compilée elle doit être déclarée comme retournant un FLONUM). Si le premier argument n'est pas déjà traduit, ROMBERG n'essaiera pas d'appeler TRANSLATE mais renverra une erreur.

La précision de l'intégration dépend des variables globales ROMBERGTOL (valeur par défaut 1.E-4) et ROMBERGIT (valeur par défaut 11). ROMBERG retournera un résultat si la différence relative des approximations successives est moindre que ROMBERGTOL. Elle essaiera de réduire de moitié la taille du pas ROMBERGIT fois avant d'abandonner. Le nombre d'itérations et d'évaluations de fonction que fera ROMBERG dépend de ROMBERGABS et ROMBERGMIN, faites DESCRIBE(ROMBERGABS,ROMBERGMIN); pour les détails.

ROMBERG peut être appelée récursivement et ainsi doubler ou tripler les intégrales.

```
Exemple:
INTEGRATE(INTEGRATE(X*Y/(X+Y),Y,0,X/2),X,1,3);
                          13/3 (2 LOG(2/3) + 1)
%,NUMER;
                          0.81930233
DEFINE_VARIABLE(X,0.0,FLOAT,"Global variable in fonction F")$
F(Y):=(MODE_DECLARE(Y,FLOAT), X*Y/(X+Y) )$
G(X):=ROMBERG('F,0,X/2)$
ROMBERG(G,1,3);
                           0.8193023
```

L'avantage avec cette méthode est que la fonction F peut être utilisée pour d'autres buts, comme le tracé de courbe. L'inconvénient est que vous devez penser à donner un nom à la fonction F et à sa variable indépendante X. Ou, sans la variable globale :

```
G1(X):=(MODE_DECLARE(X,FLOAT), ROMBERG(X*Y/(X+Y),Y,0,X/2))$
ROMBERG(G1,1,3);
                                0.8193023
```

L'avantage est ici un peu mince.

```
Q(A,B):=ROMBERG(ROMBERG(X*Y/(X+Y),Y,0,X/2),X,A,B)$
Q(1,3);
                                0.8193023
```

Il l'est encore plus de cette façon, et les variables n'ont pas à être déclarées car elles sont dans le contexte de ROMBERG.

L'utilisation de ROMBERG pour les intégrales multiples peut avoir cependant de grands désavantages. La quantité de calculs supplémentaires nécessités à cause de l'information géométrique ignorée en exprimant les intégrales multiples de cette façon

peut être incroyable. L'utilisateur devra être sûr de comprendre et d'utiliser les commutateurs ROMBERGTOL et ROMBERGIT.

La version IMSL de l'intégration de Romberg est maintenant disponible dans Macsyma. Faites DESCRIBE(DCADRE); pour plus d'information.

ROMBERGABS Variable

par défaut: [0.0] (0.0B0) - En supposant que les estimations successives produites par ROMBERG sont Y[0], Y[1], Y[2] etc., alors ROMBERG s'arrêtera après N itérations si (en gros) (ABS(Y[N]-Y[N-1]) <= ROMBERGABS OR ABS(Y[N]-Y[N-1])/(IF Y[N]=0.0 THEN 1.0 ELSE Y[N]) <= ROMBERGTOL) est TRUE (la condition sur le nombre d'itérations donné par ROMBERGMIN devra aussi être satisfaite).

Ainsi si ROMBERGABS est 0.0 (par défaut) vous obtenez juste le test d'erreur relative. L'utilité d'une variable additionnelle se présente lorsque vous voulez faire une intégration, et où la contribution dominante provient d'une petite région. Alors vous pouvez faire l'intégration d'abord sur cette petite région dominante, en utilisant le test de précision relative, suivie de l'intégration sur le reste de la région avec le test de précision absolue. Exemple : supposez que vous vouliez calculer (numériquement)

```
Integral(exp(-x),x,0,50)
```

avec une précision relative de 1 partie sur 10,000,000. Définissez la fonction. N est un compteur, ainsi nous pouvons voir combien d'évaluations de la fonction seront nécessaires.

```
F(X):=(MODE_DECLARE(N,INTEGER,X,FLOAT),N:N+1,EXP(-X))$
TRANSLATE(F)$
  /* Essayer tout d'abord de faire toute l'intégrale d'un coup */
BLOCK([ROMBERGTOL:1.E-6,ROMBERABS:0.],N:0,ROMBERG(F,0,50));
                ==> 1.00000003
N; ==> 257  /* Nombre d'évaluations de la fonction*/
```

Faisons maintenant l'intégration intelligemment, en faisant en premier Integral(exp(-x),x,0,10) puis en fixant ROMBERGABS à 1.E-6*(cette intégrale partielle).

```
BLOCK([ROMBERGTOL:1.E-6,ROMBERGABS:0.,SUM:0.],
  N:0,SUM:ROMBERG(F,0,10),ROMBERGABS:SUM*ROMBERGTOL,ROMBERGTOL:0.,
      SUM+ROMBERG(F,10,50));  ==> 1.00000001  /* Comme avant */
N;  ==> 130
```

Ainsi si F(X) était une fonction longue à calculer, la seconde méthode serait environ deux fois plus rapide.

ROMBERGIT Variable

par défaut: [11] - La précision de la commande d'intégration de ROMBERG dépend des variables globales ROMBERGTOL[1.E-4] et ROMBERGIT[11]. ROMBERG renverra un résultat si la différence relative des approximations successives est inférieure à ROMBERGTOL. Elle essaiera de diviser par deux la taille du pas ROMBERGIT fois avant d'abandonner.

ROMBERGMIN Variable

par défaut: [0] - dirige le nombre minimal d'évaluations de fonction que ROMBERG fera. ROMBERG évaluera d'abord son premier argument, au moins

2^(ROMBERGMIN+2)+1 fois. C'est utile pour l'intégration des fonctions oscillantes, lorsque le test de convergence normale peut quelquefois passer par erreur.

ROMBERGTOL Variable

par défaut: [1.E-4] - La précision de la commande d'intégration de ROMBERG dépend des variables globales ROMBERGTOL[1.E-4] et ROMBERGIT[11]. ROMBERG renverra un résultat si la différence relative des approximations successives est inférieure à ROMBERGTOL. Elle essaiera de diviser par deux la taille du pas ROMBERGIT fois avant d'abandonner.

TLDEFINT (*exp,var,ll,ul*) Fonction

est juste LDEFINT avec TLIMSWITCH mise à TRUE.

20 Équations

20.1 Définitions pour les équations

%RNUM_LIST Variable

par défaut: [] - Lorsque %R variables sont introduites dans les solutions par la commande ALGSYS, elles sont ajoutées à la %RNUM_LIST dans l'ordre où elles ont été créées. C'est pratique pour faire par la suite des substitutions dans la solution. On recommande d'utiliser cette liste plutôt que de faire CONCAT('%R,J).

ALGEXACT Variable

par défaut: [FALSE] - affecte le comportement de ALGSYS comme ceci : si ALGEXACT est TRUE, ALGSYS appellera toujours SOLVE puis utilisera REALROOTS sur les échecs de SOLVE. Si ALGEXACT est FALSE, SOLVE est appelée seulement si l'éliminant n'est pas univariant, ou s'il est quadratique ou biquadratique. Ainsi ALGEXACT:TRUE ne garantit pas uniquement des solutions exactes, mais seulement que ALGSYS fera de son mieux pour fournir des solutions exactes, et ne donnera que des approximations lorsque tout le reste échouera.

ALGSYS ([*exp1, exp2, ...*], [*var1, var2, ...*]) Fonction

résoud la liste des polynômes simultanés ou des équations polynômiales (qui peuvent être non linéaires) sur la liste des variables. Les symboles %R1, %R2, etc. seront utilisés pour représenter des paramètres arbitraires nécessaires à la solution (la variable %RNUM_LISTE les contient). Lors du processus décrit ci-dessous, ALGSYS s'exécute au besoin récursivement.

La méthode est la suivante :

(1) Tout d'abord les équations sont FACTORisées et découpées en sous-systèmes.

(2) Pour chaque sous-système Si, une équation E et une variable var sont sélectionnées (la var est choisie de façon à avoir le plus petit degré non zéro). Puis le résultant de E et Ej par rapport à var est calculé pour chacune des équations restantes Ej du sous-système Si. Ceci donne un nouveau sous-système S'i avec une variable en moins (var a été éliminée). Le processus recommence maintenant en (1).

(3) éventuellement, un sous-système consistant en une unique équation est obtenu. Si l'équation est multivariable et qu'aucune approximation sous forme de nombres en virgule flottante n'a été introduite, alors SOLVE est appelée pour trouver la solution exacte (l'utilisateur doit réaliser que SOLVE peut ne pas être capable de donner une solution ou si elle le fait que la solution peut être une très grande expression).

Si l'équation est univariable et est ou linéaire, ou quadratique, ou bi-quadratique, alors à nouveau SOLVE est appelée si aucune approximation n'a été introduite. Si des approximations ont été utilisées ou que l'équation n'est pas univariable ni linéaire, quadratique, ou bi-quadratique, alors si le commutateur REALONLY[FALSE] est TRUE, la fonction REALROOTS est appelée pour trouver les solutions à valeurs

réelles. Si REALONLY:FALSE alors ALLROOTS est appelée qui recherche les solutions réelles et complexes. Si ALGSYS produit une solution qui a moins de chiffres significatifs que requis, l'utilisateur peut donner à ALGEPSILON[10^8] une plus grande valeur. Si ALGEXACT[FALSE] est mise à TRUE, SOLVE sera toujours appelée.

(4) Finalement, les solutions obtenues au pas (3) sont ré-insérées dans les niveaux précédents et le processus recommence en (1).

L'utilisateur devra être conscient que :

Lorsque ALGSYS rencontre une équation multivariable qui contient des approximations en virgule flottante (en général dûes au fait que des solutions exactes n'ont pas été trouvées à une étape précédente), alors la fonction n'essaie pas d'appliquer des méthodes exactes à de telles équations et affiche à la place le message:

"ALGSYS cannot solve - system too complicated." ("ALGSYS ne peut résoudre - système trop compliqué")

Les interactions avec RADCAN peuvent produire des expressions grandes ou compliquées. Dans ce cas, l'utilisateur peut utiliser PICKAPART ou REVEAL pour analyser la solution. Occasionnellement, RADCAN peut introduire un %I apparent dans une solution qui est en fait une valeur réelle.

Faites EXAMPLE(ALGSYS); pour voir des exemples.

ALLROOTS (*poly*) Fonction

cherche toutes les racines réelles et complexes du polynôme réel poly qui doit être univariable et peut être une équation, p. ex. poly=0. Pour les polynômes complexes un algorithme par Jenkins et Traub est utilisé (Algorithm 419, Comm. ACM, vol. 15, (1972), p. 97). Pour les polynômes réels l'algorithme utilisé est dû à Jenkins (Algorithm 493, TOMS, vol. 1, (1975), p.178). Le drapeau POLYFACTOR[FALSE] si vrai provoque la factorisation par ALLROOTS du polynôme sur les nombres réels si le polynôme est réel, ou sur les nombres complexes, si le polynôme est complexe. ALLROOTS peut donner des résultats imprécis en cas de racines multiples (si poly est réel et que vous obtenez des réponses imprécises, essayez ALLROOTS(%I*poly);). Faites EXAMPLE(ALLROOTS); pour un exemple.

ALLROOTS rejette les non polynômes. Elle requiert que le numérateur après utilisation de RAT soit un polynôme, et que le dénominateur soit au plus un nombre complexe. ALLROOTS retournera donc toujours une expression équivalente (mais factorisée), si POLYFACTOR est TRUE.

BACKSUBST Variable

par défaut: [TRUE] - si mise à FALSE empêchera les substitution arrières après que les équations aient été triangulées. Ceci peut être nécessaire pour de très gros problèmes dans lesquels des substitution arrières provoqueraient la génération d'extrèmement grandes expressions (sur MC ceci excéderait la capacité de stockage).

BREAKUP Variable

par défaut: [TRUE] - si FALSE SOLVE exprimera les solutions d'équations cubiques ou quartiques comme simples expressions plutôt que constituées de plusieurs sous-expressions communes qui est le défaut. BREAKUP:TRUE ne fonctionne que si PROGRAMMODE est FALSE.

DIMENSION (*équation ou liste d'équations*) Fonction

Le fichier "share1/dimen.mc" contient des fonctions d'analyse dimensionnelle automatique. LOAD(DIMEN); les chargera pour vous. Une démonstration est disponible dans share1/dimen.dem. Faites DEMO("dimen"); pour la lancer.

DISPFLAG Variable

par défaut: [TRUE] - si FALSE dans un BLOCK supprime l'affichage de la sortie générée par les fonctions solve appelées dans le BLOCK. Terminer le BLOCK par un signe dollar, $, fixe DISPFLAG à FALSE.

FUNCSOLVE (*eqn,g(t)*) Fonction

donne [g(t) = ...] ou [], selon qu'il existe ou non une fonction rationnelle g(t) satisfaisant eqn, qui doit être (dans ce cas) un polynôme du premier ordre, linéaire en g(t) et g(t+1).

```
(C1) FUNCSOLVE((N+1)*FOO(N)-(N+3)*FOO(N+1)/(N+1) =
    (N-1)/(N+2),FOO(N));
                                        N
(D1)                FOO(N) = ---------------
                                 (N + 1) (N + 2)
```

Attention: c'est une implantation très rudimentaire – beaucoup de tests de sécurité et de généralisations évidentes manquent.

GLOBALSOLVE Variable

par défaut: [FALSE] - si TRUE les variables qui sont résolues par SOLVE seront affectées à la solution de l'ensemble des équations simultanées.

IEQN (*ie,unk,tech,n,guess*) Fonction

Routine de résolution d'une équation intégrale. Faites LOAD(INTEQN); pour y accéder.

Attention : pour libérer de l'espace de stockage, un KILL(LABELS) est inclus dans ce fichier. Par conséquent, avant de charger le package équation intégrale, l'utilisateur devra nommer toutes les expressions qu'il veut garder.

ie est l'équation intégrale, nk est la fonction inconnue, tech est la technique à essayer parmis celles données ci-dessus (tech = FIRST signifie : essayer la première technique qui trouve une solution; tech = ALL signifie : essayer toutes les techniques applicables); n est le nombre maximum de termes à prendre pour TAYLOR, NEUMANN, FIRSTKINDSERIES, ou FREDSERIES (il est aussi la profondeur maximale de récursion pour la méthode de différentiation); guess est l'estimation initiale pour NEUMANN ou FIRSTKINDSERIES. Les valeurs par défaut pour les 2ème au 5ème paramètres sont : unk: P(X), où P est la première fonction rencontrée dans un intégrande inconnu de MACSYMA et X est la variable qui survient comme argument de la première occurrence de P trouvée hors d'une intégrale dans le cas d'équations SECONDKIND, ou est la seule autre variable en dehors de la variable d'intégration des équations FIRSTKIND. Si la tentative pour trouver X échoue, il sera demandé à l'utilisateur de fournir la variable indépendante ; tech: FIRST; n: 1; guess: NONE, oblige NEUMANN et FIRSTKINDSERIES à utiliser F(X) comme estimation initiale.

IEQNPRINT Variable

par défaut: [TRUE] - gère le comportement du résultat retourné par la commande IEQN (which see ???). Si IEQNPRINT est mise à FALSE, les listes retournées par la fonction IEQN sont de la forme [SOLUTION, TECHNIQUE USED, NTERMS, FLAG] où FLAG est absent si la solution est exacte. Sinon, c'est le mot APPROXIMATE ou INCOMPLETE correspondant à une forme de solution inexacte ou non fermée, respectivement. Si une méthode de suites a été utilisée, NTERMS donne le nombre de termes pris (qui peut être moins que les n donnés à IEQN si une erreur à empêché la génération de termes ultérieurs).

LHS (*eqn*) Fonction

le membre gauche de l'équation eqn.

LINSOLVE ([*exp1, exp2, ...*], [*var1, var2, ...*]) Fonction

résoud la liste des équations linéaires simultanées par rapport à la liste des variables. Les expi doivent être des polynômes des variables et peuvent être des équations. Si GLOBALSOLVE[FALSE] est mise à TRUE alors les variables qui auront été résolues par SOLVE seront affectées à la solution de l'ensemble des équations simultanées.

BACKSUBST[TRUE] si mise à FALSE empêchera la substitution arrière après que les équations aient été triangulées. Ce peut être nécessaire pour de très gros problèmes où la substitution arrière provoquerait la génération de très grandes expressions (sous MC ceci excéderait la capacité de stockage).

LINSOLVE_PARAMS[TRUE] - si TRUE, LINSOLVE génère aussi les symboles %Ri utilisés pour représenter les paramètres arbitraires décrit dans le manuel sous ALGSYS. Si FALSE, LINSOLVE se comporte comme précédemment, i.e. lorsqu'un système d'équations sous-déterminé lui est présenté, il les résoud pour certaines des variables en termes des autres.

```
(C1) X+Z=Y$
(C2) 2*A*X-Y=2*A**2$
(C3) Y-2*Z=2$
(C4) LINSOLVE([D1,D2,D3],[X,Y,Z]),GLOBALSOLVE:TRUE;
SOLUTION
(E4)                              X : A + 1
(E5)                               Y : 2 A
(E6)                              Z : A - 1
(D6)                            [E4, E5, E6]
```

LINSOLVEWARN Variable

par défaut: [TRUE] - si FALSE fait supprimer l'affichage du message "Les équations dépendantes sont éliminées".

LINSOLVE_PARAMS Variable

par défaut: [TRUE] - si TRUE, LINSOLVE génère aussi les symboles %Ri utilisés pour représenter les paramètres arbitraires décrit dans le manuel sous ALGSYS. Si FALSE,

LINSOLVE se comporte comme précédemment, i.e. lorsqu'un système d'équations sous-déterminé lui est présenté, il les résoud pour certaines des variables en termes des autres.

MULTIPLICITIES Variable

par défaut: [NOT_SET_YET] - sera défini comme une liste des multiples solutions individuelles retournées par SOLVE ou REALROOTS.

NROOTS (*poly, low, high*) Fonction

cherche le nombre de racines réelles du polynôme réel univariable poly dans l'intervalle demi-ouvert (low,high]. Les limites de l'intervalle peuvent aussi être respectivement MINF,INF pour moins l'infini et plus l'infini. La méthode des suites de Sturm est utilisée.

```
(C1) POLY1:X**10-2*X**4+1/2$
(C2) NROOTS(POLY1,-6,9.1);
RAT REPLACED 0.5 BY 1/2 = 0.5
(D2)                                4
```

NTHROOT (*p,n*) Fonction

où p est un polynôme à coefficients entiers et n est entier positif, retourne q, un polynôme sur les entiers, tel que q^n=p ou bien affiche un message d'erreur indiquant que p n'est pas une puissance nième parfaite. Cette routine est bien plus rapide que FACTOR ou même que SQFR.

PROGRAMMODE Variable

par défaut: [TRUE] - si FALSE, SOLVE, REALROOTS, ALLROOTS, et LINSOLVE afficheront des "étiquettes-E" (étiquettes de ligne intermédiaire) aux réponses étiquetées. Si TRUE, SOLVE, etc. retournent des réponses en tant qu'éléments d'une liste (sauf si BACKSUBST est FALSE, auquel cas PROGRAMMODE:FALSE est aussi utilisé).

REALONLY Variable

par défaut: [FALSE] - si TRUE ALGSYS retournera seulement les solutions indépendantes de %I.

REALROOTS (*poly, bound*) Fonction

cherche toutes les racines réelles du polynôme réel univariable poly avec une tolérance bornée bound laquelle, si inférieure à 1, provoque la recherche exacte de toutes les racines entières. Le paramètre bound peut être arbitrairement petit afin d'obtenir toute précision désirée. Le premier argument peut aussi être une équation. REALROOTS définit MULTIPLICITIES, utile en cas de racines multiples. REALROOTS(poly) est équivalente à REALROOTS(poly,ROOTSEPSILON). ROOTSEPSILON[1.0E-7] est un nombre réel utilisé pour établir l'intervalle de confiance des racines. Faites EXAMPLE(REALROOTS); pour voir un exemple.

RHS (*eqn*) Fonction

le membre droit de l'équation eqn.

ROOTSCONMODE Variable

par défaut: [TRUE] - Détermine le comportement de la commande ROOTSCONTRACT. Faites DESCRIBE(ROOTSCONTRACT); pour les détails.

ROOTSCONTRACT (*exp*) Fonction

convertit les produits de racines en racines de produits. Par exemple,

```
ROOTSCONTRACT(SQRT(X)*Y^(3/2)) ==> SQRT(X*Y^3)
```

Lorsque RADEXPAND est TRUE et DOMAIN est REAL (leurs défauts), ROOTSCONTRACT convertit ABS en SQRT, p. ex.

```
ROOTSCONTRACT(ABS(X)*SQRT(Y)) ==> SQRT(X^2*Y)
```

Il existe une option ROOTSCONMODE (valeur par défaut TRUE), affectant comme suit ROOTSCONTRACT :

Problème	Valeur de ROOTSCONMODE	Résultat de l'application de ROOTSCONTRACT
X^(1/2)*Y^(3/2)	FALSE	(X*Y^3)^(1/2)
X^(1/2)*Y^(1/4)	FALSE	X^(1/2)*Y^(1/4)
X^(1/2)*Y^(1/4)	TRUE	(X*Y^(1/2))^(1/2)
X^(1/2)*Y^(1/3)	TRUE	X^(1/2)*Y^(1/3)
X^(1/2)*Y^(1/4)	ALL	(X^2*Y)^(1/4)
X^(1/2)*Y^(1/3)	ALL	(X^3*Y^2)^(1/6)

Les exemples ci-dessus et quelques autres peuvent être essayés en tapant

```
EXAMPLE(ROOTSCONTRACT);
```

Lorsque ROOTSCONMODE est FALSE, ROOTSCONTRACT contracte seulement le nombre "wrt" rationel des exposants dont les dénominateurs sont les mêmes. La clé des exemples ROOTSCONMODE:TRUE$ ci-dessus est simplement que 2 divise 4 mais pas 3. ROOTSCONMODE:ALL$ implique de prendre le ppcm (plus petit commun multiple) des dénominateurs des exposants.

ROOTSCONTRACT utilise RATSIMP d'une manière semblable à LOGCONTRACT (voir le manuel).

ROOTSEPSILON Variable

par défaut: [1.0E-7] - un nombre réel utilisé pour établir l'intervalle de confiance des racines trouvées par la fonction REALROOTS.

SOLVE (*exp, var*) Fonction

résoud l'équation algébrique exp pour la variable var et retourne une liste des équations solutions en var. Si exp n'est pas une équation, elle est supposée être une expression devant être mise à zéro. Var peut être une fonction (p. ex. F(X)), ou une autre expression non atomique, mais pas une somme ou un produit. Elle peut être omise si exp ne contient qu'une seule variable. Exp peut être une expression

rationnelle, et peut contenir des fonctions trigonométriques, exponentielles, etc. La méthode suivante est utilisée : Soit E l'expression et X la variable. Si E est linéaire en X alors elle est résolue trivialement par rapport à X. Autrement, si E est de la forme A*X**N+B alors le résultat est (-B/A)**(1/N) fois les Nièmes racines de l'unité.

Si E n'est pas linéaire en X alors le pgcd des exposants de X en E (soit N) est divisé entre les exposants et la multiplicité des racines est multipliée par N. Puis SOLVE est appelée à nouveau sur le résultat.

Si E se factorise alors SOLVE est appelée sur chacun des facteurs. Finalement SOLVE utilisera au besoin les formules quadratiques, cubiques, ou quartiques.

Pour le cas où E est un polynôme d'une certaine fonction de la variable X, soit F(X), alors elle est d'abord résolue pour F(X) (appelons C le résultat), puis l'équation F(X)=C peut être résolue par rapport à X pourvu que l'inverse de la fonction F soit connu.

BREAKUP[TRUE] si FALSE fera exprimer par SOLVE les solutions des équations cubiques ou quartiques sous formes d'expressions uniques plutôt que comme plusieurs sous-expressions communes, qui est le défaut.

MULTIPLICITIES[NOT_SET_YET] - sera défini comme une liste des multiplicités des solutions individuelles retournées par SOLVE, REALROOTS, ou ALLROOTS.

Essayez APROPOS(SOLVE) pour les commutateurs qui affectent SOLVE. DESCRIBE peut alors être utilisé sur les noms des commutateurs individuels si leur finalité n'est pas claire.

SOLVE([eq1, ..., eqn], [v1, ..., vn]) résoud un système d'équations polynômiales simultanées (linéaires ou non linéaires) en appelant LINSOLVE ou ALGSYS et elle retourne une liste des solutions de la liste des variables. Dans le cas de LINSOLVE cette liste contiendra une liste unique de solutions. Elle prend deux listes comme arguments. La première liste (eqi, i=1,...,n) représente les équations à résoudre ; la seconde liste est celles des inconnues à déterminer. Si le nombre total de variables dans les équations est égal au nombre des équations, la liste en second argument peut être omise. Pour les systèmes linéaires, si les équations données ne sont pas compatibles, le message INCONSISTENT sera affiché (voir le commutateur SOLVE_INCONSISTENT_ERROR) ; s'il n'existe pas de solution unique, alors SINGULAR sera affichée. Pour les exemples, faites EXAMPLE(SOLVE);

SOLVEDECOMPOSES Variable

par défaut: [TRUE] - si TRUE, SOLVE utilisera POLYDECOMP (voir POLYDECOMP) dans sa tentative de résolution de polynômes.

SOLVEEXPLICIT Variable

par défaut: [FALSE] - si TRUE, empêche SOLVE de retourner des solutions implicites, i.e. de la forme F(x)=0.

SOLVEFACTORS Variable

par défaut: [TRUE] - si FALSE alors SOLVE n'essaiera pas de factoriser l'expression. Ce paramétrage peut être voulu dans les quelques cas où la factorisation n'est pas nécessaire

SOLVENULLWARN Variable

par défaut: [TRUE] - si TRUE l'utilisateur sera averti s'il appelle SOLVE avec une liste d'équation ou une liste de variable vide ("nulle"). Par example, SOLVE([],[]); affichera deux messages d'avertissement et retournera [].

SOLVERADCAN Variable

par défaut: [FALSE] - si TRUE SOLVE utilisera RADCAN qui ralentira SOLVE mais permettra la résolution de certains problèmes contenant des exponentiels et des logs.

SOLVETRIGWARN Variable

par défaut: [TRUE] - si mis à FALSE empêchera l'affichage par SOLVE de message d'avertissement disant qu'elle en train d'utiliser des fonctions trigonométriques inverses pour résoudre l'équation, perdant ainsi des solutions.

SOLVE_INCONSISTENT_ERROR Variable

par défaut: [TRUE] - si TRUE, SOLVE et LINSOLVE renvoient une erreur si elles rencontrent un ensemble d'équations linéaires inconsistentes, p. ex. SOLVE([A+B=1,A+B=2]). Si FALSE, elles retournent [] dans ce cas (c'est le nouveau mode, précédemment obtenu seulement par appel à ALGSYS).

ZRPOLY Fonction

La routine IMSL ZRPOLY cherche les zéros des polynômes simples (variable unique, coefficients réels, exposants entiers non négatifs), utilisant la technique de Jenkins-Traub. Pour l'utiliser, faites : LOADFILE("imsl");. La commande est POLYROOTS(polynôme);. Pour plus d'informations, faites PRINTFILE("zrpoly.usg");. Pour une démonstration, faites DEMO("zrpoly.dem");. Pour une information générale sur les packages MACSYMA-IMSL, PRINTFILE(IMSL,USAGE,SHARE2);.

ZSOLVE Fonction

Pour ceux qui peuvent utiliser des solutions numériques approchées des problèmes, il y a un package qui appelle une routine ayant été traduite de la bibliothèque IMSL fortran pour résoudre N équations simultanées non linéaires à N inconnues. Elle utilise des techniques de "boîtes noires" qui ne sont probablement pas désirables si une solution exacte peut être obtenue par l'un des résolveurs intelligents (LINSOLVE, ALGSYS, etc). Mais pour les choses que les autres résolveurs ne cherchent pas à traiter, ceci peut sans doute donner des résultats très utiles. Pour la documentation, faites PRINTFILE("zsolve.usg");. Pour une démonstration, faites batch("zsolve.mc")$.

21 Équation différentielles

21.1 Définitions pour les équations différentielles

DESOLVE ([*eq1,...,eqn*],[*var1,...,varn*]) Fonction

où les eq sont des équations différentielles des variables dépendantes var1,...,varn. Les relations fonctionnelles doivent être explicitement indiquées à la fois dans les équations et les variables. Par exemple

```
(C1) 'DIFF(F,X,2)=SIN(X)+'DIFF(G,X);
(C2) 'DIFF(F,X)+X^2-F=2*'DIFF(G,X,2);
n'est PAS le format convenable.  La façon correcte est :
(C3) 'DIFF(F(X),X,2)=SIN(X)+'DIFF(G(X),X);
(C4) 'DIFF(F(X),X)+X^2-F(X)=2*'DIFF(G(X),X,2);
L'appel est alors DESOLVE([D3,D4],[F(X),G(X)]);
Si les conditions initiales à 0 sont connues, elles devront être
fournies avant d'appeler DESOLVE en utilisant ATVALUE.
(C11) 'DIFF(F(X),X)='DIFF(G(X),X)+SIN(X);
                        d          d
(D11)                   -- F(X) = -- G(X) + SIN(X)
                        dX         dX
(C12) 'DIFF(G(X),X,2)='DIFF(F(X),X)-COS(X);
                         2
                        d           d
(D12)                   --- G(X) = -- F(X) - COS(X)
                          2         dX
                        dX
(C13) ATVALUE('DIFF(G(X),X),X=0,A);
(D13)                                 A
(C14) ATVALUE(F(X),X=0,1);
(D14)                                 1
(C15) DESOLVE([D11,D12],[F(X),G(X)]);
                 X                                X
(D16) [F(X)=A %E  - A+1, G(X) = COS(X) + A %E  - A + G(0) - 1]
/* VERIFICATION */
(C17) [D11,D12],D16,DIFF;
                    X       X      X                 X
(D17)         [A %E  = A %E , A %E  - COS(X) = A %E  - COS(X)]
```

Si DESOLVE ne peut obtenir une solution, elle retourne "FALSE".

IC1 (*exp,var,var*) Fonction

Afin de résoudre les problèmes à valeur initiale (IVP) et ceux à valeur bornée (BVP), la routine IC1 est proposée dans le package ODE2 pour les équations du premier ordre, et IC2 et BC2 pour le second ordre IVP et BVP, respectivement. Faites LOAD(ODE2) pour y accéder. Ils sont utilisés comme dans les exemples suivants :

```
(C3) IC1(D2,X=%PI,Y=0);
                            COS(X) + 1
(D3)                 Y = - ----------
                                 3
                                X
(C4) 'DIFF(Y,X,2) + Y*'DIFF(Y,X)^3 = 0;
                          2
                         d Y      dY 3
(D4)                     --- + Y (--)  = 0
                           2      dX
                         dX
(C5) ODE2(%,Y,X);
                   3
                  Y  - 6 %K1 Y - 6 X
(D7)              ------------------ = %K2
                           3
(C8) RATSIMP(IC2(D7,X=0,Y=0,'DIFF(Y,X)=2));
                        3
                    2 Y  - 3 Y + 6 X
(D9)              - ---------------- = 0
                             3
(C10) BC2(D7,X=0,Y=1,X=1,Y=3);
                   3
                  Y  - 10 Y - 6 X
(D11)             --------------- = - 3
                          3
```

ODE (*equation,y,x*) Fonction

un pot-pourri de résolveurs de Différentielles Ordinaires combinés de façon à essayer des méthodes de plus en plus difficiles après chaque échec. Par exemple, la première tentative se fait avec ODE2, par conséquent un utilisateur utilisant ODE peut être sûr qu'il a toutes les capacités de ODE2 dès le début et s'il a utilisé ODE2 dans des programmes ils tourneront encore s'il y substitue ODE (les valeurs retournées et les séquences d'appel sont identiques). De plus, ODE a nombre de particularités utilisateur qui peuvent aider un résolveur expérimenté si le système de base ne peut traiter l'équation. L'équation est de la même forme que requise par ODE2 (which see???) et les y et x sont des variables dépendantes et indépendantes, comme avec ODE2. Pour plus de détails, faites PRINTFILE(ODE,USAGE,SHARE); .

ODE2 (*exp,dvar,ivar*) Fonction

prend trois arguments : une ODE du premier ou second ordre (seul le membre gauche est donné si le membre droit est 0), la variable dépendante, et la variable indépendante. Si réussie, elle retourne une solution explicite ou implicite de la variable dépendante. %C est utilisée pour représenter la constante dans le cas des équations du premier ordre, et %K1 et %K2 les constantes pour les équations du second ordre. Si ODE2 ne peut obtenir une solution pour quelque raison que ce soit, elle retourne FALSE, après avoir peut-être affiché un message d'erreur. Les méthodes implantées

pour les équations du premier ordre, et l'ordre d'exécution sont : linéaire, séparée, exacte - peut-être requérant un facteur d'intégration, homogène, équation de Bernoulli, et une méthode homogène généralisée. Pour le second ordre : coefficient constant, exacte, homogène linéaire avec coefficients non constants qui peuvent être transformés en coefficients constants, l'équation d'Euler ou équidimensionnelle, la méthode de variation des paramètres, et les équations qui sont indépendantes soit de la variable indépendante soit de la variable dépendante de sorte qu'elles peuvent être réduites en deux équations linéaires du premier ordre pour être résolues séquentiellement. Au cours de la résolution d'ODE, plusieurs variables sont définies dans le seul but informationnel : METHOD dénote la méthode de résolution utilisée, p. ex. LINEAR, INTFACTOR dénote tout facteur d'intégration utilisé, ODEINDEX dénote l'indice de la méthode de Bernoulli ou de méthode homogène généralisée, et YP dénote la solution particulière pour la technique de la variation des paramètres.

22 Calcul numérique

22.1 Introduction au calcul numérique

22.2 DCADRE

Ce qui suit est obsolète. Pour faire une interface avec les bibliothèques Fortran dans le MAXIMA actuel regardez les exemples dans "maxima/src/fortdef.lsp"

La version IMSL de l'intégration de Romberg est maintenant disponible dans Macsyma. Pour la documentation, faites PRINTFILE(DCADRE,USAGE,IMSL1); . Pour une démonstration, faites batch("dcadre.mc");. C'est un package d'intégration numérique utilisant avec prudence une extrapolation adaptive de Romberg.

Le package DCADRE est écrit pour appeler la routine DCADRE de la bibliothèque IMSL du Fortran. C'est une documentation pour ce programme. Envoyez bogues/commentaires à KMP. Pour charger ce package, faites

```
LOADFILE("imsl")$
```

Pour une démonstration de ce package, faites

```
batch("dcadre.mc");
```

La fonction de travail a la syntaxe suivante : IMSL_ROMBERG(fn,low,hi) où fn est une fonction à 1 argument ; low et hi devront être les limites inférieure et supérieure de l'intégration. fn doit retourner des valeurs en virgule flottante.

IMSL_ROMBERG(exp,var,low,hi) où exp devra être intégrée de var=low à hi. Le résultat de l'évaluation de exp doit toujours être un nombre en virgule flottante.

FAST_IMSL_ROMBERG(fn,low,hi) Cette fonction ne fait pas de contrôle d'erreur mais procure un gain de vitesse sur la fonction IMSL_ROMBERG. Elle suppose que fn est une fonction Lisp (ou une fonction Macsyma traduite) qui accepte un argument en virgule flottante et qui retourne toujours une valeur en virgule flottante.

Elle retourne soit [SUCCESS, answer, error] où answer est le résultat de l'intégration et erreur est la borne estimée de l'erreur absolue sur la sortie, DCADRE, comme décrit dans PURPOSE ci-dessous. soit

[WARNING, n, answer, error] où n est un code d'avertissement, answer est la réponse, et erreur est la borne estimée de l'erreur absolue sur la sortie, DCADRE, comme décrit dans PURPOSE ci-dessous. Les avertissements suivants peuvent être donnés : 65 = une ou plusieurs singularités ont été traitées avec succès. 66 = dans certains sous-intervalle(s), l'estimation de l'intégrale a été acceptée uniquement parce que l'erreur estimée était petite, bien qu'aucun comportement régulier n'ait été reconnu. soit

[ERROR, errorcode] où errorcode est le code d'erreur généré par IMSL. Les codes d'erreur suivants peuvent se produire : 131 = échec dû à un espace de travail interne insuffisant. 132 = échec. Peut être dû à trop de bruit dans la fonction (relatif aux conditions d'erreur requises) ou dû à un mauvais comportement de l'intégrande. 133 = RERR est plus grand que 0.1 ou inférieur à 0.0 ou est trop petit pour la précision de la machine.

Les drapeaux suivants ont une influence sur le fonctionnement IMSL_ROMBERG –

ROMBERG_AERR [par défaut 1.0E-5] – erreur absolue désirée dans la réponse.

ROMBERG_RERR [par défaut 0.0] – erreur relative désirée dans la réponse.

Note: si IMSL signale une erreur, un message sera affiché sur la console de utilisateur donnant la nature de l'erreur. (Ce message d'erreur peut être supprimé en mettant IMSLVERBOSE à FALSE).

Note: comme ceci utilise une routine traduite de Fortran, elle peut ne pas être invoquée récursivement. Elle ne s'appelle pas elle-même, mais l'utilisateur doit savoir qu'il ne peut taper ^A au milieu d'un calcul IMSL_ROMBERG, commencer un autre acalcul avec le même package, et espérer gagner – IMSL_ROMBERG se plaindra si elle exécute déjà un projet lorsque vous l'appelez. Ceci ne devrait provoquer que des problèmes minimes.

Projet (version modifiée de la documentation IMSL) ————————————————————-

DCADRE tente de résoudre le problème suivant: étant donnée une fonction F à valeur réelle d'un argument, deux nombres réels A et B, trouver un nombre DCADRE tel que :

```
|   / B                 |        [                                  | / B       | ]
|   [                   |        [                                  | [         | ]
|   I F(x)dx - DCADRE   | <= max [ ROMBERG_AERR, ROMBERG_RERR * |   | I F(x)dx  | ]
|   ]                   |        [                                  | ]         | ]
|   / A                 |        [                                  | / A       | ]
```

Algorithme (version modifiée de la documentation IMSL)

Cette routine utilise un procédé où DCADRE est calculé comme somme des estimations de l' intégrale de F(x) sur des sous-intervalles convenablement choisis dans l'intervalle d'intégration donné. Commençant avec cet intervalle d'intégration lui-même comme premier sous-intervalle, une prudente extrapolation de Romberg est utilisée pour trouver une estimation acceptable dans un sous-intervalle donné. Si cette tentative échoue, le sous-intervalle est divisé en deux sous-intervalles d'égale longueur, chacun d'eux étant considéré séparément.

Notes de programmation (version modifiée de la documentation IMSL)

- 1. DCADRE (la base traduite du Fortran pour IMSL_ROMBERG) peut, dans bien des cas, traiter les sauts de discontinuités et certaines discontinuités algébriques. Voyez la référence pour tous les détails.
- 2. Le paramètre d'erreur relative ROMBERG_RERR doit être dans l'intervalle [0.0,0.1]. Par exemple, ROMBERG_RERR=0.1 indique que l'estimation de l'intégrale doit être exacte à un chiffre, alors que ROMBERG_RERR=1.0E-4 demande une précision à quatre chiffres. Si DCADRE détermine que la précision relative requise ne peut être atteinte, IER est mise à 133 (ROMBERG_RERR devra être assez grand de sorte qu'en lui ajoutant 100.0, le résultat est un nombre plus grand que 100.0 (ce ne serait pas vrai pour de très petits nombres en virgule flottante, à cause de la nature de la machine arithmétique).
- 3. Le paramètre d'erreur absolue , ROMBERG_AERR, devra être non négatif. Afin de lui donner une valeur raisonnable, l'utilisateur doit connaître la grandeur approximative de l'intégrale à calculer. Dans bien des cas, AERR=0.0 est satisfaisant. Dans ce cas, seule l'erreur relative requise est satisfaite par le calcul.

- 4. Nous citons d'après la référence, "Un homme très prudent n'acceptera DCADRE que si IER, le code d'avertissement ou d'erreur, est 0 ou 65. L'homme simplement raisonnable conservera la foi même si IER vaut 66. L'homme aventureux a souvent raison en acceptant DCADRE même si IER vaut 131 ou 132". Même lorsque IER n'est pas 0, DCADRE retourne la meilleure estimation qui a été calculée.

Les références à cette technique se trouvent dans de Boor, Calr, "CADRE: An Algorithm for Numerical Quadrature," Mathematical Software (John R. Rice, Ed.), New York, Academic Press, 1971, Chapitre 7.

22.3 ELLIPT

Un package dans le répertoire SHARE pour les routines numériques des fonctions elliptiques et des intégrales elliptiques complètes (notation de Abramowitz et Stegun, Chs 16 et 17). Faites LOAD(ELLIPT); pour utiliser ce package. Actuellement tous les arguments DOIVENT êTRE en virgule flottante. Autrement vous obtiendrez des incohérences. Vous êtes averti. Les fonctions disponibles sont les elliptiques jacobiennes

```
AM(U,M) - amplitude avec module M
AM1(U,M1) - amplitude avec module complémentaire M1
AM(U,M):=AM1(U,1-M); donc utiliser AM1 si M ~ 1
SN(U,M):=SIN(AM(U,M));
CN(U,M):=COS(AM(U,M));
DN(U,M):=SQRT(1-M*SN(U,M)^2);
(Ces fonctions sont définies comme cela.  D'autres CD, NS etc.
 peuvent être définies de la même façon.)
Intégrales elliptiques complètes
ELLIPTK(M) - Intégrale elliptique complète du premier ordre
ELLIPTK1(M1) - La même, mais avec module complémentaire.
ELLIPTK(M):=ELLIPTK1(1-M); donc utiliser si M ~ 1
ELLIPTE(M) - Intégrale elliptique complète du second ordre
ELLIPTE1(M1) - La même, mais avec module complémentaire.
ELLIPTE(M):=ELLIPTE1(1-M); donc utiliser si M ~ 1
```

22.4 FOURIER

Package pour la transformée rapide de Fourier, faites DESCRIBE(FFT) pour les détails. Il y a aussi un package des suites de Fourier. Il peut être chargé par LOAD(FOURIE). Il calcule également les coefficients de l'intégrale de Fourier et a diverses autres fonctions pour faire des choses telle que remplacer toutes les occurrences de F(ARG) par ARG dans une expression (comme changer ABS(a*x+b) en a*x+b). Faites PRINTFILE(FOURIE,USAGE,DSK,SHARE1); pour avoir la liste des fonctions incluses.

22.5 NDIFFQ

Un package situé dans le répertoire SHARE pour les solutions numériques des équations différentielles. LOAD("NDIFFQ"); le chargera. Voici un exemple de son utilisation :

```
Define_Variable(N,0.3,FLOAT);
Define_Variable(H,0.175,FLOAT);
```

```
F(X,E):=(Mode_Declare([X,E],FLOAT),N*EXP(X)/(E+X^(2*H)*EXP(H*X)));
Compile(F);
Array([X,E],FLOAT,35);
Init_Float_Array(X,1.0E-3,6.85); /* Remplir X avec l'intervalle */
E[0]:5.0;                        /* Condition initiale */
Runge_Kutta(F,X,E);              /* La résoudre */
Graph2(X,E);                     /* Graphe de la solution */
```

P.S. Runge_Kutta(F,X,E,E_Prime) sera l'appel pour une équation du second ordre.

22.6 Définitions pour le calcul numérique

FFT (*real-array*, *imag-array*) Fonction

Transformée rapide de Fourier. Ce package est chargé par LOAD(FFT);. Il y a aussi une commande IFT, pour la transformée inverse de Fourier. Ces fonctions calculent une transformée rapide (complexe) de Fourier sur des tableaux à 1 ou 2 dimensions de FLOATING-POINT, obtenus par :

```
ARRAY(<ary>,FLOAT,<dim1>); ou
ARRAY(<ary>,FLOAT,<dim1>,<dim2>);
```

Pour les tableaux 1D

```
<dim1> = 2^n-1
```

et pour les 2D

```
<dim1>=<dim2>=2^n-1
```

(i.e. le tableau est carré). (Souvenez-vous que les tableaux de MACSYMA sont indicés à partir de l'origine 0 de sorte qu'il y aura 2^n et (2^n)^2 éléments de tableau dans les deux cas ci-dessus). Ce package contient aussi deux autres fonctions, POLARTORECT et RECTTOPOLAR. Faites DESCRIBE(cmd) pour les détails, et PRINTFILE(FFT,USAGE,SHARE); pour l'implantation.

FORTINDENT Variable

par défaut: [0] - contrôle l'indentation de la marge gauche des expressions affichées par la commande FORTRAN. 0 donne un affichage normal (i.e. 6 espaces), et des valeurs positives afficheront les expressions plus loin vers la droite.

FORTMX (*nom,matrice*) Fonction

convertit une matrice MACSYMA en une suite d'instructions FORTRAN d'affectation de la forme nom(i,j)=<élément correspondant de la matrice>. Cette commande est désormais obsolète. FORTMX(nom,matrice); peut être maintenant obtenue par FORTRAN(nom=matrice);. (Si "nom" est lié, FORTRAN('nom=matrice); peut être nécessaire). Convertissez le code qui utilise la commande FORTMX, car il sera un jour éliminé.

FORTRAN (*exp*) Fonction

convertit exp en une expression FORTRAN linéaire en FORTRAN légal avec 6 espaces insérés au début de chaque ligne, des lignes de continuation, et ** plutôt que ^ pour

l'exponentiation. Lorsque l'option FORTSPACES[FALSE] est TRUE, la commande FORTRAN remplit les 80 colonnes par des espaces. Si FORTRAN est appelée sur un atome symbolique lié, p. ex. FORTRAN(X); où X:A*B$ a été fait, alors X={valeur de X}, p. ex. X=A*B sera généré. En particulier, si p. ex. M:MATRIX(...); a été fait, alors FORTRAN(M); créera les instructions d'affectation appropriées de la forme nom(i,j)=<élément correspondant de la matrice>.

FORTINDENT[0] contrôle la marge gauche des expressions affichées, 0 est la marge normale (i.e. de 6 espaces), l'augmenter fera afficher l'expression plus à droite.

FORTSPACES Variable

par défaut: [FALSE] - si TRUE, la commande FORTRAN remplira les 80 colonnes avec des espaces.

HORNER (*exp, var*) Fonction

convertit exp en une représentation réarrangée selon la règle de Horner, en utilisant var comme variable principale si elle est spécifiée. Var peut aussi être omise, auquel cas la variable principale de la forme CRE de exp est utilisée. HORNER améliore quelquefois la stabilité si expr doit être évaluée numériquement. Elle est aussi utile si MACSYMA est utilisé pour créer des programmes devant s'exécuter en FORTRAN (voir DESCRIBE(STRINGOUT);)

```
(C1) 1.0E-20*X^2-5.5*X+5.2E20;
                                    2
(D1)                    1.0E-20 X  - 5.5 X + 5.2E+20
(C2) HORNER(%,X),KEEPFLOAT:TRUE;
(D2)                  X (1.0E-20 X - 5.5) + 5.2E+20
(C3) D1,X=1.0E20;
ARITHMETIC OVERFLOW
(C4) D2,X=1.0E20;
(D4)                            6.9999999E+19
```

IFT (*real-array, imag-array*) Fonction

Transformée inverse de Fourier. Faites LOAD(FFT); pour charger ce package. Ces fonctions (FFT et IFT) calculent une transformée rapide (complexe) de Fourier sur des tableaux FLOATING-POINT à 1 ou 2 dimensions, obtenus par :

ARRAY(<ary>,FLOAT,<dim1>); ou ARRAY(<ary>,FLOAT,<dim1>,<dim2>); . Pour les tableaux 1D <dim1> doit être égal à 2^n-1, et pour les 2D <dim1>=<dim2>=2^n-1 (i.e. le tableau est carré). (Souvenez-vous que les tableaux de MACSYMA sont indicés à partir de l'origine 0 de sorte qu'il y aura 2^n et (2^n)^2 éléments de tableau dans les deux cas ci-dessus). Faites PRINTFILE(FFT,USAGE,SHARE); pour des détails sur l'implantation.

INTERPOLATE (*func,x,a,b*) Fonction

cherche les zéros de func lorsque x varie. Les deux derniers arguments donne la plage à examiner. La fonction doit avoir des signes différents aux extrémités. Si cette

condition n'est pas remplie, l'action de la fonction sera contrôlée par INTPOLERROR[TRUE]). Si INTPOLERROR est TRUE alors une erreur se produit, autrement la valeur de INTPOLERROR est retournée (ainsi pour le tracé INTPOLERROR devra être mis à 0.0). Sinon (étant donné que MACSYMA peut évaluer le premier argument dans la portée spécifiée, et qu'elle est continue) INTERPOLATE trouvera un zéro (ou l'un d'eux s'il y en a plus d'un).

La précision de INTERPOLATE est contrôlée par INTPOLABS[0.0] et INTPOLREL[0.0] qui doivent être des nombres non négatifs en virgule flottante. INTERPOLATE s'arrêtera lorsque le premier argument devient inférieur ou égal à INTPOLABS ou si les approximants successifs vers la racine ne diffèrent pas de plus de INTPOLREL * <l'un des approximants>. Les valeurs par défaut de INTPOLABS et INTPOLREL sont 0.0 donc INTERPOLATE donne une réponse aussi bonne que possible avec la seule précision arithmétique que nous avons. Le premier argument peut être une équation. L'ordre des deux derniers arguments est sans importance. Ainsi

```
INTERPOLATE(SIN(X)=X/2,X,%PI,.1);
   est équivalent à
INTERPOLATE(SIN(X)=X/2,X,.1,%PI);
```

La méthode utilisée est une recherche binaire dans l'intervalle spécifié par les deux derniers arguments. Lorsqu'elle estime que la fonction est assez fermée pour être linéaire, elle commence à utiliser une interpolation linéaire.

Une syntaxe alternative a été ajoutée pour interpoler, qui remplace les deux premiers arguments par un nom de fonction. La fonction DOIT être une fonction TRADUITE ou compilée d'un argument. Aucune vérification du résultat n'est faite, aussi assurez-vous que la fonction retourne bien un nombre en virgule flottante.

```
F(X):=(MODE_DECLARE(X,FLOAT),SIN(X)-X/2.0);
INTERPOLATE(SIN(X)-X/2,X,0.1,%PI)       time= 60 msec
INTERPOLATE(F(X),X,0.1,%PI);            time= 68 msec
TRANSLATE(F);
INTERPOLATE(F(X),X,0.1,%PI);            time= 26 msec
INTERPOLATE(F,0.1,%PI);                 time=  5 msec
```

Il y a aussi une routine d'interpolation par la méthode de Newton, faites DESCRIBE(NEWTON); .

INTPOLABS Variable

par défaut: [0.0] - La précision de la commande INTERPOLATE est contrôlée par INTPOLABS[0.0] et INTPOLREL[0.0] qui doivent être des nombres non négatifs en virgule flottante. INTERPOLATE s'arrêtera lorsque le premier argument devient inférieur ou égal à INTPOLABS ou si les approximants successifs vers la racine ne diffèrent pas de plus de INTPOLREL * <l'un des approximants>. Les valeurs par défaut de INTPOLABS et INTPOLREL sont 0.0 donc INTERPOLATE donne une réponse aussi bonne que possible avec la seule précision arithmétique que nous avons.

INTPOLERROR Variable

par défaut: [TRUE] - Gouverne le comportement de INTERPOLATE. Lorsque celle-ci est appelée, elle détermine si la fonction à interpoler satisfait ou non à la condition que les valeurs de la fonction aux extrémités de l'intervalle d'interpolation ont des

signes différents. Si c'est le cas l'interpolation se fait. S'ils sont de même signe, et que INTPOLERROR est TRUE, une erreur est signalée. S'ils ont le même signe et que INTPOLERROR n'est pas TRUE, la valeur de INTPOLERROR est retournée. Ainsi pour le tracé, INTPOLERROR doit être 0.0.

INTPOLREL Variable

par défaut: [0.0] - La précision de la commande INTERPOLATE est contrôlée par INTPOLABS[0.0] et INTPOLREL[0.0] qui doivent être des nombres non négatifs en virgule flottante. INTERPOLATE s'arrêtera lorsque le premier argument devient inférieur ou égal à INTPOLABS ou si les approximants successifs vers la racine ne diffèrent pas de plus de INTPOLREL * <l'un des approximants>. Les valeurs par défaut de INTPOLABS et INTPOLREL sont 0.0 donc INTERPOLATE donne une réponse aussi bonne que possible avec la seule précision arithmétique que nous avons.

NEWTON (*exp,var,X0,eps*) Fonction

Le fichier NEWTON 1 du répertoire SHARE contient une fonction qui fera une interpolation par la méthode de Newton. Elle est accédée par LOAD(NEWTON); . La methode de Newton peut faire ce que INTERPOLATE refuse de traiter, car INTERPOLATE requiert que tout s'évalue à un "flonum". Ainsi NEWTON(x^2-a^2,x,a/2,a^2/100); dira qu'elle ne sait pas si flonum*a^2<a^2/100. Faire ASSUME(a>0); puis NEWTON fonctionne à nouveau. Vous obtenez x=a+<petit flonum>*a qui est entièrement symbolique. INTERPOLATE(x^2-a^2,x,a/2,2*a); se plaindra que .5*a n'est pas un flonum...

Un intégrateur adaptif qui utilise la règle de la quadrature de Newton-Cotes à 8 panneaux est disponible dans SHARE1;QQ FASL. Faites DESCRIBE(QQ) pour les détails.

POLARTORECT (*magnitude-array, phase-array*) Fonction

convertit les formes magnitude et phase en formes réelle et imaginaire en plaçant la partie réelle dans le tableau magnitude et la partie imaginaire dans le tableau phase.

```
<real>=<magnitude>*COS(<phase>) ==>
<imaginary>=<magnitude>*SIN(<phase>
```

La fonction fait partie du package FFT. Faites LOAD(FFT); pour l'utiliser. Comme FFT et IFT, cette fonction accepte des tableaux à 1 ou 2 dimensions. Cependant, les dimensions du tableau n'ont pas à être des puissances de 2, et les tableaux 2D ne sont pas forcément carrés.

RECTTOPOLAR (*real-array, imag-array*) Fonction

inverse POLARTORECT. La phase est donnée dans l'intervalle -%PI à %PI. Cette fonction fait partie du package FFT. Faites LOAD(FFT); pour l'utiliser. Comme FFT et IFT cette fonction accepte des tableaux à 1 ou 2 dimensions. Cependant, les dimensions du tableau n'ont pas à être des puissances de 2, et les tableaux 2D ne sont pas forcément carrés.

23 Statistiques

23.1 Définitions pour les statistiques

GAUSS (*mean,sd*) Fonction

retourne un nombre aléatoire en virgule flottante à partir d'une distribution normale avec une moyenne MEAN et une déviation standard SD. Elle fait partie du package de la fonction de BESSEL, faites LOAD(BESSEL); pour l'utiliser.

24 Tableaux et tables

24.1 Définitions pour les tableaux et les tables

ARRAY (*nom, dim1, dim2, ..., dimk*) Fonction

Définit un tableau à k-dimensions. Un maximum de cinq dimensions peut être utilisé. Les indices pour la ième dimension sont des entiers allant de 0 à dimi. Si l'utilisateur affecte une valeur à une variable indicée avant d'avoir déclaré le tableau correspondant, un tableau non déclaré est défini. Si l'utilisateur a plus d'un tableau à définir de la même façon, ils peuvent tous être définis en même temps, avec ARRAY([liste-de-noms],dim1, dim2, ..., dimk). Les tableaux non déclarés, aussi dits tableaux "hachés" (car le codage "hash" est fait sur les indices), sont plus généraux que les tableaux déclarés. L'utilisateur ne déclare pas leur taille maximale, et ils croissent dynamiquement par hachage lorsque des valeurs sont affectées à plus d'éléments. Les indices des tableaux non déclarés ne sont pas forcément des nombres. Cependant, à moins qu'un tableau soit plutôt vide, il est probablement plus efficace de le déclarer si possible plutôt que de le laisser non déclaré. La fonction ARRAY peut être utilisée pour transformer un tableau non déclaré en un tableau déclaré.

ARRAYAPPLY (*tableau,[sub1, ... ,subk]*) Fonction

est comme APPLY sauf que le premier argument est un tableau.

ARRAYINFO (*a*) Fonction

retourne une liste d'information concernant le tableau a. Pour les tableaux "hachés" elle retourne une liste de "HASHED", le nombre d'indices, et les indices de tous les éléments ayant une valeur. Pour les tableaux déclarés elle retourne une liste de "DECLARED", le nombre d'indices, et les bornes qui ont été données à la fonction ARRAY lorsqu'elle a été appelée sur a. Faites EXAMPLE(ARRAYINFO); pour un exemple.

ARRAYMAKE (*nom,[i1,i2,...]*) Fonction

retourne nom[i1,i2,...].

ARRAYS Variable

par défaut: [] - une liste de tous les tableaux qui ont été alloués, tant déclarés que non déclarés. Les fonctions qui traitent les tableaux sont : ARRAY, ARRAYAPPLY, ARRAYINFO, ARRAYMAKE, FILLARRAY, LISTARRAY, et REARRAY.

BASHINDICES (*expr*) Fonction

transforme l'expression expr en donnant à chaque sommation et produit un unique indice. Ce qui donne à CHANGEVAR une plus grande précision lorsqu'elle travaille sur des sommations ou des produits. La forme de l'unique indice est J<nombre>. La quantité <nombre> est déterminée par référence à GENSUMNUM, qui peut être modifiée par l'utilisateur. Par exemple, GENSUMNUM:0$ la redéfinit.

FILLARRAY (*tableau,liste-ou-tableau*) Fonction

remplit "tableau" depuis la liste-ou-tableau. Si "tableau" est un tableau en virgule flottante (entier) alors liste-ou-tableau devra être soit une liste de nombres en virgule flottante (entiers) ou un autre tableau en virgule flottante (entiers). Si les dimensions des tableaux sont différentes tableau est rempli par ordre de (rangée majeure row-major order ???). S'il n'y a pas assez d'éléments dans liste-ou-tableau le dernier élément est utilisé pour remplir le reste du tableau. S'il y en a trop les éléments restants sont ignorés. FILLARRAY retourne son premier argument.

GETCHAR (*a, i*) Fonction

retourne le ième caractère de la chaîne entre guillemets ou du nom atomique a. Cette fonction est utile pour manipuler la liste des LABELS.

LISTARRAY (*tableau*) Fonction

retourne une liste des éléments d'un tableau déclaré ou "haché". l'ordre est en (row-major ???). Les éléments que vous n'avez pas encore définis seront représentés par #####.

MAKE_ARRAY (*type,dim1,dim2,...,dimn*) Fonction

crée un tableau. "type" peut être 'ANY, 'FLONUM, 'FIXNUM, 'HASHED ou 'FUNCTIONAL. Semblable à la commande ARRAY, sauf que le tableau créé est un objet tableau fonctionnel. L'avantage sur ARRAY est qu'il n'a pas de nom, et dès qu'un pointeur sur lui est supprimé, il l'est aussi. P. ex., Y:MAKE_ARRAY(....); fait pointer Y sur un objet qui prend de la place, mais faire Y:FALSE, et Y ne pointe plus sur cet objet, celui-ci va partir dans le ramasse-miettes.

Note: les "dimi" sont ici différents de ceux de la commande ARRAY, car ils vont de 0 à i-1, i.e. une "dimension" de 10 signifie que vous avez les éléments de 0 à 9.

Y:MAKE_ARRAY('FUNCTIONAL,'F,'HASHED,1) - Le second argument de MAKE_ARRAY est dans ce cas la fonction à appeler pour calculer les éléments du tableau, et les autres arguments sont passés récursivement à MAKE_ARRAY pour générer la "mémoire" de l'objet (array function ??? fonction tableau).

REARRAY (*tableau,dim1, ... ,dimk*) Fonction

peut être utilisé pour changer la taille ou les dimensions d'un tableau. Le nouveau tableau sera rempli avec les éléments de l'ancien en ordre (row-major ???). Si l'ancien tableau était trop petit, FALSE, 0.0 ou 0 seront utilisés pour remplir les éléments restants, selon le type du tableau. Ce type ne peut pas être changé.

REMARRAY (*nom1, nom2, ...*) Fonction

supprime les tableaux associés aux fonctions et libère l'espace de stockage occupé. Si nom est ALL tous les tableaux sont supprimés. Il peut être nécessaire d'utiliser cette fonction pour redéfinir les valeurs d'un tableau "haché".

USE_FAST_ARRAYS Variable

[TRUE sur Lispm] - Si TRUE seuls deux types de tableaux sont reconnus.

1) Le tableau "art-q" (t en Common Lisp) qui peut avoir plusieurs dimensions indexées par des entiers, et pouvant contenir comme entrée tout objet lisp ou macsyma. Pour construire un tel tableau, entrez A:MAKE_ARRAY(ANY,3,4); , alors A aura comme valeur, un tableau de douze positions, avec des indices de base zéro.

2) Le tableau hash_table qui est par défaut le type de tableau créé si l'on fait B[X+1]:Y^2 (et que B n'est pas déjà un tableau, une liste, ou une matrice – s'il est l'un de ces objets une erreur se produira puisque x+1 ne serait pas un indice valable pour un tableau "art-q", une liste ou une matrice). Ses indices (appelés aussi des clés) peuvent être un objet quelconque. Il ne prend qu'une SEULE CLé à la fois (B[X+1,U]:Y ignorera le u). Le référencement se fait par B[X+1]==> Y^2. Bien sûr la clé peut être une liste, p. ex. B[[x+1,u]]:y serait correcte. Ceci est incompatible avec les anciens tableaux "hash" de Macsyma, mais économise les "cons" (fonction de création de fonction du Lisp).

L'un des avantages de stocker les tableaux comme valeurs du symbole est que les conventions usuelles à propos des variables locales d'une fonction s'appliquent aussi aux tableaux. Le type hash_table utilise aussi moins de "cons" et est plus efficace que l'ancien type hashar de macsyma. Pour obtenir un comportement logique du code traduit et compilé mettez TRANSLATE_FAST_ARRAYS [TRUE] à TRUE.

25 Matrices et algèbre linéaire

25.1 Introduction aux matrices et à l'algèbre linéaire

25.1.1 DOT

L'opérateur point, ".", pour la multiplication (non commutative) des matrices. Lorsque le "." est utilisé de cette façon, il devra être entouré d'espaces, p. ex. A . B. Ceci le distingue entièrement du point décimal d'un nombre en virgule flottante. Faites APROPOS(DOT); pour une liste des commutateurs qui affectent l'opérateur point.

25.1.2 Vecteurs

Le fichier SHARE;VECT > contient un package d'analyse de vecteur, share/vect.dem contient la démonstration correspondante, et SHARE;VECT ORTH contient les définitions de divers systèmes de coordonées curvilinéaires orthogonales. LOAD(VECT); chargera ce package pour vous. Le package d'analyse de vecteur peut combiner et simplifier les expressions symboliques contenant des produits "points" et des produits croisés, avec les opérateurs de gradient, de divergence, CURL et Laplaciens. La distribution de ces opérateurs sur les sommes ou produits est sous contrôle de l'utilisateur, comme le sont d'autres divers développements, y compris des développements en composants des systèmes spécifiques de coordonnées orthogonales. Existe aussi une possibilité de dériver le potentiel scalaire ou vectoriel d'un champ.

Le package contient les commandes suivantes : VECTORSIMP, SCALEFACTORS, EXPRESS, POTENTIAL, et VECTORPOTENTIAL. Faites DESCRIBE(cmd) sur les noms de ces commandes, ou PRINTFILE(VECT,USAGE,SHARE); pour les détails.

Attention: le package VECT déclare "." comme étant un opérateur commutatif.

25.2 Définitions pour les matrices et l'algèbre linéaire

ADDCOL (*M,list1,list2,...,listn*) Fonction
ajoute la/les colonne(s) donnée(s) par la/les listes (ou les matrices) à la matrice M.

ADDROW (*M,list1,list2,...,listn*) Fonction
ajoute la/les ligne(s) données par la/les listes (ou les matrices) à la matrice M.

ADJOINT (*matrice*) Fonction
calcule l'(adjoint ???) d'une matrice.

AUGCOEFMATRIX ([*eq1, ...*], [*var1, ...*]) Fonction
la matrice des coefficients augmenté pour les variables var1,... du système d'équations linéaires eq1,.... C'est la matrice coefficient avec une colonne ajoutée pour les termes constants de chaque équation (i.e. ceux ne dépendant pas de var1,...). Faites EXAMPLE(AUGCOEFMATRIX); pour un exemple.

CHARPOLY (*M, var*) Fonction

calcule le polynôme caractéristique de la matrice M par rapport à var, i.e., DETERMINANT(M - DIAGMATRIX(LENGTH(M),var)). Pour des exemples de cette commande, faites EXAMPLE(CHARPOLY);.

COEFMATRIX ([*eq1, ...*], [*var1, ...*]) Fonction

la matrice des coefficients pour les variables var1,... du système d'équations linéaires eq1,...

COL (*M,i*) Fonction

donne une matrice de la i-ème colonne de la matrice M.

COLUMNVECTOR (*X*) Fonction

une fonction du package EIGEN. Faites LOAD(EIGEN) pour l'utiliser. COLUMNVECTOR prend une liste pour argument et retourne un vecteur colonne des composants qui sont les éléments de la liste. Le premier élément est le premier composant,...etc...(Utile si vous voulez utiliser en partie les sorties des fonctions de ce package de calculs sur les matrices).

CONJUGATE (*X*) Fonction

une fonction du package EIGEN dans le répertoire SHARE. Elle retourne le conjugué complexe de son argument. Ce package peut être chargé par LOAD(EIGEN);. Pour une description complète de ce package, faites PRINTFILE("eigen.usg");.

COPYMATRIX (*M*) Fonction

crée une copie de la matrice M. C'est la seule façon de faire une copie en dehors de recréer M élément par élément. Copier une matrice peut être utile lorsque SETELMX est utilisée.

DETERMINANT (*M*) Fonction

calcule le déterminant de M par une méthode semblable à l'élimination gaussienne. La forme du résultat dépend du paramétrage du commutateur RATMX. Il y a une routine spéciale pour le traitement des déterminants creux qui peut être utilisée en définissant les commutateurs RATMX:TRUE et SPARSE:TRUE.

DETOUT Variable

par défaut: [FALSE] - si TRUE le déterminant d'une matrice dont l'inverse est calculé est laissé en dehors de l'inverse. Pour que ce commutateur ait un effet DOALLMXOPS et DOSCMXOPS devront être FALSE (voyez leurs descriptions). Alternativement ce commutateur peut être mis à EV qui définit correctement les deux autres.

DIAGMATRIX (*n, x*) Fonction

retourne une matrice diagonale de taille n par n avec x pour tous les éléments diagonaux. Une matrice identité est créée par DIAGMATRIX(n,1), ou par IDENT(n).

DOALLMXOPS Variable

par défaut: [TRUE] - si TRUE toutes les opérations en relation avec les matrices sont exécutées. Si FALSE le paramétrage des commutateurs DOT individuels contrôle les opérations à exécuter.

DOMXEXPT Variable

par défaut: [TRUE] - si TRUE,

```
%E^MATRIX([1,2],[3,4]) ==>
MATRIX([%E,%E^2],[%E^3,%E^4])
```

En général, cette transformation affecte les expressions de la forme <base>^<puissance> où <base> est une expression supposée scalaire ou constante, et <puissance> est une liste ou une matrice. Cette transformation est inactivée si ce commutateur est mis à FALSE.

DOMXMXOPS Variable

par défaut: [TRUE] - si TRUE toutes les opérations matrice-matrice ou matrice-liste sont exécutées (mais pas les opérations scalaire-matrice); elles ne le sont pas si ce commutateur est FALSE.

DOMXNCTIMES Variable

par défaut: [FALSE] - provoque l'exécution des produits non commutatifs de matrices.

DONTFACTOR Variable

par défaut: [] - peut être défini comme liste de variables par rapport auxquelles la factorisation n'a pas lieu (elle est initialement vide). La factorisation ne se produit pas non plus par rapport à toutes les variables qui sont moins importantes (en utilisant l'ordre des variables de la forme CRE) que celles de la liste DONTFACTOR.

DOSCMXOPS Variable

par défaut: [FALSE] - si TRUE les opérations scalaire-matrice sont exécutées.

DOSCMXPLUS Variable

par défaut: [FALSE] - si TRUE SCALAR + MATRIX donnent une matrice en réponse. Ce commutateur est indépendant de DOALLMXOPS.

DOT0NSCSIMP Variable

par défaut: [TRUE] - provoque la simplification en un produit commutatif d'un produit non commutatif de zéro et d'un terme non scalaire.

DOT0SIMP Variable

par défaut: [TRUE] - provoque la simplification en un produit commutatif d'un produit non commutatif de zéro et d'un terme non scalaire.

DOT1SIMP Variable

par défaut: [TRUE] - provoque la simplification en un produit commutatif d'un produit non commutatif de un et d'un autre terme.

DOTASSOC Variable

par défaut: [TRUE] - simplifie (A.B).C en A.(B.C).

DOTCONSTRULES Variable

par défaut: [TRUE] - provoque la simplification en un produit commutatif d'un produit non commutatif d'une constante et d'un autre terme. Activer ce drapeau active aussi effectivement DOT0SIMP, DOT0NSCSIMP, et DOT1SIMP.

DOTDISTRIB Variable

par défaut: [FALSE] - si TRUE simplifie A.(B+C) en A.B+A.C

DOTEXPTSIMP Variable

par défaut: [TRUE] - simplifie A.A en A^^2

DOTIDENT Variable

par défaut: [1]. La valeur à retournée par X^^0.

DOTSCRULES Variable

par défaut: [FALSE] - si TRUE simplifie A.SC ou SC.A en SC*A et A.(SC*B) en SC*(A.B)

ECHELON (*M*) Fonction

produit la forme "en échelon" de la matrice M. C'est à dire que les opérations élémentaires sur les lignes sont exécutées sur M de telle sorte que le premier élément non zéro dans chaque ligne de la matrice résultante est un 1 et les éléments de colonne sous le premier de chaque ligne sont tous zéro.

```
                          [2  1 - A  -5 B ]
(D2)                      [               ]
                          [A    B      C  ]

(C3) ECHELON(D2);
                  [      A - 1        5 B      ]
                  [1   - -----      - ---      ]
                  [        2           2       ]
(D3)              [                            ]
                  [                2 C + 5 A B ]
                  [0     1         ------------]
                  [                       2    ]
                  [                2 B + A  - A]
```

EIGENVALUES (*mat*) Fonction

Un package du répertoire SHARE; contient des fonctions de calcul des EIGENVALUES (valeurs propres) et EIGENVECTORS (vecteurs propres) et des calculs apparentés sur les matrices. Pour avoir les informations faites PRINTFILE(EIGEN,USAGE,SHARE); . EIGENVALUES(mat) prend une matrice

comme argument et retourne une liste des listes dont la première sous-liste est celle des valeurs propres de la matrice et l'autre sous-liste celle de la liste des (multiplicities == multiplicités???) des valeurs propres dans l'ordre correspondant. [La fonction SOLVE de MACSYMA est utilisée ici pour trouver les racines du polynôme caractéristique de la matrice. Quelquefois SOLVE ne trouve pas les racines du polynôme ; dans ce cas rien dans ce package sauf CONJUGATE, INNERPRODUCT, UNITVECTOR, COLUMNVECTOR et GRAMSCHMIDT ne fonctionnera, à moins que vous connaisiez les valeurs propres. Dans certains cas SOLVE peut générer des valeurs propres très désordonnées. Vous pouvez simplifier vous-même les réponses avant de continuer. Cela est prévu et sera expliqué plus loin. (Ceci se produit en général lorsque SOLVE retourne une expression réelle pas très claire sur une valeur propre qui est supposée être réelle...)] La commande EIGENVALUES est disponible directement depuis MACSYMA. Pour utiliser les autres fonctions vous devez avoir chargé le package EIGEN, soit par un appel précédent à EIGENVALUES, soit en faisant LOADFILE("eigen"); .

EIGENVECTORS (*MAT*) Fonction

prend une matrice en argument et retourne une liste des listes dont la première sous-liste est la sortie de la commande EIGENVALUES et les autres sous-listes sont les vecteurs propres de la matrice correspondant respectivement à ces valeurs propres. Cette fonction peut travailler directement depuis MACSYMA, mais si vous voulez la contrôler par son drapeau (voir ci-dessous), vous devez tout d'abord charger le package EIGEN du répertoire SHARE. Vous pouvez le faire par LOADFILE("eigen");. Les drapeaux qui contrôlent cette fonction sont :

NONDIAGONALIZABLE[FALSE] sera mis à TRUE ou FALSE, selon que la matrice est non diagonalisable ou diagonalisable après qu'une commande EIGENVECTORS a été exécutée.

HERMITIANMATRIX[FALSE]. Si mis à TRUE les vecteurs propres dégénérés de la matrice hermitienne seront orthogonalisés à l'aide de l'algorithme de Gram-Schmidt.

KNOWNEIGVALS[FALSE]. Si mis à TRUE le package EIGEN supposera que les valeurs propres de la matrice sont connues de l'utilisateur et stockées sont le nom global LISTEIGVALS. LISTEIGVALS devra être une liste semblable à la sortie de la commande EIGENVALUES. (La fonction ALGSYS de MACSYMA est utilisée ici pour résoudre les vecteurs propres. Quelquefois si les valeurs propres sont désordonnées, ALGSYS peut ne pas trouver une solution. Dans ce cas il vous est demandé d'essayer de simplifier les valeurs propres en les trouvant d'abord à l'aide de la commande EIGENVALUES puis en utilisant toutes les merveilleuses astuces que vous pouvez posséder pour les réduire à quelque chose de plus simple. Vous pouvez utiliser le drapeau KNOWNEIGVALS pour aller plus loin).

EMATRIX (*m, n, x, i, j*) Fonction

crée une matrice m par n dont tous les éléments sont zéro sauf l'élément i,j qui est x.

ENTERMATRIX (*m, n*) Fonction

crée une matrice élément par élément avec MACSYMA demandant les valeurs pour chacune des m*n entrées.

```
(C1) ENTERMATRIX(3,3);
La matrice est-elle 1. Diagonale  2. Symétrique  3. Antisymétrique
 4. Générale

Répondez par 1, 2, 3 ou 4
1;
Row 1 Column 1:  A;
Row 2 Column 2:  B;
Row 3 Column 3:  C;
La matrice est entrée.
                                [ A  0  0 ]
                                [         ]
(D1)                            [ 0  B  0 ]
                                [         ]
                                [ 0  0  C ]
```

GENMATRIX (*tableau, i2, j2, i1, j1*) Fonction

génère une matrice depuis le tableau en utilisant tableau(i1,j1) pour le premier élément (supérieur gauche) et tableau(i2,j2) pour le dernier (inférieur droit) de la matrice. Si j1=i1 alors j1 peut être omis. Si j1=i1=1 alors i1 et j1 peuvent tous deux être omis. Si un élément sélectionné du tableau n'existe pas un élément symbolique sera utilisé.

```
(C1) H[I,J]:=1/(I+J-1)$
(C2) GENMATRIX(H,3,3);
                              [   1  1]
                              [1  -  -]
                              [   2  3]
                              [       ]
                              [1  1  1]
(D2)                          [-  -  -]
                              [2  3  4]
                              [       ]
                              [1  1  1]
                              [-  -  -]
                              [3  4  5]
```

GRAMSCHMIDT (*X*) Fonction

une fonction du package EIGEN. Faites LOAD(EIGEN) pour l'utiliser. GRAMSCHMIDT prend comme argument une LIST des listes des sous-listes qui sont d'égale longueur mais pas nécessairement orthogonales (par rapport au produit interne défini ci-dessus) et retourne une liste similaire, chaque sous-liste étant orthogonale à toutes les autres. (Les résultats retournés peuvent contenir des entiers factorisés. Ceci est dû au fait que la fonction FACTOR de MACSYMA est utilisée pour simplifier chaque sous-étape de l'algorithme de Gram-Schmidt, ce qui empêche les expressions de de-

venir très désordonnées et aide à réduire les tailles des nombres qui sont créés par le processus).

HACH (*a,b,m,n,l*) Fonction

Une implantation de l'algorithme de programmation linéaire d'Hacijan est disponible en faisant BATCH("kach.mc"$. Pour les détails d'utilisation faire BATCH("kach.dem");

IDENT (*n*) Fonction

produit une matrice identité de n par n.

INNERPRODUCT (*X,Y*) Fonction

Une fonction du package EIGEN. Faites LOAD(EIGEN) pour l'utiliser. INNERPRODUCT prend deux listes d'égale longueur en arguments et retourne leur produit interne (scalaire) défini par (conjugué complexe de X).Y (l'opération "point (dot)" est la même que celle définie pour les vecteurs).

INVERT (*matrice*) Fonction

cherche l'inverse d'une matrice en utilisant la méthode de l'adjoint. Ceci permet à un utilisateur de calculer l'inverse d'une matrice avec des entrées "bfloat" ou des polynômes à coefficients en virgule flottante sans conversion en forme cre. La commande DETERMINANT est utilisée pour calculer les cofacteurs, ainsi si RATMX est FALSE (par défaut) l'inverse est calculé sans changer la représentation des éléments. L'implantation actuelle est inefficace pour les matrices d'ordres supérieurs.

Le drapeau DETOUT si vrai laisse en dehors de l'inverse le déterminant factorisé.

Note: les résultats ne sont pas automatiquement développés. Si la matrice a d'origine des polynôme en entrées, une meilleure sortie est générée avec EXPAND(INVERT(mat)),DETOUT. S'il est prévu de diviser par le déterminant ceci peut s'obtenir avec XTHRU(%) ou par EXPAND(ADJOINT(mat))/EXPAND(DETERMINANT(mat)). INVERT(mat):=ADJOINT(mat)/DETERMINANT(mat).

Voir aussi DESCRIBE("^^"); pour une autre méthode d'inversion de matrice.

LMXCHAR Variable

par défaut: [] - Le caractère utilisé pour afficher le délimiteur (gauche) d'une matrice (voir aussi RMXCHAR).

MATRIX (*row1, ..., rown*) Fonction

définit une matrice rectangulaire ayant les lignes indiquées. Chaque ligne a la forme d'une liste d'expressions, p. ex. [A, X**2, Y, 0] est une liste de 4 éléments. Il y a nombre de commandes MACSYMA qui traitent des matrices, par exemple: DETERMINANT, CHARPOLY, GENMATRIX, ADDCOL, ADDROW, COPYMATRIX, TRANSPOSE, ECHELON, et RANK. Il y a aussi un package dans le répertoire SHARE pour calculer les EIGENVALUES (valeurs propres). Essayez DESCRIBE sur celles-ci pour avoir davantage d'informations.

La multiplication des matrices est effectuée en utilisant l'opérateur point, ".", qui est aussi pratique si l'utilisateur désire représenter d'autres opérations algébriques non-commutatives. L'exponentielle de l'opération "." est "^^" . Ainsi, pour une matrice A, A.A = A^^2 et, si elle existe, A^^-1 est l'inverse de A.

Les opérations +,-,*,** sont toutes des opérations élément par élément; toutes les opérations sont normalement effectuées en entier, y compris l'opération . (point). Beaucoup de commutateurs existent afin de contrôler les règles de simplification impliquant les opérations point et liste-matrice.

Options en relation avec les matrices:

LMXCHAR, RMXCHAR, RATMX, LISTARITH, DETOUT, DOALLMXOPS, DOMXEXPT DOMXMXOPS, DOSCMXOPS, DOSCMXPLUS, SCALARMATRIX, et SPARSE. Faites DESCRIBE(option) pour en avoir les détails.

MATRIXMAP (*fn, M*) Fonction

applique la fonction fn à chaque élément de la matrice M.

MATRIXP (*exp*) Fonction

est TRUE si exp est une matrice, sinon FALSE.

MATRIX_ELEMENT_ADD Variable

par défaut: [+] - Peut être mise à "?"; peut aussi être le nom d'une fonction, ou une expression LAMBDA. De cette façon une riche variété de structures algébriques peut être simulée. Pour plus de détails, faites DEMO("matrix.dem1"); et DEMO("matrix.dem2");.

MATRIX_ELEMENT_MULT Variable

par défaut: [*] - Peut être mise à "."; peut aussi être le nom d'une fonction, ou une expression LAMBDA. De cette façon une riche variété de structures algébriques peut être simulée. Pour plus de détails, faites DEMO("matrix.dem1"); et DEMO("matrix.dem2");.

MATRIX_ELEMENT_TRANSPOSE Variable

par défaut: [FALSE] - D'autres paramètrages utiles sont TRANSPOSE et NONSCALARS; peut aussi être le nom d'une fonction, ou une expression LAMBDA. De cette façon une riche variété de structures algébriques peut être simulée. Pour plus de détails, faites DEMO("matrix.dem1"); et DEMO("matrix.dem2");.

MATTRACE (*M*) Fonction

calcule la trace [somme des éléments de la diagonale principale] de la matrice carrée M. Utilisée par NCHARPOLY, une alternative à CHARPOLY de MACSYMA. Elle est utilisée en faisant LOADFILE("nchrpl");

MINOR (*M, i, j*) Fonction

calcule le mineur i,j de la matrice M, i.e. M avec la ligne i et la colonne j supprimées.

NCEXPT (*A,B*) Fonction

si une expression (non-commutative) exponentielle est trop large pour être affichée comme A^^B elle va apparaître sous la forme NCEXPT(A,B).

NCHARPOLY (*M,var*) Fonction

cherche le polynôme caractéristique de la matrice M par rapport à var. C'est une alternative à CHARPOLY de MACSYMA. NCHARPOLY travaille en calculant les traces des puissances de la matrice donnée, qui sont connues comme étant égales aux sommes des puissances des racines du polynôme caractéristique. Depuis ces quantités les fonctions symétriques des racines peuvent être calculées, qui ne sont rien de plus que les coefficients du polynôme caractéristique. CHARPOLY travaille en formant le déterminant de VAR * IDENT [N] - A. Ainsi NCHARPOLY gagne, par exemple, dans le cas de grandes matrices denses remplies d'entiers, puisqu'elle évite l'arithmétique sur le polynôme. Elle peut être utilisée en faisant LOADFILE("nchrpl");

NEWDET (*M,n*) Fonction

calcule aussi le déterminant de M mais utilise l'algorithme de l'arbre mineur de Johnson-Gentleman. M peut être le nom d'une matrice ou d'un tableau. L'argument n est l'ordre; il est optionnel si M est une matrice.

NONSCALAR declaration

fait se comporter ai comme le fait une liste ou une matrice par rapport à l'opérateur point.

NONSCALARP (*exp*) Fonction

est TRUE si exp est un non scalaire, i.e. il contient des atomes déclarés comme non scalaires, listes, ou matrices.

PERMANENT (*M,n*) Fonction

calcule le permanent de la matrice M. Un permanent est comme un déterminant mais sans changement de signe.

RANK (*M*) Fonction

calcule le rang de la matrice M, i.e. l'ordre du plus grand sous-déterminant non singulier de M. Attention : RANK peut renvoyer une réponse fausse s'il ne peut déterminer qu'un élément de matrice qui est équivalent à zéro l'est en fait.

RATMX Variable

par défaut: [FALSE] - si FALSE provoquera l'exécution du déterminant et de l'addition, la soustraction, et la multiplication de la matrice en fonction de la représentation des éléments de la matrice et laissera en représentation générale le résultat de l'inversion de la matrice. Si TRUE, les 4 opérations mentionnées ci-dessus seront exécutées sous forme CRE, ainsi que le résultat de l'inversion de la matrice. Notez que ceci peut provoquer le développement des éléments (cela dépend du paramétrage de RATFAC) ce qui n'est pas toujours désirable.

ROW (*M, i*) Fonction

donne une matrice de la i-ème ligne de la matrice M.

SCALARMATRIXP Variable

par défaut: [TRUE] - si TRUE, alors dès qu'une matrice 1 x 1 est produite comme résultat du calcul d'un produit "point" de matrices, elle sera convertie en un scalaire, à savoir l'unique élément de la matrice. Si mis à ALL, cette conversion se produit dès qu'une matrice 1 x 1 est simplifiée. Si FALSE, aucune conversion ne sera faite.

SETELMX (*x, i, j, M*) Fonction

change le i,j élément de M en x. La matrice modifiée est retournée comme valeur. La notation M[i,j]:x peut aussi être utilisée, en modifiant M de façon semblable, mais en retournant x comme valeur.

SIMILARITYTRANSFORM (*MAT*) Fonction

une fonction du package EIGEN. Faites LOAD(EIGEN) pour l'utiliser. SIMILARITYTRANSFORM prend une matrice en argument et retourne une liste qui est la sortie de la commande UNITEIGENVECTORS. De plus, si le drapeau NONDIAGONALIZABLE est FALSE deux matrices globales LEFTMATRIX et RIGHTMATRIX seront créées. Ces matrices ont la propriété que LEFTMATRIX.MAT.RIGHTMATRIX est une matrice diagonale ayant les valeurs propres de MAT sur la diagonale. Si NONDIAGONALIZABLE est TRUE ces deux matrices ne seront pas générées. Si le drapeau HERMITIANMATRIX est TRUE alors LEFTMATRIX est la conjugué complexe de la transposée de RIGHTMATRIX. Autrement LEFTMATRIX est l'inverse de RIGHTMATRIX. RIGHTMATRIX est la matrice dont les colonnes sont les vecteurs propres unité de MAT. Les autres drapeaux (voir DESCRIBE(EIGENVALUES); et DESCRIBE(EIGENVECTORS);) ont le même effets car SIMILARITYTRANSFORM appelle les autres fonctions du package afin de pouvoir former RIGHTMATRIX.

SPARSE Variable

par défaut: [FALSE] - si TRUE et si RATMX:TRUE alors DETERMINANT utilisera des routines spéciales pour calculer les déterminants creux.

SUBMATRIX (*m1, ..., M, n1, ...*) Fonction

crée une nouvelle matrice composée de la matrice M dont les rangées mi et les colonnes ni sont supprimées.

TRANSPOSE (*M*) Fonction

produit la transposée de la matrice M.

TRIANGULARIZE (*M*) Fonction

produit la forme triangulaire supérieure de la matrice M qui n'est pas forcément carrée.

UNITEIGENVECTORS (*MAT*) Fonction

une fonction du package EIGEN. Faites LOAD(EIGEN) pour l'utiliser. UNITEIGENVECTORS prend une matrice comme argument et retourne une liste des listes dont la première sous-liste est la sortie de la commande EIGENVALEURS et les autres sous-listes sont les vecteurs propres unité de la matrice correspondant respectivement à ces valeurs propres. Le drapeau mentionné dans la description de la commande EIGENVECTORS a le même effet dans celle-ci. De plus il existe un drapeau pouvant être utile :

KNOWNEIGVECTS[FALSE] - si TRUE le package EIGEN supposera que les vecteurs propres de la matrice sont connus de l'utilisateur et qu'ils sont stockés sous le nom global LISTEIGVECTS. LISTEIGVECTS devra être une liste semblable à la sortie de la commande EIGENVECTORS. (Si KNOWNEIGVECTS est TRUE et que la liste des vecteurs propres est le paramètre du drapeau, NONDIAGONALIZABLE peut ne pas être correct. Dans ce cas donnez-lui la valeur convenable. L'auteur suppose que l'utilisateur sait ce qu'il fait et qu'il n'essaiera pas de diagonaliser une matrice dont les vecteurs propres ne parcourent pas l'espace vectoriel de la dimension appropriée ...).

UNITVECTOR (*X*) Fonction

une fonction du package EIGEN. Faites LOAD(EIGEN) pour l'utiliser. UNITVECTOR prend une LIST en argument et retourne une liste unité, i.e. une liste de taille unité.

VECTORSIMP (*vecteur-expression*) Fonction

Cette fonction emploie des simplifications additionnelles, ainsi que divers développement optionnels selon le paramétrage des drapeaux globaux suivants :

```
EXPANDALL, EXPANDDOT, EXPANDDOTPLUS, EXPANDCROSS, EXPANDCROSSPLUS,
EXPANDCROSSCROSS, EXPANDGRAD, EXPANDGRADPLUS, EXPANDGRADPROD,
EXPANDDIV, EXPANDDIVPLUS, EXPANDDIVPROD, EXPANDCURL, EXPANDCURLPLUS,
EXPANDCURLCURL, EXPANDLAPLACIAN, EXPANDLAPLACIANPLUS,
EXPANDLAPLACIANPROD.
```

Tous ces drapeaux ont par défaut la valeur FALSE. Le suffixe PLUS se réfère à l'emploi de l'additivité ou la distributivité. Le suffixe PROD se réfère au développement d'un opérande qui est toute sorte de produit. EXPANDCROSSCROSS se réfère au remplacement de p~(q~r) par (p.r)*q-(p.q)*r, et EXPANDCURLCURL à celui de CURL CURL p par GRAD DIV p + DIV GRAD p. EXPANDCROSS:TRUE a le même effet que EXPANDCROSSPLUS:EXPANDCROSSCROSS:TRUE, etc. Deux autres drapeaux, EXPANDPLUS et EXPANDPROD, ont le même effet que mettre tous les drapeaux suffixés de même à vrai. Si TRUE, un autre drapeau nommé EXPANDLAPLACIANTODIVGRAD, remplace l'opérateur LAPLACIAN par la composition DIV GRAD. Tous ces drapeaux sont initialement FALSE. Par commodité, tous ces drapeaux ont été déclarés EVFLAG.

Pour les coordonnées curvilinéaires orthogonales, les variables globales COORDINATES[[X,Y,Z]], DIMENSION[3], SF[[1,1,1]], et SFPROD[1] sont définies à l'appel de la fonction.

VECT_CROSS Variable

par défaut:[FALSE] - Si TRUE permet à DIFF(X~Y,T) de travailler si ~ est définie dans SHARE;VECT (où VECT_CROSS est mis à TRUE, de toute façon).

ZEROMATRIX (*m,n*) Fonction

prend les entiers m,n en arguments et retourne une matrice m par n de 0.

"[" symbole spécial

[et] sont les caractères que MACSYMA utilise pour délimiter une liste.

26 Affine

26.1 Définitions pour affine

FAST_LINSOLVE (*eqns,variables*) Fonction

Résoud le système linéaire d'équations EQNS par rapport aux VARIABLES et retourne un résultat convenant à SUBLIS. La fonction est plus rapide que linsolve pour les systèmes d'équations creux.

GROBNER_BASIS (*eqns*) Fonction

Prend comme argument une liste d'équations et retourne leur "base de grobner". La fonction POLYSIMP peut être alors utilisée pour simplifier d'autres fonctions relatives aux équations.

GROBNER_BASIS([3*X^2+1,Y*X])$

POLYSIMP(Y^2*X+X^3*9+2)==> -3*x+2

Polysimp(f)==> 0 si et seulement si f est dans l'idéal généré par EQNS, i.e. si et seulement si f est une combinaison polynomiale des éléments de EQNS.

SET_UP_DOT_SIMPLIFICATIONS (*eqns,*[*check-thru-degree*]) Fonction

Les eqns sont des équations polynomiales de variables non commutatives. La valeur de CURRENT_VARIABLES est la liste des variables utilisées pour calculer les degrés. Les équations doivent être homogènes, pour que la procédure se termine.

Si vous avez vérifié des simplifications "recouvrantes" dans DOT_SIMPLIFICATIONS au-dessus du degré de f, alors ce qui suit est vrai : DOTSIMP(f)==> 0 si et seulement si f est dans l'idéal généré par les EQNS, i.e. si et seulement si f est une combinaison polynomiale des éléments de EQNS.

Le degré est celui retourné par NC_DEGREE. Ceci est à son tour influencé par le poids des variables individuelles.

DECLARE_WEIGHT (*var1,wt1,var2,wt2,...*) Fonction

Affecte à VAR1 le poids wt1, à VAR2 le poids wt2.. Ce sont les poids utilisés dans les calculs de NC_DEGREE.

NC_DEGREE (*poly*) Fonction

Degré d'un polynôme non commutatif. Voir DECLARE_WEIGHTS.

DOTSIMP (*f*) Fonction

==> 0 si et seulement si f est dans l'idéal généré par les EQNS, i.e. si et seulement si f est une combinaison polynomiale des éléments de EQNS.

FAST_CENTRAL_ELEMENTS (*variables,degree*) Fonction

Si SET_UP_DOT_SIMPLIFICATIONS a été précédemment exécutée, cherche les polynômes centraux des variables dans le degré donné. Par exemple :

```
set_up_dot_simplifications([y.x+x.y],3);
fast_central_éléments([x,y],2);
[y.y,x.x];
```

CHECK_OVERLAPS (*degree,add-to-simps*) Fonction

vérifie le recouvrement par "degree", en s'assurant que vous avez suffisamment de règles de simplification pour chaque degré, afin que dotsimp fonctionne correctement. Ce processus peut être accéléré si vous connaissez à l'avance la dimension de l'espace des monômes. Si c'est une dimension globale finie, HILBERT devra être utilisée. Si vous ne connaissez pas les dimensions monomiales, ne spécifiez pas RANK_FUNCTIION. Un troisième argument optionnel, RESET, demande de ne pas se préoccuper de réinirialiser les choses.

MONO (*vari,n*) Fonction

VARI est une liste de variables. Retourne la liste des monômes indépendents relatifs aux simplifications-point courantes, de degré N.

MONOMIAL_DIMENSIONS (*n*) Fonction

Calcule la suite de Hilbert de degré n dans l'algèbre courante.

EXTRACT_LINEAR_EQUATIONS (*List_nc_polys,monoms*) Fonction

Prend une liste des coefficients des polynômes dans list_nc_polys des monômes. monoms est une liste de monômes non commutatifs. Les coefficients devront être des scalaires. Utilisez LIST_NC_MONOMIALS pour construire la liste des monômes.

LIST_NC_MONOMIALS (*polys_or_list*) Fonction

retourne une liste des monômes non commutatifs se trouvant dans un polynôme ou une collection de polynômes.

PCOEFF (*poly monom* [*variables-to-exclude-from-cof* (*list-variables monom*)]) Fonction

Cette fonction est appelée au niveau Lisp, et elle utilise un format interne de polynôme.

```
CL-MAXIMA>>(setq me (st-rat #$x^2*u+y+1$))
(#:Y 1 1 0 (#:X 2 (#:U 1 1) 0 1))

CL-MAXIMA>>(pcoeff me (st-rat #$x^2$))
(#:U 1 1)
```

Règle : si une variable apparaît dans un monôme elle doit être une puissance exacte, et si elle fait partie des variables à exclure elle ne doit pas apparaître sauf si elle une puissance exacte dans le monôme. (pcoeff pol 1 ..) va exclure des variables en les substituant à zéro.

NEW-DISREP (*poly*) Fonction

Du Lisp, retourne le format maxima général d'un argument qui est sous forme "st-rat":

```
(displa(new-disrep (setq me (st-rat #$x^2*u+y+1$))))

       2
Y + U X  + 1
```

CREATE_LIST (*form,var1,list1,var2,list2,...*) Fonction

Crée une liste en évaluant FORM avec VAR1 bornée par chaque élément de LIST1, et pour chacun de ces liens lie VAR2 à chaque élément LIST2,... Le nombre d'éléments du résultat sera longueur(list1)*longueur(list2)*... Chaque VARn doit être en fait un symbole – il ne sera pas évalué. Les arguments LISTn seront évalués une fois au début de l'itération.

```
(C82) create_list1(x^i,i,[1,3,7]);
(D82) [X,X^3,X^7]
```

Avec une double itération :

```
(C79) create_list([i,j],i,[a,b],j,[e,f,h]);
(D79) [[A,E],[A,F],[A,H],[B,E],[B,F],[B,H]]
```

Au lieu de LISTn deux arguments peuvent être donnés, chacun d'eux s'évaluant à un nombre. Ils seront les limites inclusives basse et haute de l'itération.

```
(C81) create_list([i,j],i,[1,2,3],j,1,i);
(D81) [[1,1],[2,1],[2,2],[3,1],[3,2],[3,3]]
```

Notez que les limites ou la liste pour la variable j peuvent dépendre de la valeur courante de i.

ALL_DOTSIMP_DENOMS Variable

Si sa valeur est FALSE les dénominateurs rencontrés en faisant dotsimps ne seront pas collectés. Pour le faire :

```
ALL_DOTSIMP_DENOMS:[];
```

et ils seront ("nconc'd" ??) à la fin de la liste.

27 Tenseur

27.1 Introduction aux tenseurs

- Package Indicial Tensor Manipulation. Il peut être chargé par LOADFILE("itensr");. Un manuel pour les packages Tensor est disponible dans share/tensor.descr. Une démo est disponible avec DEMO("itenso.dem1");, et des démos supplémentaires se trouvent dans ("itenso.dem2"), ("itenso.dem3") et la suite.

- Il y a deux packages pour les tenseurs dans MACSYMA, CTENSR et ITENSR. CTENSR est le Component Tensor Manipulation, et est accédé par LOAD(CTENSR);. ITENSR est l'Indicial Tensor Manipulation, et est chargé par LOAD(ITENSR);. Un manuel pour CTENSR et ITENSR est disponible depuis le LCS Publications Office. Demandez MIT/LCS/TM-167. De plus, des démos existent dans le répertoire TENSOR; sous les noms de fichiers CTENSO DEMO1, DEMO2, etc. et ITENSO DEMO1, DEMO2, etc. Faites DEMO("ctenso.dem1"); ou DEMO("itenso.dem2");. Envoyez les bogues ou les commentaires à RP ou TENSOR.

27.2 Définitions pour les tenseurs

CANFORM (*exp*) Fonction

[Package Tensor] Simplifie exp en renommant les indices factices et en réarrangeant tous les indices comme dicté par les conditions de symétrie qui leur sont imposées. Si ALLSYM est TRUE tous les indices sont supposés symétriques, autrement l'information sur la symétrie fournie par les déclarations DECSYM sera utilisée. Les indices factices sont renommés de la même façon comme dans la fonction RENAME. Quand CANFORM est appliquée à une grande expression le calcul peut prendre un temps considérable. Cette durée peut être raccourcie en appelant d'abord RENAME sur l'expression. Voir aussi l'exemple de DECSYM.

Note : CANFORM peut ne pas être capable de réduire complètement une expression à sa plus simple forme, bien qu'elle retourne toujours un résultat mathématiquement correct.

CANTEN (*exp*) Fonction

[Package Tensor] Simplifies exp en renommant (voir RENAME) et en permutant les indices fictifs. CANTEN est restreinte aux sommes de produits de tenseurs dans lesquels aucune dérivée n'est présente. Comme telle elle est limitée et ne devra être utilisée que si CANFORM n'est pas capable de mener à bien la simplification requise.

CARG (*exp*) Fonction

retourne l'argument (angle de phase) de exp. à cause des conventions et restrictions, la valeur principale ne peut être garantie, sauf si exp est numérique.

COUNTER Variable

par défaut: [1] - détermine le suffixe numérique à utiliser pour créer le prochain indice factice du package tenseur. Le préfixe est déterminé par l'option DUMMYX[#].

DEFCON (*tenseur1,<tenseur2,tenseur3>*) Fonction

donne au tenseur1 la propriété que la contraction d'un produit de tenseur1 et tenseur2 donnera tenseur3 avec les indices appropriés. Avec un seul argument, tenseur1, alors la contraction du produit de tenseur1 par tout objet indicé ayant les indices appropriés (disons un tenseur) donnera un objet indicé de ce nom, i.e. tenseur, et avec un nouvel ensemble d'indices reflet des contractions exécutées. Par exemple, si METRIC: G, alors DEFCON(G) va implanter les (montées et descentes ??? raising et lowering) des indices au travers de la contraction avec le tenseur métrique. Plus d'un DEFCON peuvent être donnés pour le même objet indicé; le dernier qui applique une contraction particulière sera utilisé.

CONTRACTIONS est une liste des objets indicés auxquels DEFCON a donné des propriétés de contraction.

FLUSH (*exp,tenseur1,tenseur2,...*) Fonction

Package Tensor - met à zéro, dans exp, toutes les occurrences du tenseuri qui n'a pas d'indices dérivés (derivative indices ???).

FLUSHD (*exp,tenseur1,tenseur2,...*) Fonction

Package Tensor - met à zéro, dans exp, toutes les occurrences du tenseuri qui a des indices dérivés (derivative indices ???).

FLUSHND (*exp,tenseur,n*) Fonction

Package Tensor - met à zéro, dans exp, toutes les occurrences de l'objet tenseur différencié qui ont n ou plus d'indices dérivés comme le montre l'exemple suivant.

```
(C1) SHOW(A([I],[J,R],K,R)+A([I],[J,R,S],K,R,S));
                                 J R S      J R
(D1)                            A        + A
                                 I,K R S    I,K R
(C2) SHOW(FLUSHND(D1,A,3));
                                       J R
(D2)                                  A
                                       I,K R
```

KDELTA (*L1,L2*) Fonction

est la fonction delta généralisée de Kronecker définie dans le package Tensor avec la liste L1 des indices covariants et L2 celle des indices contravariants. KDELTA([i],[j]) retourne la delta ordinaire de Kronecker. La commande EV(EXP,KDELTA) provoque l'évaluation d'une expression contenant KDELTA([],[]) à la dimension du (collecteur manifold ???).

LC (*L*) Fonction

est le tensur de permutation (ou de Levi-Civita) qui renvoie 1 si la liste L consiste en une permutation paire d'entiers, -1 si la permutation est impaire, et 0 si certains indices sont répétés dans L.

LORENTZ (*exp*) Fonction

impose la condition de Lorentz en substituant 0 pour tous les objets indicés de exp qui ont un indice dérivé identique à un indice contravariant.

MAKEBOX (*exp*) Fonction

affiche exp de la même manière que SHOW; cependant, tout tenseur d'Alembertien se produisant dans exp sera indiqué par le symbole []. Par exemple, []P([M],[N]) représente G([],[I,J])*P([M],[N],I,J).

METRIC (*G*) Fonction

spécifie la métrique en affectant la variable METRIC:G; dans l'addition, les propriétés de contraction de la métrique G sont définies en exécutant les commandes DEFCON(G), DEFCON(G,G,KDELTA). La variable METRIC, par défaut: [], est liée à la métrique, affectée par la commande METRIC(g).

NTERMSG () Fonction

donne à l'utilisateur une image rapide de la "taille" du tenseur de Einstein. Il retourne une liste de paires dont les seconds éléments donnent le nombre de termes des composants spécifiés par les premiers éléments.

NTERMSRCI () Fonction

retourne une liste de paires, dont les seconds éléments donnent le nombre de termes du composant de RICCI spécifié par les premiers éléments. De cette façon, il est possible de trouver rapidement les expressions non zéro et de tenter la simplification.

NZETA (*Z*) Fonction

retourne la valeur complexe de la fonction de Plasma Dispersion pour le complexe Z.

```
NZETAR(Z) ==> REALPART(NZETA(Z))
```

NZETAI(Z) retourne IMAGPART(NZETA(Z)). Cette fonction est liée à la fonction d'erreur complexe par

```
NZETA(Z) = %I*SQRT(%PI)*EXP(-Z^2)*(1-ERF(-%I*Z)).
```

RAISERIEMANN (*dis*) Fonction

retourne les composants contravariants du tenseur de courbure de Riemann comme éléments d'un tableau UR[I,J,K,L]. Ils sont affichés si dis est TRUE.

RATEINSTEIN Variable

par défaut: [] - si TRUE une simplification rationnelle sera exécutée sur les composants non zéro des tenseurs de Einstein; si FACRAT:TRUE alors les composants seront aussi factorisés.

RATRIEMAN Variable

- Ce commutateur est renommé RATRIEMANN.

RATRIEMANN Variable

par défaut: [] - l'un des commutateurs qui contrôlent la simplification des tenseurs de Riemann; si TRUE, une simplification rationnelle est faite; si FACRAT:TRUE alors chacun des composants sera de plus factorisé.

REMCON (*tenseur1,tenseur2,...*) Fonction

supprime toutes les propriétés de contraction du tenseuri. REMCON(ALL) supprime toutes les propriétés de tous les objets indicés.

RICCICOM (*dis*) Fonction

[Package Tensor] - Cette fonction calcule d'abord les composants covariants LR[i,j] du tenseur de Ricci (LR est un mnémonique pour "lower Ricci"). Puis le tenseur mixte de Ricci est calculé en utilisant le tenseur métrique contravariant. Si la valeur de l'argument de RICCICOM est TRUE, ces composants mixtes, RICCI[i,j] (l'indice i est covariant et l'indice j est contravariant), seront affichés directement. Autrement, RICCICOM(FALSE) calculera simplement les entrées du tableau RICCI[i,j] sans afficher les résultats.

RINVARIANT () Fonction

[Package Tensor] forme l'invariant obtenu par contraction des tenseurs.

```
R[i,j,k,l]*UR[i,j,k,l].
```

Cet objet n'est pas automatiquement simplifié car il peut être très grand.

SCURVATURE () Fonction

retourne la courbure scalaire (obtenu par contraction du tenseur de Ricci) du (collecteur ??? manifold) riemannien avec la métrique donnée.

SETUP () Fonction

A été renommé TSETUP();. Définit une métrique pour les calculs sur les tenseurs.

WEYL (*dis*) Fonction

calcule le tenseur conforme de Weyl. Si l'argument dis est TRUE, les composants non zéro W[I,J,K,L] seront affichés pour l'utilisateur. Autrement ces composants seront simplement calculés et stockés. Si le commutateur RATWEYL est TRUE, les composants seront rationnellement simplifiés; si FACRAT est TRUE les résultats seront de plus factorisés.

28 Ctenseur

28.1 Introduction aux Ctenseurs

- Component Tensor Manipulation Package. Pour utiliser le package CTENSR, tapez TSETUP(); qui le chargera automatiquement depuis MACSYMA (s'il n'est pas déjà chargé) puis invitera l'utilisateur à entrer son système de coordonnées. Il lui est d'abord demandé de spécifier la dimension du (collecteur ??? manifold). Si la dimension est 2, 3 ou 4 alors la liste des coordonnées sera par défaut [X,Y], [X,Y,Z] ou [X,Y,Z,T] respectivement. Ces noms peuvent être changés en affectant une nouvelle liste de coordonnées à la variable OMEGA (décrite ci-dessous) et l'utilisateur est interrogé là-dessus.

** Il faut prendre garde à éviter les conflits entre les noms des coordonnées et ceux d'autres définitions d'objets **.

Ensuite, l'utilisateur entre la métrique, soit directement, soit depuis un fichier en spécifiant sa position. Comme exemple d'un fichier de métriques communes, voyez TENSOR;METRIC FILE. La métrique est stockée dans la matrice LG. Finalement, la métrique inverse est calculée et placée dans la matrice UG. Une option permet de faire tous les calculs dans une série puissance.

À titre d'exemple, un échantillon de protocole est commencé ci-dessous pour une métrique statique, à symétrie sphérique (coordonnées standard) qui sera appliquée au problème de la dérivation des équations du vide d'Einstein (qui mène à la solution de Schwarzschild). Beaucoup des fonctions de CTENSR seront affichées comme exemples pour la métrique standard.

```
(C2) TSETUP();

Entrez la dimension du système de coordonnées:
4;
Voulez-vous changer les noms des coordonnées?
N;
Voulez-vous
1. Entrer une nouvelle métrique ?
2. Entrer une métrique à partir d'un fichier ?
3. Faire une approximation d'une métrique par une série de Taylor ?
Entrez 1, 2 ou 3
1;
La matrice est-elle 1. Diagonale  2. Symétrique  3. Antisymétrique  4. Générale
Répondez par 1, 2, 3 ou 4
1;
Row 1 Column 1:  A;
Row 2 Column 2:  X^2;
Row 3 Column 3:  X^2*SIN(Y)^2;
Row 4 Column 4:  -D;
La matrice est entrée.
Entrez les dépendences fonctionnelles avec la fonction DEPENDS ou 'N' si aucune
DEPENDS([A,D],X);
Voulez-vous voir la métrique?
```

```
Y;
                              [ A  0       0        0  ]
                              [                        ]
                              [     2                  ]
                              [ 0  X       0        0  ]
                              [                        ]
                              [         2    2         ]
                              [ 0  0   X  SIN (Y)   0  ]
                              [                        ]
                              [ 0  0       0       - D ]
Voulez-vous voir la métrique inverse?
N;
```

28.2 Définitions pour les ctenseurs

CHR1 ([*i,j,k*]) Fonction

donne le symbole de Christoffel du premier ordre via la définition

```
(g      + g      - g     )/2 .
  ik,j     jk,i     ij,k
```

Pour évaluer les symboles de Christoffel d'une métrique particulière, la variable METRIC doit avoir un nom comme dans l'exemple de CHR2 ci-dessous.

CHR2 ([*i,j*],[*k*]) Fonction

donne le symbole de Christoffel du second ordre défini par la relation

```
                          ks
CHR2([i,j],[k]) = g    (g      + g      - g     )/2
                           is,j     js,i     ij,s
```

CHRISTOF (*arg*) Fonction

Une fonction du package CTENSR (Component Tensor Manipulation). Elle calcule les deux sortes des symboles de Christoffel. L'argument détermine quels résultats doivent être immédiatement affichés. Les symboles de Christoffel du premier et du second ordres sont placés dans les tableaux LCS[i,j,k] et MCS[i,j,k] respectivement et sont définis comme étant symétriques par les deux premiers indices. Si l'argument de CHRISTOF est LCS ou MCS alors l'unique valeur non zéro de LCS[i,j,k] ou MCS[i,j,k], respectivement, sera affichée. Si l'argument est ALL alors les uniques valeurs non zéro de LCS[i,j,k] et MCS[i,j,k] seront affichées. Si l'argument est FALSE l'affichage des éléments ne se fait pas. Les éléments du tableau MCS[i,j,k] sont définis de telle sorte que l'indice final est contravariant.

COVDIFF (*exp,v1,v2,...*) Fonction

donne la dérivée covariante de exp par rapport aux variables vi en termes des symboles de Christoffel du second ordre (CHR2). Afin de les évaluer, il faut utiliser EV(exp,CHR2).

CURVATURE ([*i,j,k*],[*h*]) Fonction

(Package Indicial Tensor) donne le tenseur de courbure de Riemann en termes des symboles de Christoffel du second ordre (CHR2). La notation suivante est utilisée :

```
                 h             h             h         %1         h
CURVATURE      = - CHR2        - CHR2      CHR2     + CHR2
                 i j k         i k,j         %1 j      i k        i j,k
                                 h           %1
                           + CHR2        CHR2
                                 %1 k        i j
```

DIAGMETRIC Variable

par défaut:[] - Une option du package CTENSR (Component Tensor Manipulation). Si DIAGMETRIC est TRUE des routines spéciales calculent tous les objets géométriques (qui contiennent explicitement le tenseur métrique) en prenant en considération la diagonalité de la métrique. Il en résulte une réduction du temps de calcule.

Note: cette option est automatiquement fixée par TSETUP si une métrique diagonale est spécifiée.

DIM Variable

par défaut:[4] - Une option du package CTENSR (Component Tensor Manipulation). DIM est la dimension du (collecteur ?? manifold), 4 par défaut. La commande DIM:N; redéfinit la dimension à une autre valeur entière.

EINSTEIN (*dis*) Fonction

Une fonction du package CTENSR (Component Tensor Manipulation). EINSTEIN calcule le tenseur mixte d'Einstein après que les symboles de Christoffel et le tenseur de Ricci aient été obtenus (avec les fonctions CHRISTOF et RICCICOM). Si l'argument dis est TRUE, les valeurs non zéro du tenseur mixte d'Einstein G[i,j] seront affichées, où j est l'indice contravariant. RATEINSTEIN[TRUE], si TRUE, provoque la simplification rationnelle de ces composants. Si RATFAC[FALSE] est TRUE alors les composants seront de plus factorisés.

LRICCICOM (*dis*) Fonction

Une fonction du package CTENSR (Component Tensor Manipulation). LRICCICOM calcule les composants covariants (symétriques) LR[i,j] du tenseur de Ricci. Si l'argument dis est TRUE, alors les composants non nuls sont affichés.

MOTION (*dis*) Fonction

Une fonction du package CTENSR (Component Tensor Manipulation). MOTION calcule les équations géodésiques du mouvement d'une métrique donnée. Elles sont placées dans le tableau EM[i]. Si l'argument dis est TRUE alors ces équations sont affichées.

OMEGA Variable

par défaut:[] - Une option du package CTENSR (Component Tensor Manipulation). OMEGA assigne une liste de coordonnées à la variable. Bien qu'elles soient normalement définies lorsque la fonction TSETUP est appelée, on peut redéfinir ces

coordonnées par l'affectation OMEGA:[j1,j2,...jn] où les j sont les nouveaux noms des coordonnées. Un appel à OMEGA retourne la liste des noms des coordonnées. Voir aussi DESCRIBE(TSETUP); .

RIEMANN (*dis*) Fonction

Une fonction du package CTENSR (Component Tensor Manipulation). RIEMANN calcule le tenseur de courbure de Riemann depuis une métrique donnée et les symboles de Christoffel correspondants. Si dis est TRUE, les composants non nuls R[i,j,k,l] seront affichés. Tous les indices indiqués sont covariants. Comme avec le tenseur d'Einstein, divers commutateurs fixés par l'utilisateur contrôlent la simplification des composants du tenseur de Riemann. Si RATRIEMAN[TRUE] est TRUE alors une simplification rationnelle sera effectuée. Si RATFAC[FALSE] est TRUE chacun des composants sera de plus factorisé.

TRANSFORM Fonction

La commande TRANSFORM du package CTENSR a été renommée TTRANSFORM.

TSETUP () Fonction

Une fonction du package CTENSR (Component Tensor Manipulation) qui charge automatiquement ce package CTENSR depuis MACSYMA (s'il n'est pas déjà chargé) puis invite l'utilisateur à l'utiliser. Faites DESCRIBE(CTENSR); pour plus de détails.

TTRANSFORM (*matrice*) Fonction

Une fonction du package CTENSR (Component Tensor Manipulation) qui exécute une transformation de coordonnées sur une matrice carrée symétrique arbitraire. L'utilisateur doit entrer les fonctions qui définissent la transformation. Cette fonction s'appelait précédemment TRANSFORM.

29 Séries

29.1 Introduction aux séries

Maxima contient des fonctions de Taylor et des séries puissance pour trouver les séries de fonctions différenciables. Il a aussi des outils tel que Nusum capable de trouver la forme fermée de certaines séries. Les opérations telles que l'addition et la multiplication fonctionnent comme de coutume sur les séries. Cette section présente les diverses variables globales qui contrôlent le développement.

29.2 Définitions pour les séries

CAUCHYSUM Variable

par défaut: [FALSE] - Lors de la multiplication de sommes ayant INF comme limite supérieure, si SUMEXPAND est TRUE et CAUCHYSUM est mise à TRUE alors le produit de Cauchy sera utilisé plutôt que le produit usuel. Dans le produit de Cauchy l'indice de la sommation interne est une fonction des indices de la sommation externe et ne varient donc pas indépendamment. Ainsi SUM(F(I),I,0,INF)*SUM(G(J),J,0,INF) devient SUM(SUM(F(I)*G(J-I),I,0,J),J,0,INF)

DEFTAYLOR (*fonction, exp*) Fonction

permet à l'utilisateur de définir la série de Taylor d'une fonction arbitraire d'une variable comme exp qui peut être un polynôme en cette variable ou être donnée implicitement comme série puissance avec la fonction SUM.

Afin d'afficher l'information donnée à DEFTAYLOR on peut utiliser POWERSERIES(F(X),X,0) (voir ci-dessous).

```
(C1) DEFTAYLOR(F(X),X**2+SUM(X**I/(2**I*I!**2),
       I,4,INF));
(D1)                           [F]
(C2) TAYLOR(%E**SQRT(F(X)),X,0,4);
                     2         3          4
                    X    3073 X    12817 X
(D2)/R/     1 + X + -- + ------- + -------- +  . . .
                    2     18432     307200
```

MAXTAYORDER Variable

par défaut: [TRUE] - si TRUE, pendant la manipulation algébrique de la série (tronquée) de Taylor, TAYLOR essaiera de retenir le plus possible des termes assurés d'être corrects.

NICEINDICES (*expr*) Fonction

prend l'expression et change tous les indices des sommes et produits en quelque chose de facilement compréhensible. La fonction prend chaque indice possible "I", sauf si "I" est une expression interne, auquel cas elle essaye en séquence J,K,L,M,N,I0,I1,I2,I3,I4,... jusqu'à ce qu'elle trouve un indice légal.

NICEINDICESPREF Variable

par défaut: [I,J,K,L,M,N] - la liste qu'utilise NICEINDICES pour trouver les indices des sommes et produits. Ceci permet à l'utilisateur de fixer l'ordre préférentiel pour que NICEINDICES trouve les "bons indices". Soit p. ex., NICEINDICESPREF:[Q,R,S,T,INDEX]$. Alors si NICEINDICES trouve qu'elle ne peut utiliser aucun de ceux-ci comme indices d'une sommation particulière, elle utilisera le premier comme une base à essayer et y ajoutera des nombres. Ici, si la liste est épuisée, ce sera Q0, puis Q1, etc, qui seront tentés.

NUSUM (*exp,var,low,high*) Fonction

exécute une sommation indéfinie de exp par rapport à var en utilisant une procédure de décision dûe à R.W. Gosper. exp et la réponse potentielle doivent pouvoir s'exprimer comme produits de n-ième puissances, de factorielles, de binômes, et de fonctions rationnelles. Les termes de sommation "définie" et "indéfinie" sont utilisés de façon analogue à intégration "définie" et "indéfinie". Faire une somme "indéfinie" signifie donner à une forme fermée de la somme sur des intervalles de longueur variable, pas juste p. ex. de 0 à inf. Ainsi, comme il n'y a pas de formule pour la somme partielle générale de la série binôme, NUSUM ne peut le faire.

PADE (*taylor-series,num-deg-bound,denom-deg-bound*) Fonction

retourne une liste de toutes les fonctions rationnelles qui ont le développement en série de Taylor donné, où la somme des degrés du numérateur et du dénominateur est inférieur ou égal au niveau de troncature de la série puissance, i.e. sont les "meilleurs" approximants, et qui de plus satisfont les limites de degré spécifiées. Son premier argument doit être une série de Taylor univariable; le second et le troisième sont des entiers positifs spécifiant les bornes des degrés du numérateur et du dénominateur. Le premier argument de PADE peut aussi être une série de Laurent, et les bornes peuvent alors être INF, dans ce cas toutes les fonctions rationnelles dont le degré total est inférieur ou égal à la longueur de la série puissance sont retournées. Le degré total est num-degree + denom-degree. La longueur d'une série puissance est "le niveau de troncature" + 1 - minimum(0,"ordre de la série").

```
(C15) ff:taylor(1+x+x^2+x^3,x,0,3);
                                        2    3
(D15)/T/                     1 + X + X  + X  + . . .
(C16) pade(ff,1,1);
                                          1
(D16)                                [- -----]
                                       X - 1
(c1) ff:taylor(-(83787*X^10-45552*X^9-187296*X^8
                  +387072*X^7+86016*X^6-1507328*X^5
                  +1966080*X^4+4194304*X^3-25165824*X^2
```

```
                    +67108864*X-134217728)
       /134217728,x,0,10);
(C25) PADE(ff,4,4);
(D25) []
```

Il n'y a pas de fonction rationnelle de degré 4 numérateur/dénominateur, dans ce développement en série puissance. Pour avoir suffisamment de coefficients inconnus à résoudre, il faut en général que la somme du degré du numérateur et de celui du dénominateur soit au moins égale au degré de la série puissance.

```
(C26) PADE(ff,5,5);
(D26) [-(520256329*X^5-96719020632*X^4-489651410240*X^3
           -1619100813312*X^2 -2176885157888*X-2386516803584)
/(47041365435*X^5+381702613848*X^4+1360678489152*X^3
                +2856700692480*X^2
        +3370143559680*X+2386516803584)]
```

POWERDISP Variable

par défaut: [FALSE] - si TRUE fera afficher les sommes avec leurs termes en ordre inverse. Ainsi les polynômes s'afficheront comme séries puissance tronquées, i.e., avec la plus petite puissance en premier.

POWERSERIES (*exp, var, pt*) Fonction

génère la forme générale du développement en séries puissance pour exp de la variable var autour du point pt (qui peut être INF pour infini). Si POWERSERIES est incapable de développer exp, la fonction de TAYLOR peut donner les premiers termes de la série.

VERBOSE[FALSE] - si TRUE fera afficher les commentaires sur la progression de POWERSERIES au cours de son exécution.

```
(C1) VERBOSE:TRUE$
(C2) POWERSERIES(LOG(SIN(X)/X),X,0);
Impossible développer
                                LOG(SIN(X))
Donc nous essaierons à nouveau après application de la règle :
                                      d
                                     / -- (SIN(X))
                                     [ dX
                        LOG(SIN(X)) = I ----------- dX
                                     ]   SIN(X)
                                     /
De la première simplification nous retournons :
                               /
                               [
                               I COT(X) dX - LOG(X)
                               ]
                               /
                    INF
                    ====        I1  2 I1             2 I1
```

```
                        \      (- 1)   2     BERN(2 I1) X
                         >     ------------------------------
                        /                I1 (2 I1)!
                        ====
                        I1 = 1
(D2)                    ------------------------------------
                                         2
```

PSEXPAND Variable

par défaut: [FALSE] - si TRUE provoque l'affichage des expressions des fonctions rationnelles étendues entièrement développées. (RATEXPAND le fera aussi). Si FALSE, les expressions multivariables seront affichées comme dans le package des fonctions rationnelles. Si PSEXPAND:MULTI, alors les termes des variables de même degré total seront regroupés.

REVERT (*expression,variable*) Fonction

Inverse la série de Taylor. "Variable" est la variable du développement de Taylor origine. Faites LOAD(REVERT) pour avoir accès à cette fonction. Essayez aussi

```
REVERT2(expression,variable,hipower)
```

REVERT ne fonctionne que sur les développements autour de 0.

SRRAT (*exp*) Fonction

Cette commande a été renommée TAYTORAT.

TAYLOR (*exp, var, pt, pow*) Fonction

développe l'expression exp en série de Taylor tronquée (ou en série de Laurent, au besoin) de la variable var autour du point pt. Les termes de (var-pt)**pow sont générés. Si exp est de la forme f(var)/g(var) et que g(var) n'a pas de termes jusqu'au degré pow alors TAYLOR essaiera de développer g(var) jusqu'au degré 2*pow. S'il n'y a toujours pas de termes non nuls, TAYLOR continuera de doubler le degré du développement de g(var) jusqu'à atteindre pow*2**n où n est la valeur de la variable TAYLORDEPTH[3]. Si MAXTAYORDER[FALSE] est mise à TRUE, alors pendant la manipulation algébrique de la série (tronquée) de Taylor, TAYLOR essaiera de conserver autant de termes certainement corrects. Faites EXAMPLE(TAYLOR); pour les exemples.

TAYLOR(exp,[var1,pt1,ord1],[var2,pt2,ord2],...) retourne une série puissance tronquée des variables vari autour des points pti, tronquée à ordi.

PSEXPAND[FALSE] - si TRUE fera afficher les expressions des fonctions rationnelles étendues entièrement développées. (RATEXPAND le fera aussi). Si FALSE, les expressions multivariables seront affichées comme dans le package des fonctions rationnelles. Si PSEXPAND:MULTI, alors les termes des variables de même degré total seront regroupés.

TAYLOR(exp, [var1, var2, . . .], pt, ord), où chaque pt et ord peut être remplacé par une liste qui correspondra à la liste des variables, c.-à-d. que les n-ièmes items de chaque liste seront associés.

TAYLOR(exp, [x,pt,ord,ASYMP]) donnera un développement de exp en puissances négatives de (x-pt). Le terme d'ordre le plus élevé sera (x-pt)^(-ord). ASYMP est un outil syntaxique qui n'est pas affecté. Voir aussi le commutateur TAYLOR_LOGEXPAND pour le contrôle du développement.

TAYLORDEPTH Variable

par défaut: [3] - S'il n'y a toujours pas de termes non nuls, TAYLOR continuera de doubler le degré du développement de g(var) jusqu'à atteindre pow*2**n où n est la valeur de la variable TAYLORDEPTH[3].

TAYLORINFO (*exp*) Fonction

retourne FALSE si exp n'est pas une série de Taylor. Autrement, une liste de listes est retournée, décrivant les particularités du développement de Taylor. Par exemple,

```
(C3) TAYLOR((1-Y^2)/(1-X),X,0,3,[Y,A,INF]);
             2                          2
(D3)/R/ 1 - A  - 2 A (Y - A) - (Y - A)
                     2                          2
            + (1 - A  - 2 A (Y - A) - (Y - A) ) X
          2                           2   2
 + (1 - A  - 2 A (Y - A) - (Y - A) ) X
                     2                          2   3
            + (1 - A  - 2 A (Y - A) - (Y - A) ) X
     + . . .
(C4) TAYLORINFO(D3);
(D4)                        [[Y, A, INF], [X, 0, 3]]
```

TAYLORP (*exp*) Fonction

une fonction prédicat qui retourne TRUE si et seulement si l'expression 'exp' est une représentation d'une série de Taylor.

TAYLOR_LOGEXPAND Variable

par défaut: [TRUE] - contrôle les développements des logarithmes dans les séries de TAYLOR. Si TRUE tous les log sont entièrement développés de sorte que la reconnaissance des problèmes de zéro impliquant des identités logarithmiques ne perturbe pas le processus de développement. Cependant, ce procédé n'est pas toujours mathématiquement correct car il ignore les informations de branchement. Si TAYLOR_LOGEXPAND est FALSE, le seul développement logarithmique qui se produira sera celui nécessaire à l'obtention d'une série puissance formelle.

TAYLOR_ORDER_COEFFICIENTS Variable

par défaut: [TRUE] - contrôle l'ordre des coefficients dans l'expression. Le défaut (TRUE) est tel que les coefficients de la série de Taylor seront ordonnés canoniquement.

TAYLOR_SIMPLIFIER Fonction

Une fonction d'un seul argument que TAYLOR utilise pour simplifier les coefficients des séries puissance.

TAYLOR_TRUNCATE_POLYNOMIALS Variable

par défaut: [TRUE] - Si FALSE les polynômes entrés dans TAYLOR sont considérés comme ayant une précison infinie; autrement (le défaut) ils sont tronqués, sur la base des niveaux d'entrée de la troncature.

TAYTORAT (*exp*) Fonction

convertit exp d'une forme de TAYLOR en forme CRE, i.e. cette fonction est identique à RAT(RATDISREP(exp)) mais bien plus rapide.

TRUNC (*exp*) Fonction

provoque l'affichage de exp (qui est en représentation générale) comme si ses sommes étaient des séries de Taylor tronquées. P. ex., comparez EXP1:X^2+X+1; avec EXP2:TRUNC(X^2+X+1);. Notez que IS(EXP1=EXP2); donne TRUE.

UNSUM (*fun,n*) Fonction

est la première différence arrière fun(n) - fun(n-1).

```
(C1) G(P):=P*4^N/BINOMIAL(2*N,N);
                                                N
                                              P 4
(D1)                          G(P) := ----------------
                                         BINOMIAL(2 N, N)
(C2) G(N^4);
                                           4  N
                                          N  4
(D2)                                 ----------------
                                     BINOMIAL(2 N, N)
(C3) NUSUM(D2,N,0,N);
                              4        3       2              N
          2 (N + 1) (63 N  + 112 N  + 18 N  - 22 N + 3) 4      2
(D3)      ------------------------------------------------ - ------
                          693 BINOMIAL(2 N, N)                3 11 7
(C4) UNSUM(%,N);
                                           4  N
                                          N  4
(D4)                                 ----------------
                                     BINOMIAL(2 N, N)
```

VERBOSE Variable

par défaut: [FALSE] - si TRUE provoque l'affichage des commentaires concernant la progression de POWERSERIES pendant son exécution.

30 Théorie des nombres

30.1 Définitions pour la théorie des nombres

BERN (*x*) Fonction

donne le Xième nombre de Bernoulli pour l'entier X.

ZEROBERN[TRUE] si mise à FALSE exclut les nombres de BERNOULLI nuls. (Voir aussi BURN).

BERNPOLY (*v, n*) Fonction

génère le n-ième polynôme de Bernoulli de la variable v.

BFZETA (*exp,n*) Fonction

Version BFLOAT de la fonction Zêta de Riemann. Le second argument est le nombre de chiffres à retenir et retourner, c'est une bonne idée d'en donner deux de plus. Cette fonction est disponible en faisant LOAD(BFFAC); .

BGZETA (*S, FPPREC*) Fonction

BGZETA est comme BZETA, mais elle évite les erreurs de débordement arithmétique sur les grands arguments, elle est plus rapide sur les arguments de taille moyenne (disons S=55, FPPREC=69), et un un peu plus lente sur les petits arguments. Elle peut éventuellement remplacer BZETA. BGZETA est disponible en faisant LOAD(BFAC);.

BHZETA (*S,H,FPPREC*) Fonction

donne les FPPREC chiffres de

```
SUM((K+H)^-S,K,0,INF)
```

Elle est disponible en faisant LOAD(BFFAC);.

BINOMIAL (*X, Y*) Fonction

le coefficient binomial X*(X-1)*...*(X-Y+1)/Y!. Si X et Y sont entiers, alors la valeur numérique du coefficient binomial est calculé. Si Y, ou la valeur X-Y, est un entier, le coefficient binomial est exprimé sous forme d'un polynôme.

BURN (*N*) Fonction

est comme BERN(N), mais sans calculer tous les Bernoullis de plus petits indices. Ainsi BURN fonctionne efficacement pour les grands N isolés. (BERN(402) met à peu près 645 secondes contre les 13.5 s de BURN(402). L'accroissement de temps de BERN semble être exponentiel, alors que celui de BURN est sensiblement cubique. Mais si vous faites ensuite BERN(404), il ne prendra que 12 s, car BERN conserve tout dans un tableau, alors que BURN(404) prendra 14 s ou peut-être 25, cela dépend si MACSYMA a besoin pour BFLOAT d'une meilleure valeur de %PI. BURN est disponible en faisant LOAD(BFFAC);.

BURN utilise une observation de WGD : les zêtas (transcendants) peuvent donner une approximation des nombres (rationnels) de Bernoulli avec une efficacité tolérable.

BZETA Fonction

Cette fonction est obsolète, voyez BFZETA.

CF (*exp*) Fonction

convertit exp en une fraction continuée. exp est une expression composée d'opérateurs arithmétiques et de listes qui représentent les fractions continuées. Une fraction continuée a+1/(b+1/(c+...)) est représentée par la liste [a,b,c,...]. a,b,c,.. doivent être des entiers. Exp peut aussi impliquer SQRT(n) où n est un entier. Dans ce cas CF donnera à la fraction continuée autant de termes que la valeur de la variable CFLENGTH[1] multipliée par la période. Ainsi le défaut est de donner une période (CF fixe LISTARITH à FALSE afin qu'elle puisse exécuter sa fonction).

CFDISREP (*liste*) Fonction

convertit la fraction continuée représentée par la liste en sa représentation générale.

```
(C1) CF([1,2,-3]+[1,-2,1 ]);
(D1)                  [1, 1, 1, 2]
(C2) CFDISREP(%);
                                1
(D2)                1 + ---------
                                  1
                           1 + -----
                                    1
                               1 + -
                                    2
```

CFEXPAND (*x*) Fonction

donne une matrice des numérateurs et des dénominateurs du dernier et de l'avant dernier convergents de la fraction continuée x.

```
(C1) CF(SQRT(3));
(D1)                  [1, 1, 2, 1, 2, 1, 2, 1]
(C2) CFEXPAND(%);
                                [71  97]
(D2)                            [      ]
                                [41  56]
(C3) D2[1,2]/D2[2,2],NUMER;
(D3)                           1.7321429
```

CFLENGTH Variable

par défaut: [1] - contrôle le nombre de termes de la fraction continuée que donnera la fonction CF, comme la valeur CFLENGTH[1] fois la période. Ainsi le défaut est de donner une période.

CGAMMA Fonction

La fonction Gamma dans le plan complexe. Faites LOAD(CGAMMA) pour utiliser ces fonctions Cgamma, Cgamma2, et LogCgamma2. Elles évaluent la fonction Gamma sur le plan complexe avec l'algorithme de Kuki, CACM algorithm 421. Les calculs sont exécutés en simple précision et l'erreur relative est typiquement autour de 1.0E-7; l'évaluation en un point prend moins de 1 ms. L'algorithme donne une estimation de l'erreur, mais l'implantation de Macsyma ne l'utilise pas actuellement.

Cgamma est la fonction générale et peut être appelée avec un argument symbolique ou numérique. Avec les arguments symboliques, elle les retourne tels quels; avec des arguments réels flottants ou rationnels, elle utilise la fonction Gamma de Macsyma; et pour les arguments numériques complexes, elle utilise l'algorithme de Kuki.

Cgamma2 de deux arguments, réel et imaginaire, est prévues pour les seuls arguments numériques; LogCgamma2 est la même, mais elle calcule la fonction log-gamma. Ces deux fonctions sont parfois plus efficaces.

CGAMMA2 Fonction

Voir CGAMMA.

DIVSUM (*n*,*k*) Fonction

additionne tous les facteurs de n élevé à la puissance k-ième. Si un seul argument est donné alors k est pris égal à 1.

EULER (*X*) Fonction

donne le Xième nombre d'Euler de l'entier X. Pour la constante de Euler-Mascheroni, voyez %GAMMA.

FACTORIAL (*X*) Fonction

La fonction factorielle. FACTORIAL(X) = X! . Voir aussi MINFACTORIAL et FACTCOMB. L'opérateur factoriel est !, et l'opérateur factoriel double est !!.

FIB (*X*) Fonction

le Xième nombre de Fibonacci avec FIB(0)=0, FIB(1)=1, et FIB(-N)=(-1)^(N+1) *FIB(N). PREVFIB est FIB(X-1), le nombre de Fibonacci précédant le dernier calculé.

FIBTOPHI (*exp*) Fonction

convertit FIB(n) en sa définition de forme fermée. Ceci implique la constante %PHI (= (SQRT(5)+1)/2 = 1.618033989). Si vous voulez que le package des fonctions rationnelles connaisse %PHI faites TELLRAT(%PHI^2-%PHI-1)$ ALGEBRAIC:TRUE$.

INRT (*X*,*n*) Fonction

prend deux arguments entiers, X et n, et retourne la n-ième racine entière de la valeur absolue de X.

JACOBI (*p,q*) Fonction

est le symbole de Jacobi de p et q.

LCM (*exp1,exp2,...*) Fonction

retourne le plus petit commun multiple de ses arguments. Faites LOAD(FUNCTS); pour avoir accès à cette fonction.

MAXPRIME Variable

par défaut: [489318] - le plus grand nombre pouvant être donné à la commande PRIME(n), qui retourne le n-ième premier.

MINFACTORIAL (*exp*) Fonction

cherche dans exp les occurrences de deux factorielles qui diffèrent d'un entier. Elle remplace alors l'un par un polynôme multiplié par l'autre. Si exp contient des coefficients binomiaux ils seront convertis en rapports de factoriels.

```
(C1) N!/(N+1)!;
                                    N!
(D1)                              --------
                                  (N + 1)!
(C2) MINFACTORIAL(%);
                                     1
(D2)                               -----
                                   N + 1
```

PARTFRAC (*exp, var*) Fonction

développe l'expression exp en fractions partielles par rapport à la variable principale, var. PARTFRAC fait une complète décomposition en fraction partielle. L'algorithme employé est basé sur le fait que les dénominateurs du développement de la fraction partielle (les facteurs du dénominateur originel) sont premiers relatifs. Les numérateurs peuvent être écrits comme combinaisons linéaires des dénominateurs, et le développement (falls out = se produit ou échoue ???). Voir EXAMPLE(PARTFRAC); pour des exemples.

PRIME (*n*) Fonction

donne le n-ième premier. MAXPRIME[489318] est le plus grand nombre accepté en argument. Note: la commande PRIME ne fonctionne pas dans maxima, car elle requiert un gros fichier de premiers, que ne désire pas la majorité des utilisateurs. PRIMEP travaille cependant.

PRIMEP (*n*) Fonction

retourne TRUE si n est un premier, FALSE sinon.

QUNIT (*n*) Fonction

donne l'unité principale du champ du nombre quadratique réel SQRT(n), où n est un entier, i.e. l'élément dont la norme est l'unité. Ceci revient à résoudre l'équation de Pell A**2 - n*B**2=1.

```
(C1) QUNIT(17);
(D1)                SQRT(17)+4
(C2)  EXPAND(%*(SQRT(17)-4));
(D2)                 1
```

TOTIENT (*n*) Fonction

est le nombre d'entiers inférieurs ou égaux à n qui sont premiers relatifs de n.

ZEROBERN Variable

par défaut: [TRUE] - si mise à FALSE exclut les nombres de BERNOULLI nuls (voir la fonction BERN).

ZETA (*X*) Fonction

donne la fonction zêta de Riemann pour certaines valeurs entieres de X.

ZETA%PI Variable

par défaut: [TRUE] - si FALSE, supprime ZETA(n) en donnant coeff*%PI^n pour n pair.

31 Symétries

31.1 Définitions pour les symétries

COMP2PUI (*n, l*) Fonction

réalise le passage des fonctions symétriques complètes, données dans la liste l, aux fonctions symétriques élémentaires de 0 à n. Si la liste l contient moins de n+1 éléments les valeurs formelles viennent la compléter. Le premier élément de la liste l donne le cardinal de l'alphabet s'il existe, sinon on le met égal à n.

```
COMP2PUI(3,[4,g]);
                        2          3
              [4, g, - g  + 2 h2, g  - 3 h2 g + 3 h3]
```

CONT2PART (*pc,lvar*) Fonction

rend le polynôme partitionné associé à la forme contractée pc dont les variables sont dans lvar.

```
pc : 2*a^3*b*x^4*y + x^5$
CONT2PART(pc,[x,y]);
                              3
                      [[2 a  b, 4, 1], [1, 5]]
```

Autres fonctions de changements de représentations :
CONTRACT, EXPLOSE, PART2CONT, PARTPOL, TCONTRACT, TPARTPOL.

CONTRACT (*psym,lvar*) Fonction

rend une forme contractée (i.e. un monôme par orbite sous l'action du groupe symétrique) du polynôme psym en les variables contenues dans la liste lvar. La fonction EXPLOSE réalise l'opération inverse. La fonction TCONTRACT teste en plus la symétrie du polynôme.

```
psym : EXPLOSE(2*a^3*b*x^4*y,[x,y,z]);

        3      4      3      4      3    4
     2 a  b y z  + 2 a  b x z  + 2 a  b y  z
               3    4        3      4      3    4
          + 2 a  b x  z + 2 a  b x y  + 2 a  b x  y

CONTRACT(psym,[x,y,z]);

                          3    4
                       2 a  b x  y
```

Autres fonctions de changements de représentations :
CONT2PART, EXPLOSE, PART2CONT, PARTPOL, TCONTRACT, TPARTPOL.

DIRECT ([*P1,...,Pn*],*y*,*f*,[*lvar1,...,lvarn*]) Fonction

calcule l'image directe (voir M. GIUSTI,D. LAZARD et A. VALIBOUZE, ISSAC 1988, Rome) associée à la fonction f, en les listes de variables lvar1,...,lvarn, et aux polynômes P1,...,Pn d'une variable y. l'arité de la fonction f est importante pour le calcul. Ainsi, si l'expression de f ne dépend pas d'une variable, non seulement il est inutile de donner cette variable mais cela diminue considérablement les calculs si on ne le fait pas.

```
DIRECT([z^2  - e1* z + e2, z^2  - f1* z + f2], z, b*v + a*u,
             [[u, v], [a, b]]);

                   2                          2          2
                  z  - e1 f1 z - 4 e2 f2 + e1  f2 + e2 f1

DIRECT([z^3-e1*z^2+e2*z-e3,z^2  - f1* z + f2], z, b*v + a*u,
             [[u, v], [a, b]]);

 6           5           4       2     4          2  4
Y  - 2 E1 F1 Y  - 6 E2 F2 Y  + 2 E1  F2 Y  + 2 E2 F1  Y

    2   2  4
+ E1  F1  Y

              3                   3       3        3          3  3
+ 9 E3 F1 F2 Y  + 5 E1 E2 F1 F2 Y  - 2 E1  F1 F2 Y  - 2 E3 F1  Y

    3  3       2   2  2       2      2  2     4   2  2
- 2 E1 E2 F1  Y  + 9 E2  F2  Y  - 6 E1  E2 F2  Y  + E1  F2  Y

    2     2       2   2     2       2      2     2
- 9 E1 E3 F1  F2 Y  - 6 E2  F1  F2 Y  + 3 E1  E2 F1  F2 Y

           4  2
+ 2 E1 E3 F1  Y

    2   4  2                 2         2         2
+ E2  F1  Y  - 27 E2 E3 F1 F2  Y + 9 E1  E3 F1 F2  Y

         2      2
+ 3 E1 E2  F1 F2  Y

    3         2                3            2      3
- E1  E2 F1 F2  Y + 15 E2 E3 F1  F2 Y - 2 E1  E3 F1  F2 Y

       2   3
- E1 E2  F1  F2 Y

    5          2   3                  3       3      3
- 2 E2 E3 F1  Y - 27 E3  F2  + 18 E1 E2 E3 F2  - 4 E1  E3 F2
```

```
       3   3
 - 4 E2  F2

    2   2   3        2   2   2                2   2     3      2   2
+ E1  E2  F2  + 27 E3  F1  F2  - 9 E1 E2 E3 F1  F2  + E1  E3 F1  F2

    3   2   2       2   4                 4        2   6
+ E2  F1  F2  - 9 E3  F1  F2 + E1 E2 E3 F1  F2 + E3  F1
```

Recherche du polynôme dont les racines sont les sommes a+u où a est racine de z^2 - e1* z + e2 et u est racine de z^2 - f1* z + f2

```
DIRECT([z^2  - e1* z + e2,z^2  - f1* z + f2], z,a+u,[[u],[a]]);

 4         3         3         2     2  2           2         2
Y  - 2 F1 Y  - 2 E1 Y  + 2 F2 Y  + F1  Y  + 3 E1 F1 Y  + 2 E2 Y

    2  2
+ E1  Y
                                  2                  2
- 2 F1 F2 Y - 2 E1 F2 Y - E1 F1  Y - 2 E2 F1 Y - E1  F1 Y

                 2
- 2 E1 E2 Y + F2

                           2           2                2
+ E1 F1 F2 - 2 E2 F2 + E1  F2 + E2 F1  + E1 E2 F1 + E2
```

DIRECT peut prendre deux drapeaux possibles : ELEMENTAIRES et PUISSANCES (valeur par défaut) qui permettent de décomposer les polynômes symétriques apparaissant dans ce calcul par les fonctions symétriques élémentaires ou les fonctions puissances respectivement.

Fonctions de SYM utilisées dans cette fonction :

```
MULTI_ORBIT (donc ORBIT), PUI_DIRECT, MULTI_ELEM
(donc ELEM), MULTI_PUI (donc PUI), PUI2ELE, ELE2PUI
(si le drapeau DIRECT est à PUISSANCES).
```

ELE2COMP (*m* , *l*) Fonction

passe des fonctions symétriques élémentaires aux fonctions complètes. Similaire à COMP2ELE et COMP2PUI.

Autres fonctions de changements de bases :

```
COMP2ELE, COMP2PUI, ELE2PUI, ELEM, MON2SCHUR, MULTI_ELEM,
MULTI_PUI, PUI, PUI2COMP, PUI2ELE, PUIREDUC, SCHUR2COMP.
```

ELE2POLYNOME (*l,z*) Fonction

donne le polynôme en z dont les fonctions symétriques élémentaires des racines sont dans la liste l. l=[n,e1,...,en] où n est le degré du polynôme et ei la i-ième fonction symétrique élémentaire.

```
ele2polynome([2,e1,e2],z);

                                  2
                                 Z  - E1 Z + E2

polynome2ele(x^7-14*x^5  + 56*x^3  - 56*X + 22,x);

              [7, 0, - 14, 0, 56, 0, - 56, - 22]
ele2polynome( [7, 0, - 14, 0, 56, 0, - 56, - 22],x);

                          7       5       3
                         X  - 14 X  + 56 X  - 56 X + 22
```

la réciproque : POLYNOME2ELE(p,z)

Autres fonctions à voir :

POLYNOME2ELE, PUI2POLYNOME.

ELE2PUI (*m, l*) Fonction

passe des fonctions symétriques élémentaires aux fonctions complètes. Similaire à COMP2ELE et COMP2PUI.

Autres fonctions de changements de bases :

`COMP2ELE, COMP2PUI, ELE2COMP, ELEM, MON2SCHUR, MULTI_ELEM, MULTI_PUI, PUI, PUI2COMP, PUI2ELE, PUIREDUC, SCHUR2COMP.`

ELEM (*ele,sym,lvar*) Fonction

décompose le polynôme symétrique sym, en les variables contenues dans la liste lvar, par les fonctions symétriques élémentaires contenues dans la liste ele. Si le premier élément de ele est donné ce sera le cardinal de l'alphabet sinon on prendra le degré du polynôme sym. S'il manque des valeurs à la liste ele des valeurs formelles du type "ei" sont rajoutées. Le polynôme sym peut être donné sous 3 formes différentes : contractée (ELEM doit alors valoir 1, sa valeur par défaut), partitionnée (ELEM doit alors valoir 3) ou étendue (i.e. le polynôme en entier) (ELEM doit alors valoir 2). L'utilisation de la fonction PUI se réalise sur le même modèle.

Sur un alphabet de cardinal 3 avec e1, la première fonction symétrique élémentaire, valant 7, le polynôme symétrique en 3 variables dont la forme contractée (ne dépendant ici que de deux de ses variables) est x^4-2*x*y se décompose ainsi en les fonctions symétriques élémentaires :

```
ELEM([3,7],x^4-2*x*y,[x,y]);
```

```
                                  2
                    28 e3 + 2 e2  - 198 e2 + 2401
```

Autres fonctions de changements de bases :

COMP2ELE, COMP2PUI, ELE2COMP, ELE2PUI, MON2SCHUR, MULTI_ELEM, MULTI_PUI, PUI, PUI2COMP, PUI2ELE, PUIREDUC, SCHUR2COMP.

EXPLOSE (*pc,lvar*) Fonction

rend le polynôme symétrique associé à la forme contractée pc. La liste lvar contient les variables.

```
EXPLOSE(a*x +1,[x,y,z]);
```

```
                          (x + y + z) a + 1
```

Autres fonctions de changements de représentations :

CONTRACT, CONT2PART, PART2CONT, PARTPOL, TCONTRACT, TPARTPOL.

KOSTKA (*part1,part2*) Fonction

(écrite par P. ESPERET) calcule le nombre de Kostka associé aux partition part1 et part2

```
kostka([3,3,3],[2,2,2,1,1,1]);
                                       6
```

LGTREILLIS (*n,m*) Fonction

rend la liste des partitions de poids n et de longueur m.

LGTREILLIS(4,2);

[[3, 1], [2, 2]]

Voir également : LTREILLIS, TREILLIS et TREINAT.

LTREILLIS (*n,m*) Fonction

rend la liste des partitions de poids n et de longueur inférieure ou égale à m.

```
 ltreillis(4,2);
```

```
                              [[4, 0], [3, 1], [2, 2]]
```

Voir également : LGTREILLIS, TREILLIS et TREINAT.

MON2SCHUR (*l*) Fonction

la liste l représente la fonction de Schur S_l : On a l=[i1,i2,...,iq] avec i1 <= i2 <= ... <= iq . La fonction de Schur S_[i1,i2...,iq] est le mineur de la matrice infinie (h_{i-j}) i>=1, j>=1 composée des q premières lignes et des colonnes i1+1,i2+2,...,iq+q.

On écrit cette fonction de Schur en fonction des formes monomiales en utilisant les fonctions TREINAT et KOSTKA. La forme rendue est un polynôme symétrique dans une de ses représentations contractées avec les variables x1, x2, ...

```
mon2schur([1,1,1]);

                                X1 X2 X3

mon2schur([3]);

                                          2        3
                           X1 X2 X3 + X1  X2 + X1
MON2SCHUR([1,2]);
    2
       2 x1 x2 x3 + x1  x2
```

ce qui veut dire que pour 3 variables cela donne :

```
  2 x1 x2 x3 + x1^2 x2 + x2^2 x1 + x1^2 x3 + x3^2 x1
   + x2^2 x3 + x3^2 x2
```

Autres fonctions de changements de bases :

`COMP2ELE, COMP2PUI, ELE2COMP, ELE2PUI, ELEM, MULTI_ELEM, MULTI_PUI, PUI, PUI2COMP, PUI2ELE, PUIREDUC, SCHUR2COMP.`

MULTI_ELEM (*l_elem,multi_pc,l_var*) Fonction

décompose un polynôme multi-symétrique sous la forme multi-contractée multi_pc en les groupes de variables contenues dans la liste de listes l_var sur les groupes de fonctions symétriques élémentaires contenues dans l_elem.

MULTI_ELEM([[2,e1,e2],[2,f1,f2]],a*x+a^2+x^3,[[x,y],[a,b]]);

2 3 - 2 f2 + f1 + e1 f1 - 3 e1 e2 + e1

Autres fonctions de changements de bases :

`COMP2ELE, COMP2PUI, ELE2COMP, ELE2PUI, ELEM, MON2SCHUR, MULTI_PUI, PUI, PUI2COMP, PUI2ELE, PUIREDUC, SCHUR2COMP.`

MULTI_ORBIT (*P,[lvar1, lvar2,...,lvarp]*) Fonction

P est un polynôme en l'ensemble des variables contenues dans les listes lvar1, lvar2 ... lvarp. Cette fonction ramène l'orbite du polynôme P sous l'action du produit des groupes symétriques des ensembles de variables représentés par ces p LISTES.

```
MULTI_ORBIT(a*x+b*y,[[x,y],[a,b]]);

          [b y + a x, a y + b x]

multi_orbit(x+y+2*a,[[x,y],[a,b,c]]);

              [Y + X + 2 C, Y + X + 2 B, Y + X + 2 A]
```

Voir également : ORBIT pour l'action d'un seul groupe symétrique

MULTI_PUI Fonction

est à la fonction PUI ce que la fonction MULTI_ELEM est à la fonction ELEM.

```
MULTI_PUI([[2,p1,p2],[2,t1,t2]],a*x+a^2+x^3,[[x,y],[a,b]]);

                                      3
                                3 P1 P2   P1
                    T2 + P1 T1 + ------- - ---
                                   2       2
```

MULTINOMIAL (*r,part*) Fonction

où r est le poids de la partition part. Cette fonction ramène le coefficient multinomial associé : si les parts de la partitions part sont i1, i2, ..., ik, le résultat de MULTINOMIAL est r!/(i1!i2!...ik!).

MULTSYM (*ppart1, ppart2,N*) Fonction

réalise le produit de deux polynômes symétriques de N variables en ne travaillant que modulo l'action du groupe symétrique d'ordre N. Les polynômes sont dans leur représentation partitionnée.

Soient les 2 polynômes symétriques en x, y : 3*(x+y) + 2*x*y et 5*(x^2+y^2) dont les formes partitionnées sont respectivement [[3,1],[2,1,1]] et [[5,2]], alors leur produit sera donné par :

```
MULTSYM([[3,1],[2,1,1]],[[5,2]],2);

          [[10, 3, 1], [15, 2, 1], [15, 3, 0]]
```

soit 10*(x^3*y+y^3*x)+15*(x^2*y +y^2*x) +15(x^3+y^3)

Fonctions de changements de représentations d'un polynôme symétrique :

CONTRACT, CONT2PART, EXPLOSE, PART2CONT, PARTPOL, TCONTRACT, TPARTPOL.

ORBIT (*P,lvar*) Fonction

calcule l'orbite du polynôme P en les variables de la liste lvar sous l'action du groupe symétrique de l'ensemble des variables contenues dans la liste lvar.

```
orbit(a*x+b*y,[x,y]);
                          [A Y + B X, B Y + A X]
orbit(2*x+x^2,[x,y]);
                                 2         2
                               [Y  + 2 Y, X  + 2 X]
```

Voir également : MULTI_ORBIT pour l'action d'un produit de groupes symétriques sur un polynôme.

PART2CONT (*ppart,lvar*) Fonction

passe de la forme partitionnée à la forme contractée d'un polynôme symétrique. La forme contractée est rendue avec les variables contenues dans lvar.

```
PART2CONT([[2*a^3*b,4,1]],[x,y]);
                                3    4
                             2 a  b x  y
```

Autres fonctions de changements de représentations :

CONTRACT, CONT2PART, EXPLOSE, PARTPOL, TCONTRACT, TPARTPOL.

PARTPOL (*psym, lvar*) Fonction

psym est un polynôme symétrique en les variables de lvar. Cette fonction ramène sa représentation partitionnée.

```
PARTPOL(-a*(x+y)+3*x*y,[x,y]);
                          [[3, 1, 1], [- a, 1, 0]]
```

Autres fonctions de changements de représentations :

CONTRACT, CONT2PART, EXPLOSE, PART2CONT, TCONTRACT, TPARTPOL.

PERMUT (*l*) Fonction

ramène la liste des permutations de la liste l.

POLYNOME2ELE (*p,x*) Fonction

donne la liste l=[n,e1,...,en] où n est le degré du polynôme p en la variable x et ei la i-ième fonction symétrique élémentaire des racines de p.

```
POLYNOME2ELE(x^7-14*x^5  + 56*x^3  - 56*X + 22,x);

             [7, 0, - 14, 0, 56, 0, - 56, - 22]

ELE2POLYNOME( [7, 0, - 14, 0, 56, 0, - 56, - 22],x);

                          7       5       3
                         X  - 14 X  + 56 X  - 56 X + 22
```

la réciproque : ELE2POLYNOME(l,x)

PRODRAC (*L,K*) Fonction

L est une liste contenant les fonctions symétriques élémentaires sur un ensemble A. PRODRAC rend le polynôme dont les racines sont les produits K à K des éléments de A.

PUI (*pui,sym,lvar*) Fonction

décompose le polynôme symétrique sym, en les variables contenues dans la liste lvar, par les fonctions puissances contenues dans la liste pui. Si le premier élément de pui est donné ce sera le cardinal de l'alphabet sinon on prendra le degré du polynôme sym. S'il manque des valeurs à la liste pui, des valeurs formelles du type "pi" sont rajoutées. Le polynôme sym peut être donné sous 3 formes différentes : contractée (PUI doit alors valoir 1 sa valeur par défaut), partitionnée (PUI doit alors valoir 3) ou étendue (i.e. le polynôme en entier) (PUI doit alors valoir 2). La fonction ELEM s'utilise de la même manière.

```
PUI;

       1
PUI([3,a,b],u*x*y*z,[x,y,z]);

       3
     (a  - 3 b a + 2 p3) u
              ---------------------
      6
```

Autres fonctions de changements de bases :

`COMP2ELE, COMP2PUI, ELE2COMP, ELE2PUI, ELEM, MON2SCHUR, MULTI_ELEM, MULTI_PUI, PUI2COMP, PUI2ELE, PUIREDUC, SCHUR2COMP.`

PUI2COMP (*N,LPUI*) Fonction

rend la liste des N premières fonctions complètes (avec en tête le cardinal) en fonction des fonctions puissance données dans la liste LPUI. Si la liste LPUI est vide le cardinal est N sinon c'est son premier élément. Similaire à COMP2ELE et COMP2PUI.

```
PUI2COMP(2,[]);

2
      p1  + p2
                [2, p1, --------]
 2

PUI2COMP(3,[2,a1]);

     2         3
   a1  + p2  a1  + 3 p2 a1 + 2 p3
               [2, a1, --------, --------------------]
      2         6
```

Autres fonctions de changements de bases :

```
COMP2ELE, COMP2PUI, ELE2COMP, ELE2PUI, ELEM,
MON2SCHUR, MULTI_ELEM, MULTI_PUI, PUI, PUI2ELE,
PUIREDUC, SCHUR2COMP.
```

PUI2ELE (*N,LPUI*) Fonction

réalise le passage des fonctions puissances aux fonctions symétriques élémentaires. Si le drapeau PUI2ELE est GIRARD, on récupère la liste des fonctions symétriques élémentaires de 1 à N, et s'il est égal à CLOSE, la Nième fonction symétrique élémentaire.

Autres fonctions de changements de bases :

```
COMP2ELE, COMP2PUI, ELE2COMP, ELE2PUI, ELEM,
MON2SCHUR, MULTI_ELEM, MULTI_PUI, PUI, PUI2COMP,
PUIREDUC, SCHUR2COMP.
```

PUI2POLYNOME (*X,LPUI*) Fonction

calcule le polynôme en X dont les fonctions puissances des racines sont données dans la liste LPUI.

```
(C6) polynome2ele(x^3-4*x^2+5*x-1,x);
(D6)                              [3, 4, 5, 1]
(C7) ele2pui(3,%);
(D7)                              [3, 4, 6, 7]
(C8) pui2polynome(x,%);
                                3      2
(D8)                           X  - 4 X  + 5 X - 1
```

Autres fonctions à voir :

POLYNOME2ELE, ELE2POLYNOME.

PUI_DIRECT (*ORBITE,*[*lvar1,...,lvarn*],[*d1,d2,...,dn*]) Fonction

Soit f un polynôme en n blocs de variables lvar1,...,lvarn. Soit ci le nombre de variables dans lvari . Et SC le produit des n groupes symétriques de degré c1,...,cn. Ce groupe agit naturellement sur f. La liste ORBITE est l'orbite, notée SC(f), de la fonction f sous l'action de SC. (Cette liste peut être obtenue avec la fonction : MULTI_ORBIT). Les di sont des entiers tels que c1<=d1, c2<=d2,...,cn<=dn. Soit SD le produit des groupes symétriques S_d1 x S_d2 x...x S_dn.

La fonction PUI_DIRECT ramène les N premières fonctions puissances de SD(f) déduites des fonctions puissances de SC(f) où N est le cardinal de SD(f).

Le résultat est rendu sous forme multi-contractée par rapport a SD, i.e. on ne conserve qu'un élément par orbite sous l'action de SD.

```
L:[[x,y],[a,b]]$

PUI_DIRECT(MULTI_ORBIT(a*x+b*y, L), L,[2,2]);

                                  2  2
                [a x, 4 a b x y + a  x ]

PUI_DIRECT(MULTI_ORBIT(a*x+b*y, L), L,[3,2]);

                        2  2     2    2        3  3
  [2 A X, 4 A B X Y + 2 A  X , 3 A  B X  Y + 2 A  X ,

    2  2  2  2      3    3        4  4
12 A  B  X  Y  + 4 A  B X  Y + 2 A  X ,

    3  2  3  2      4    4        5  5
10 A  B  X  Y  + 5 A  B X  Y + 2 A  X ,

    3  3  3  3       4  2  4  2      5    5        6  6
40 A  B  X  Y  + 15 A  B  X  Y  + 6 A  B X  Y + 2 A  X ]

 PUI_DIRECT([y+x+2*c, y+x+2*b, y+x+2*a],[[x,y],[a,b,c]],[2,3]);

                             2             2
      [3 x + 2 a, 6 x y + 3 x  + 4 a x + 4 a ,

             2                  3       2       2       3
          9 x  y + 12 a x y + 3 x  + 6 a x  + 12 a  x + 8 a ]


PUI_DIRECT([y+x+2*c, y+x+2*b, y+x+2*a],[[x,y],[a,b,c]],[3,4]);
```

PUIREDUC (*N,LPUI*) Fonction

LPUI est une liste dont le premier élément est un entier M. PUIREDUC donne les N premières fonctions puissances en fonction des M premières.

```
PUIREDUC(3,[2]);

  3
      3 p1 p2 - p1
            [2, p1, p2, -------------]

    2
```

RESOLVANTE (*p,x,f,*[*x1,...,xd*]) Fonction

calcule la résolvante du polynôme p de la variable x et de degré n >= d par la fonction f exprimée en les variables x1,...,xd. Il est important pour l'efficacité des calculs de ne pas mettre dans la liste [x1,...,xd] les variables n'intervenant pas dans la fonction de transformation f.

Afin de rendre plus efficaces les calculs on peut mettre des drapeaux à la variable RESOLVANTE afin que des algorithmes adéquats soient utilisés :

Si la fonction f est unitaire :

- un polynôme d'une variable,
- linéaire ,
- alternée,
- une somme de variables,
- symétrique en les variables qui apparaissent dans son expression,
- un produit de variables,
- la fonction de la résolvante de Cayley (utilisable seulement en degré 5)

```
(x1*x2+x2*x3+x3*x4+x4*x5+x5*x1 -
     (x1*x3+x3*x5+x5*x2+x2*x4+x4*x1))^2
```

  generale,

le drapeau de RESOLVANTE pourra être respectivement :

- unitaire,
- linéaire,
- alternée,
- somme,
- produit,
- cayley,
- générale.

```
 resolvante:unitaire;
resolvante(x^7-14*x^5  + 56*x^3  - 56*X + 22,x,x^3-1,[x]);

  7      6        5         4          3           2
Y  + 7 Y  - 539 Y  - 1841 Y  + 51443 Y  + 315133 Y  + 376999 Y
```

```
 + 125253

resolvante : lineaire;
resolvante(x^4-1,x,x1+2*x2+3*x3,[x1,x2,x3]);

 24       20         16            12             8              4
Y   + 80 Y   + 7520 Y   + 1107200 Y   + 49475840 Y  + 344489984 Y
 + 655360000
       Meme solution pour :
resolvante : general;
resolvante(x^4-1,x,x1+2*x2+3*x3,[x1,x2,x3]);
resolvante(x^4-1,x,x1+2*x2+3*x3,[x1,x2,x3,x4])
direct([x^4-1],x,x1+2*x2+3*x3,[[x1,x2,x3]]);

resolvante:lineaire$
resolvante(x^4-1,x,x1+x2+x3,[x1,x2,x3);

       4
      Y  - 1

resolvante:symetrique$

resolvante(x^4-1,x,x1+x2+x3,[x1,x2,x3]);

       4
      Y  - 1
resolvante(x^4+x+1,x,x1-x2,[x1,x2]);
  12      8       6        4        2
 Y   + 8 Y  + 26 Y  - 112 Y  + 216 Y  + 229

resolvante:alternee$
resolvante(x^4+x+1,x,x1-x2,[x1,x2]);

  12      8       6        4        2
 Y   + 8 Y  + 26 Y  - 112 Y  + 216 Y  + 229


resolvante:produit;
resolvante(x^7-7*x+3,x,x1*x2*x3,[x1,x2,x3]);

   35      33         29        28         27        26         24
  Y   - 7 Y   - 1029 Y   + 135 Y   + 7203 Y   - 756 Y   + 1323 Y

          23          22            21           20          19
+ 352947 Y   - 46305 Y   - 2463339 Y   + 324135 Y   - 30618 Y

  18
- 453789 Y
```

```
    17              15             14              12
- 40246444 Y   + 282225202 Y   - 44274492 Y   + 155098503 Y

            11
+ 12252303 Y

   10              9             8             7             6
+ 2893401 Y   - 171532242 Y  + 6751269 Y  + 2657205 Y  - 94517766 Y

   5             3
- 3720087 Y  + 26040609 Y  + 14348907

   resolvante:symetrique$
  resolvante(x^7-7*x+3,x,x1*x2*x3,[x1,x2,x3]);

  35      33         29        28         27        26         24
 Y   - 7 Y   - 1029 Y   + 135 Y   + 7203 Y   - 756 Y   + 1323 Y

          23          22            21           20          19
+ 352947 Y   - 46305 Y   - 2463339 Y   + 324135 Y   - 30618 Y

           18
 - 453789 Y

            17              15             14              12
- 40246444 Y   + 282225202 Y   - 44274492 Y   + 155098503 Y

             11
 + 12252303 Y

           10              9             8             7             6
+ 2893401 Y   - 171532242 Y  + 6751269 Y  + 2657205 Y  - 94517766 Y

           5             3
- 3720087 Y  + 26040609 Y  + 14348907

resolvante:cayley$
resolvante(x^5-4*x^2+x+1,x,a,[]);

" resolvante de Cayley "

 6       5         4          3           2
X  - 40 X  + 4080 X  - 92928 X  + 3772160 X  + 37880832 X + 93392896
```

Pour la résolvante de Cayley, les 2 derniers arguments sont neutres et le polynôme donné en entrée doit nécessairement être de degré 5.

Voir également :

RESOLVANTE_BIPARTITE, RESOLVANTE_PRODUIT_SYM,
RESOLVANTE_UNITAIRE, RESOLVANTE_ALTERNEE1, RESOLVANTE_KLEIN,

RESOLVANTE_KLEIN3, RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE.

RESOLVANTE_ALTERNEE1 (*p,x*) Fonction

calcule la transformation de p(x) de degré n par la fonction $\prod_{1\leq i<j\leq n-1} (x_i-x_j)$.

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE_UNITAIRE,
RESOLVANTE , RESOLVANTE_KLEIN, RESOLVANTE_KLEIN3,
RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE, RESOLVANTE_BIPARTITE.

RESOLVANTE_BIPARTITE (*p,x*) Fonction

calcule la transformation de p(x) de degré n (n pair) par la fonction $x_1x_2\ldots x_{n/2}+x_{n/2+1}\ldotsx_n$

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE_UNITAIRE,
RESOLVANTE , RESOLVANTE_KLEIN, RESOLVANTE_KLEIN3,
RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE,RESOLVANTE_ALTERNEE1

```
 RESOLVANTE_BIPARTITE(x^6+108,x);

                 10        8           6             4
                Y   - 972 Y  + 314928 Y  - 34012224 Y
```

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE_UNITAIRE,
RESOLVANTE, RESOLVANTE_KLEIN, RESOLVANTE_KLEIN3,
RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE,
RESOLVANTE_ALTERNEE1.

RESOLVANTE_DIEDRALE (*p,x*) Fonction

calcule la transformation de p(x) par la fonction x_1x_2+x_3x_4.

```
resolvante_diedrale(x^5-3*x^4+1,x);

 15       12       11       10        9         8         7        6
X   - 21 X   - 81 X   - 21 X   + 207 X  + 1134 X  + 2331 X  - 945 X

           5          4          3          2
   - 4970 X  - 18333 X  - 29079 X  - 20745 X  - 25326 X - 697
```

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE_UNITAIRE,
RESOLVANTE_ALTERNEE1, RESOLVANTE_KLEIN, RESOLVANTE_KLEIN3,
RESOLVANTE_VIERER, RESOLVANTE.

RESOLVANTE_KLEIN (*p,x*) Fonction

calcule la transformation de p(x) par la fonction x_1x_2x_4+x_4.

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE_UNITAIRE,
RESOLVANTE_ALTERNEE1, RESOLVANTE, RESOLVANTE_KLEIN3,
RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE.

RESOLVANTE_KLEIN3 (*p,x*) Fonction

calcule la transformation de p(x) par la fonction x_1x_2x_4+x_4.

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE_UNITAIRE,
RESOLVANTE_ALTERNEE1, RESOLVANTE_KLEIN, RESOLVANTE,
RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE.

RESOLVANTE_PRODUIT_SYM (*p,x*) Fonction

calcule la liste de toutes les résolvantes produit du polynôme p(x).

```
  resolvante_produit_sym(x^5+3*x^4+2*x-1,x);

  5      4             10      8       7       6       5    4
[Y  + 3 Y  + 2 Y - 1, Y   - 2 Y  - 21 Y  - 31 Y  - 14 Y  - Y

      3
+ 14 Y

      2       10      8       7    6       5       4       3      2
+ 3 Y  + 1, Y   + 3 Y  + 14 Y  - Y  - 14 Y  - 31 Y  - 21 Y  - 2 Y

      5      4
+ 1, Y  - 2 Y  - 3 Y - 1, Y - 1]


resolvante:produit$
resolvante(x^5+3*x^4+2*x-1,x,a*b*c,[a,b,c]);

 10      8       7    6       5       4       3      2
Y   + 3 Y  + 14 Y  - Y  - 14 Y  - 31 Y  - 21 Y  - 2 Y  + 1
```

Voir également :

RESOLVANTE, RESOLVANTE_UNITAIRE,
RESOLVANTE_ALTERNEE1, RESOLVANTE_KLEIN, RESOLVANTE_KLEIN3,

RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE.

RESOLVANTE_UNITAIRE (*p,q,x*) Fonction

calcule la résolvante du polynôme p(x) par le polynôme q(x).

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE,
RESOLVANTE_ALTERNEE1, RESOLVANTE_KLEIN, RESOLVANTE_KLEIN3,
RESOLVANTE_VIERER, RESOLVANTE_DIEDRALE.

RESOLVANTE_VIERER (*p,x*) Fonction

calcule la transformation de p(x) par la fonction x_1x_2-x_3x_4.

Voir également :

RESOLVANTE_PRODUIT_SYM, RESOLVANTE_UNITAIRE,
RESOLVANTE_ALTERNEE1, RESOLVANTE_KLEIN, RESOLVANTE_KLEIN3,
RESOLVANTE, RESOLVANTE_DIEDRALE.

SCHUR2COMP (*P,l_var*) Fonction

P est un polynôme en les variables contenues dans la liste l_var. Chacune des variables de l_var représente une fonction symétrique complète. On représente dans l_var la i-ème fonction symétrique complète comme la concaténation de la lettre h avec l'entier i : hi. Cette fonction donne l'expression de P en fonction des fonctions de Schur.

```
SCHUR2COMP(h1*h2-h3,[h1,h2,h3]);


 s
1, 2

SCHUR2COMP(a*h3,[h3]);

  s  a
 3
```

SOMRAC (*liste,K*) Fonction

la liste contient les fonctions symétriques élémentaires d'un polynôme P . On calcule le polynôme dont les racines sont les sommes K à K distinctes des racines de P.

Voir également PRODRAC.

TCONTRACT (*pol,lvar*) Fonction

teste si le polynôme pol est symétrique en les variables contenues dans la liste lvar. Si oui il rend une forme contractée comme la fonction CONTRACT.

Autres fonctions de changements de représentations :

CONTRACT, CONT2PART, EXPLOSE, PART2CONT, PARTPOL, TPARTPOL.

TPARTPOL (*pol,lvar*) Fonction

teste si le polynôme pol est symétrique en les variables contenues dans la liste lvar. Si oui il rend sa forme partitionnée comme la fonction PARTPOL.

Autres fonctions de changements de représentations :

CONTRACT, CONT2PART, EXPLOSE, PART2CONT, PARTPOL, TCONTRACT.

TREILLIS (*n*) Fonction

ramène toutes les partitions de poids n.

```
treillis(4);

           [[4], [3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]
```

Voir également : LGTREILLIS, LTREILLIS et TREINAT.

TREINAT Fonction

TREINAT(part) ramène la liste des partitions inférieures à la partition part pour l'ordre naturel.

```
treinat([5]);

                              [[5]]
treinat([1,1,1,1,1]);

 [[5], [4, 1], [3, 2], [3, 1, 1], [2, 2, 1], [2, 1, 1, 1],

  [1, 1, 1, 1, 1]]

treinat([3,2]);

                      [[5], [4, 1], [3, 2]]
```

Voir également : LGTREILLIS, LTREILLIS et TREILLIS.

32 Groupes

32.1 Définitions pour les groupes

TODD_COXETER (*relations,subgroup*) Fonction

Cherche l'ordre de G/H où G est le groupe libre modulo RELATIONS, et H est le sous-groupe de G généré par SUBGROUP. SUBGROUP est un argument optionnel, vide [] par défaut. Ceci produit une table de multiplication pour l'action droite de G sur G/H, où les co-ensembles sont énumérés [H,Hg2,Hg3,...]. Ce peut être vu en interne avec le $todd_coxeter_state. Les tables de multiplication des variables sont dans table:todd_coxeter_state[2]. Puis table[i] donne la table pour la i-ème variable. mulcoset(coset,i) := table[varnum][coset];

Exemple:

```
(C1) symet(n):=create_list(if (j - i) = 1 then (p(i,j))^^3 else
     if (not i = j) then (p(i,j))^^2 else p(i,i) , j,1,n-1,i,1,j);
      <3>
(D1) SYMET(N) := CREATE_LIST(IF J - I = 1 THEN P(I, J)

       <2>
 ELSE (IF NOT I = J THEN P(I, J)
 ELSE P(I, I)), J, 1, N - 1, I, 1, J)
(C2) p(i,j) :=concat(x,i).concat(x,j);
(D2)                P(I, J) := CONCAT(X, I) . CONCAT(X, J)
(C3) symet(5);
                         <3>                     <2>           <3>
(D3) [X1 . X1, (X1 . X2)   , X2 . X2, (X1 . X3)   , (X2 . X3)   ,

                       <2>                <2>           <3>
X3 . X3,     (X1 . X4)  , (X2 . X4)   , (X3 . X4)   , X4 . X4]
(C4) todd_coxeter(d3);

Rows tried 426
(D4)        120
(C5) todd_coxeter(d3,[x1]);

Rows tried 213
(D5)        60
(C6) todd_coxeter(d3,[x1,x2]);

Rows tried 71
(D6)        20
(C7) table:todd_coxeter_state[2]$
(C8) table:todd_coxeter_state[2]$
(C9) table[1];
(D9) {Array: FIXNUM #(0 2 1 3 7 6 5 4 8 11 17 9 12 14 13 20
           16 10 18 19 15 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0)}
```

Notez que seuls les éléments 1 à 20 de ce tableau d9 ont un sens. table[1][4] = 7 indique que coset4.var1 = coset7.

33 Environnement d'exécution

33.1 Introduction à l'environnement d'exécution

Un fichier est chargé automatiquement lorsque vous démarrer MACSYMA, afin de le personnaliser. Il est possible d'avoir un fichier init écrit comme un fichier BATCH de commandes macsyma. Nous espérons que cela rend plus facile la personnalisation de l'environnement utilisateur de macsyma. Voici un exemple de fichier init :

```
/*-*-macsyma-*-*/
setup_autoload("share\;bessel",j0,j1,jn);
showtime:all; comgrind:true;
```

Le bizarre commentaire au début du fichier "/*-*-macsyma-*-*/" annonce que c'est un fichier en langage macsyma.

"SETUP_AUTOLOAD" peut être utilisée pour rendre auto-chargeables les fonctions des fichiers BATCH, ce qui signifie que vous pouvez alors utiliser (par exemple, ici) les fonctions J0, J1 et Jn du package BESSEL directement, car lorsque vous utilisez la fonction le package BESSEL est chargé automatiquement. Si le second nom de fichier dans l'argument de SETUP_AUTOLOAD n'est pas spécifié (l'utilisation préférée) alors une recherche standard sur les seconds noms de fichier "FASL", "TRLISP", et ">" est effectuée.

33.2 Interruptions

L'utilisateur peut interrompre le calcul de MACSYMA de plusieurs façons, en général avec un caractère de contrôle. Faites DESCRIBE(CHARACTERS); pour les détails. MACSYMA sera aussi interrompu si ^Z (contrôle-Z) est frappé, ce qui fait sortir et revenir au niveau shell d'Unix. En général Contrôle-C interrompt le calcul en vous mettant dans une boucle break. Taper :t devrait vous ramener au niveau haut de maxima.

33.3 Définitions pour l'environnement d'exécution

ALARMCLOCK (*arg1, arg2, arg3*) Fonction

exécutera la fonction sans argument dont le nom est arg3 lorsque la durée spécifiée par arg1 et arg2 sera écoulée. Si arg1 est l'atome "TIME" alors arg3 sera exécutée après arg2 secondes de temps réel alors que si arg1 est l'atome "RUNTIME" arg3 sera exécutée après arg2 millisecondes de temps cpu. Si arg2 est négatif le "timer" arg1 est stoppé.

ALLOC Fonction

prend un nombre quelconque d'arguments qui sont les mêmes que les réponses à la question "run out of space" ("plus assez d'espace"). Elle augmente les allocations en conséquence. P. ex., si l'utilisateur sait au départ que son problème va demander tel espace, il peut faire ALLOC(4); pour allouer initialement le montant maximal. Voir aussi le commutateur DYNAMALLOC.

BUG ("*message*") Fonction

semblable à mail, envoie un message à MACSYMA Mail. Peut être utilisé pour signaler des bogues dans MACSYMA. Des expressions peuvent y être incluses en s'y référant, entre guillemets doubles, p. ex. BUG("J'essais d'intégrer",D3,"mais elle réclame plus d'espace. Que dois-je faire ?");.

CLEARSCREEN () Fonction

Efface l'écran, comme le fait contrôle-L.

CONTINUE Fonction

Contrôle-^ tapé dans MACSYMA fait entrer dans LISP. L'utilisateur peut maintenant taper toute S-expression de LISP et la voir évaluée. Taper (CONTINUE) ou ^G (contrôle-G) fait ré-entrer dans MACSYMA.

DDT () Fonction

Sort de MACSYMA et retourne au niveau du système d'exploitation (comme le fait contrôle-Z sur ITS, ou contrôle-C sur Tops-20).

DELFILE (*spécification-de-fichier*) Fonction

va supprimer le fichier donné par l'argument spécification-de-fichier (i.e. premier nom, second nom, périphérique, utilisateur) du périphérique (répertoire???) donné.

DISKFREE () Fonction

Sans argument, ou un argument TRUE, retourne le nombre total de blocs libres d'espace disque du système. Si l'argument est 0, 1, ou 13, elle retourne le nombre de blocs libres d'espace disque sur les packs disques respectifs. Avec un argument SECONDARY ou PRIMARY, elle retourne le nombre total de blocs libres de l'espace disque sur le disque secondaire ou primaire, respectivement.

FEATURE declaration

Un bon ajout au système. STATUS(FEATURE) vous donne une liste des particularités du système. Actuellement la liste pour MC est : MACSYMA, NOLDMSG, MACLISP, PDP10, BIGNUM, FASLOAD, HUNK, FUNARG, ROMAN, NEWIO, SFA, PAGING, MC, et ITS. L'une quelconque de ces "particularités" peut être donnée comme second argument à STATUS(FEATURE,...);. Si la particularité spécifiée existe, TRUE est retourné, sinon FALSE. Note: ce sont des propriétés du système, pas vraiment en relation avec l'utilisateur. Voir aussi DESCRIBE(features); pour d'autres particularités liées à l'utilisateur.

FEATUREP (*a,f*) Fonction

tente de déterminer si l'objet "a" a la propriété "f" sur la base des faits dans la base de données courante. Si c'est le cas, elle retourne TRUE, sinon FALSE. Voyez DESCRIBE(FEATURES);.

```
(C1) DECLARE(J,EVEN)$
(C2) FEATUREP(J,INTEGER);
(D2)                                TRUE
```

ROOM () Fonction

renvoie une description verbeuse de l'état de stockage et du traitement de la pile de Macsyma. Utilise simplement la fonction ROOM de Lisp.

ROOM(FALSE) - donne une très courte description, contenant la plupart des mêmes informations.

STATUS (*arg*) Fonction

retourne diverses informations d'état sur le MACSYMA de l'utilisateur, selon l'argument donné. Les arguments permis et les résultats sont les suivants :

- TIME - le temps utilisé jusqu'à présent dans le calcul.
- DAY - le jour de la semaine.
- DATE - une liste année, mois, et jour.
- DAYTIME - une liste heure, minute, et seconde.
- RUNTIME - temps cpu accumulé multiplié par l'atome "MILLISECONDS" du MACSYMA courant.
- REALTIME - le temps réel (en sec) écoulé depuis que l'utilisateur a lancé son MACSYMA.
- GCTIME - le temps ramasse-miettes utilisé jusqu'à présent dans le calcul courant.
- TOTALGCTIME - le temps ramasse-miettes total utilisé par MACSYMA jusqu'à présent.
- FREECORE - le nombre de blocs de noyau que peut développer votre MACSYMA avant de manquer d'espace d'adressage (un bloc est de 1024 mots). En soustrayant cette valeur de 250*BLOCKS (le maximum que vous pouvez avoir dans MC) vous apprend combien de blocs de noyau votre MACSYMA est en train d'utiliser (un MACSYMA sans fichier "fixe" commence avec à peu près 191 blocs).
- FEATURE - vous donne une liste des particularités du système. Actuellement la liste pour MC est : MACSYMA, NOLDMSG, MACLISP, PDP10, BIGNUM, FASLOAD, HUNK, FUNARG, ROMAN, NEWIO, SFA, PAGING, MC, et ITS. L'une quelconque de ces "particularités" peut être donnée comme second argument à STATUS(FEATURE,...);. Si la particularité spécifiée existe, TRUE est retourné, sinon FALSE. Note: ce sont des propriétés du système, pas vraiment en relation avec l'utilisateur.

Pour les informations à propos de vos fichiers, voyez la commande FILEDEFAULTS();.

TIME (*Di1, Di2, ...*) Fonction

donne une liste des temps en millisecondes pris pour calculer les Di.

Note: la variable SHOWTIME, par défaut: [FALSE], peut être mise à TRUE pour afficher les temps de calcul sur chaque ligne D.

34 Options diverses

34.1 Introduction à diverses options

Dans cette section diverses options sont présentées qui ont un effet global sur le fonctionnement de Maxima. Diverses listes sont aussi discutées, telle que la liste de toutes les fonctions définies par l'utilisateur.

34.2 SHARE

Le répertoire SHARE contient des programmes, des fichiers d'information, etc. qui sont considérés comme présentant de l'intérêt pour la communauté MACSYMA. La plupart des fichiers de SHARE; ne font pas partie par eux-mêmes du système MACSYMA et doivent être chargés individuellement par l'utilisateur, p. ex. LOADFILE("array");. Beaucoup de ces fichiers ont été proposés par des utilisateurs de MACSYMA. Faites PRINTFILE(SHARE,USAGE,SHARE); pour avoir plus de détails et les conventions pour contribuer au répertoire SHARE. Pour une "table des matières" annotée du répertoire, faites PRINTFILE(SHARE,>,SHARE);

34.3 Définitions pour diverses options

ALIASES Variable

par défaut: [] - des atomes ayant un alias défini par l'utilisateur (grâce aux fonctions ALIAS, ORDERGREAT, ORDERLESS ou en DECLAREant l'atome comme NOUN).

ALLSYM Variable

par défaut: [TRUE] - si TRUE tous les objets indicés sont supposés symétriques en tous leurs indices covariants et contravariants. Si FALSE alors aucune symétrie d'aucune sorte n'est supposée sur ces indices. Les indices des dérivées sont toujours pris comme étant symétriques.

ALPHABETIC declaration

Ajoutée à l'alphabet de MACSYMA qui contient à l'origine les lettres A-Z, % et _. Ainsi, DECLARE("~",ALPHABETIC) permet à NEW~VALUE d'être utilisée comme nom.

APROPOS (*chaîne*) Fonction

prend une chaîne de caractères comme argument et cherche dans tous les noms de MACSYMA ceux contenant cette chaîne. Ainsi, APROPOS(EXP); retournera une longue liste de tous les drapeaux et fonctions ayant EXP comme partie de leur nom, comme EXPAND, EXP, EXPONENTIALIZE. Si vous ne vous souvenez que d'une partie du nom de quelque chose vous pouvez utiliser cette commande pour trouver le reste du nom.

De même, vous pouvez lancer APROPOS(TR_); pour trouver une liste de beaucoup des commutateurs en relation avec le TRANSLATOR (la plupart d'entre eux commencent par TR_).

ARGS (*exp*) Fonction

retourne une liste des arguments de exp, i.e. elle est essentiellement équivalente à

```
SUBSTPART("[",exp,0)
```

ARGS et SUBSTPART dépendent toutes deux de la définition de INFLAG.

DUMMY (*i1,i2,...*) Fonction

fixe chaque indice i1,i2,... sur le nom de la forme !n où n est un entier positif. Ceci garantit que les indices factices qui sont indispensables pour former des expressions n'entreront pas en conflit avec les indices déjà utilisés. COUNTER[par défaut 1] détermine le suffixe numérique à utiliser pour créer l'indice factice suivant. Le préfixe est déterminé par l'option DUMMYX[!].

GENINDEX Variable

par défaut: [I] est le préfixe alphabétique utilisé pour créer la variable de sommation suivante si nécessaire.

GENSUMNUM Variable

[0] est le suffixe numérique utilisé pour créer la prochaine variable de sommation. Si mise à FALSE l'indice consistera seulement en GENINDEX sans suffixe numérique.

INF Variable

infinité réelle positive.

INFINITY Variable

infinité complexe, une grandeur infinie d'angle de phase arbitraire. Voir aussi INF et MINF.

INFOLISTS Variable

par défaut: [] - une liste des noms de toutes les listes d'information de MACSYMA. Ce sont :

LABELS - toutes les étiquettes liées C,D, et E.

VALUES - tous les atomes liés, i.e. les variables utilisateur, pas les options ou commutateurs de MACSYMA, (définis par : , :: , ou une liaison fonctionnelle).

FUNCTIONS - toutes les fonctions définies par l'utilisateur (par f(x):=...).

ARRAYS - tableaux déclarés ou non déclarés (définis par : , :: , ou :=...)

MACROS - toutes les macros définies par l'utilisateur.

MYOPTIONS - toutes les options qui ont été redéfinies par l'utilisateur (qu'elles soient ou non redéfinies à leur valeur par défaut).

RULES - règles de concordance et de simplification définies par l'utilisateur (définies par TELLSIMP, TELLSIMPAFTER, DEFMATCH, ou DEFRULE).

ALIASES - atomes qui ont un alias défini par l'utilisateur (avec les fonctions ALIAS, ORDERGREAT, ORDERLESS ou en déclarant (DECLARE) l'atome comme NOUN).

DEPENDENCIES - atomes qui ont des dépendances fonctionnelles (définies par les fonctions DEPENDS ou GRADEF).

GRADEFS - fonctions qui ont des dérivées définies par l'utilisateur (définies par la fonction GRADEF).

PROPS - atomes qui ont toute propriété autre que celles mentionnées ci-dessus, telles que atvalues, matchdeclares, etc. ainsi que les propriétés spécifiées par la fonction DECLARE.

LET_RULE_PACKAGES - une liste de tous les packages de règles définies par l'utilisateur plus le package spécial DEFAULT_LET_RULE_PACKAGE. (DEFAULT_LET_RULE_PACKAGE est le nom d'un package de règle utilisé lorsqu'il n'est pas explicitement défini par l'utilisateur.)

INTEGERP (*exp*) Fonction

est TRUE si exp est un entier, sinon FALSE.

M1PBRANCH Variable

par défaut: [FALSE] - "branche principale de -1 à une puissance". Les quantités telles que (-1)^(1/3) [i.e. exposant rationnel "impair"] et (-1)^(1/4) [i.e. exposant rationnel "pair"] sont maintenant traitées comme indiqué ci-dessous :

```
                DOMAIN:REAL(par défaut)

(-1)^(1/3):       -1
(-1)^(1/4):    (-1)^(1/4)

                  DOMAIN:COMPLEX
M1PBRANCH:FALSE(par défaut)   M1PBRANCH:TRUE
(-1)^(1/3)                 1/2+%i*sqrt(3)/2
(-1)^(1/4)               sqrt(2)/2+%i*sqrt(2)/2
```

NUMBERP (*exp*) Fonction

est TRUE si exp est un entier, un nombre rationnel, un nombre en virgule flottante ou un "bigfloat", sinon FALSE.

PROPERTIES (*a*) Fonction

contient une liste montrant les noms de toutes les propriétés associées à l'atome a.

PROPS special symbol

Des atomes ayant une propriété autre que celles explicitement mentionnées dans INFOLISTS, telles que les atvalues, matchdeclares, etc. ainsi que les propriétés spécifiées par la fonction DECLARE.

PROPVARS (*prop*) Fonction

contient une liste des atomes de la liste PROPS ayant la propriété indiquée par prop. Ainsi PROPVARS(ATVALUE) donnera une liste d'atomes ayant des atvalues.

PUT (*a, p, i*) Fonction

associe à l'atome a la propriété p avec l'indicateur i. Ceci permet à l'utilisateur de donner à un atome toute propriété arbitraire.

QPUT (*a, p, i*) Fonction

est sembleble à PUT mais ses arguments ne sont pas évalués.

REM (*a, i*) Fonction

supprime de l'atome a la propriété indiquée par i.

REMOVE (*args*) Fonction

supprime certaines ou toutes les propriétés associées aux variables ou fonctions.

REMOVE(a1, p1, a2, p2, ...) supprime la propriété pi de l'atome ai. Ai et pi peuvent aussi être des listes comme avec DECLARE. Pi peut être toute propriété e.g. FUNCTION, MODE_DECLARE, etc. Cet argument peut aussi être TRANSFUN, impliquant que la version LISP traduite de la fonction doit être supprimée. Utile si l'on veut exécuter la version MACSYMA de la fonction plutôt que la version traduite. Pi peut aussi être OP ou OPERATOR, pour supprimer une extension de syntaxe donnée à ai (voir l'Appendice II). Si ai est "ALL" la propriété indiquée par pi est supprimée de tous les atomes la possédant. Contrairement aux fonctions de suppression plus spécifiques (REMVALUE, REMARRAY, REMFUNCTION, et REMRULE), REMOVE ne signale pas qu'une propriété donnée n'existe pas; elle retourne toujours "DONE".

REMVALUE (*name1, name2, ...*) Fonction

supprime du système les valeurs des variables utilisateur (qui peuvent être indicées). Si name est ALL les valeurs de toutes les variables utilisateur sont supprimées. Les valeurs sont les items portant des noms donnés par l'utilisateur, contraiement à ceux qui sont automatiquement étiquettés par MACSYMA comme Ci, Di, ou Ei.

RENAME (*exp*) Fonction

retourne une expression équivalente à exp mais avec les indices factices de chaque terme choisis parmi l'ensemble [!1,!2,...]. Chaque indice factice d'un produit sera différent; pour une somme RENAME essaiera de faire (le même de chaque indice factice de la somme = each dummy index in a sum the same ???). De plus les indices seront triés alphanumériquement.

RNCOMBINE (*exp*) Fonction

transforme exp en combinant tous les termes de exp afin d'avoir des dénominateurs identiques ou qui diffèrent les uns des autres seulement de facteurs numériques. C'est légèrement différent du comportement de COMBINE, qui rassemble les termes ayant mêmes dénominateurs. Fixer PFEFORMAT:TRUE et utiliser COMBINE donnera des résultats semblables à ceux pouvant être obtenus avec RNCOMBINE, mais RNCOMBINE de plus "cross-multiplie" les facteurs numériques des dénominateurs. Le résultat donne des formes plus nettes, et la possiblité de reconnaître certaines suppressions. Les bogues à ASB.

SCALARP (*exp*) Fonction

est TRUE si exp est un nombre, une constante, ou une variable déclarée comme SCALAR, ou composée entièrement de nombres, de constantes, et de telles variables, mais pas de matrices ou de listes.

SCALEFACTORS (*coordinatetransform*) Fonction

Ici coordinatetransform s'évalue sous la forme [[expression1, expression2, ...], indeterminate1, indeterminat2, ...], où indeterminate1, indeterminate2, etc. sont les variables de coordonnées curvilinéaires et où un ensemble de composants cartésiens rectangulaires est donné en termes de coordonnées curvilinéaires par [expression1, expression2, ...]. COORDINATES est défini par le vecteur [indeterminate1, indeterminate2,...], et DIMENSION par la longueur de ce vecteur. SF[1], SF[2], ..., SF[DIMENSION] sont définis par les facteurs d'échelle des coordonnées, et SFPROD est défini par le produit de ces facteurs d'échelle. Initialement, COORDINATES est [X, Y, Z], DIMENSION est 3, et SF[1]=SF[2]=SF[3]=SFPROD=1, correspondant aux coordonnées cartésiennes rectangulaires à 3 dimensions. Pour développer une expression en composants physiques du système de coordonnées courant, il y a une fonction de la forme

SETUP_AUTOLOAD (*fichier,func1,...,funcN*) Fonction

qui prend deux arguments ou plus : un fichier de spécification, et un ou plusieurs noms de fonction, "funcI", et qui indique que si un appel à "funcI" est fait et que "funcI" n'est pas définie, le fichier spécifié par "fichier" doit être automatiquement chargé par LOAD, que ce fichier devra contenir une définition pour "funcI" (c'est par ce processus qu'en appelant par exemple INTEGRATE dans un nouveau MACSYMA divers fichiers vont être chargés). Comme pour d'autres commandes de traitement de fichiers dans MACSYMA, les arguments de SETUP_AUTOLOAD ne sont pas évalués. Exemple:

SETUP_AUTOLOAD("bessel")$ J1(0.0);.

Note: SETUP_AUTOLOAD ne fonctionne pas avec les fonctions de tableau.

35 Règles et modèles

35.1 Introduction aux règles et modèles

Cette section présente la recherche de concordances définies par l'utilisateur et les règles de simplification (fixées par TELLSIMP, TELLSIMPAFTER, DEFMATCH, ou DEFRULE). Vous pouvez préférer les procédures principales de simplification, ou bien avoir vos propres règles appliquées explicitement avec APPLY1 et APPLY2.

Il existe des mécanismes supplémentaires pour les règles polynomiales sous TELLRAT, et pour l'algèbre commutative et non commutative dans le chapitre AFFINE.

35.2 Définitions pour les règles et les modèles

APPLY1 (*exp, règle1, ..., règlen*) Fonction

applique sans arrêt la première règle à exp jusqu'à ce qu'elle échoue, puis de la même façon la même règle à toutes les sous-expressions de exp, de gauche à droite, jusqu'à ce que la première règle ait échoué sur toutes les sous-expressions. Soit exp' le résultat de la transformation de exp de cette manière. Alors la seconde règle est appliquée de la même façon en commençant au début de exp'. Lorsque la dernière règle échoue sur la dernière sous-expression, l'application est terminée.

APPLY2 (*exp, règle1, ..., règlen*) Fonction

diffère de APPLY1 en ce que si la première règle échoue sur une sous-expression donnée, alors la seconde règle est appliquée à son tour, etc. L'ensemble complet de règles est appliqué à la sous-expression suivante seulement si elles échouent toutes sur une sous-expression donnée. Si l'une des règles réussit, la même sous-expression est re-traitée, en commençant par la première règle.

MAXAPPLYDEPTH[10000] est la profondeur maximale que pourront atteindre APPLY1 et APPLY2.

APPLYB1 (*exp, règle1, ..., règlen*) Fonction

est semblable à APPLY1 mais commence à partir du bas vers le haut au lieu de l'inverse. C'est à dire qu'elle traite la plus petite sous-expression de exp, puis la plus petite suivante, etc.

MAXAPPLYHEIGHT[10000] est la hauteur maximale qu'atteindra APPLYB1 avant d'abandonner.

CURRENT_LET_RULE_PACKAGE Variable

par défaut:[DEFAULT_LET_RULE_PACKAGE] - le nom du package de règle qui est actuellement utilisé. L'utilisateur peut réinitialiser cette variable au nom d'un autre package de règle précédemment défini par la commande LET. Lorsque l'une des fonctions contenue dans le package let est appelée sans un nom de package la valeur de

CURRENT_LET_RULE_PACKAGE

est utilisée. Si un appel tel que LETSIMP(expr,règle_pkg_name); est lancé, le package de règle rule_pkg_name est utilisé pour cette seule commande LETSIMP, i.e. la valeur de CURRENT_LET_RULE_PACKAGE n'est pas changée.

DEFAULT_LET_RULE_PACKAGE Variable

le nom du package de règle utilisé lorsqu'aucun n'est explicitement donné par l'utilisateur, avec LET ou en changeant la valeur de CURRENT_LET_RULE_PACKAGE.

DEFMATCH (*nomprog, modèle, parm1, ..., parmn*) Fonction

crée une fonction de n+1 arguments de nom nomprog qui vérifie si une expression peut correspondre à un modèle particulier. Le modèle est une certaine expression contenant les variables et paramètres du modèle. Les "parms" sont donnés explicitement en arguments à DEFMATCH alors que les variables du modèle (si fournies) sont données implicitement par une fonction MATCHDECLARE précédente. Le premier argument de la fonction nomprog créée, est une expression à comparer au "modèle" et les n autres arguments sont les variables réelles se trouvant dans l'expression qui prendront la place des variables formelles se trouvant dans le "modèle". Ainsi les "parms" de DEFMATCH sont comme les arguments (dummy ??? fictifs, formels) de l'instruction SUBROUTINE du FORTRAN. Lorsque la fonction est "appelée" les arguments réels sont substitués. Par exemple:

```
(C1)  NONZEROANDFREEOF(X,E):=  IF E#0 AND FREEOF(X,E)
            THEN TRUE ELSE FALSE$
(IS(E#0 AND FREEOF(X,E)) est une définition de fonction
équivalente)
(C2)  MATCHDECLARE(A,NONZEROANDFREEOF(X),B,FREEOF(X))$
(C3)  DEFMATCH(LINEAR,A*X+B,X)$
```

qui définit la fonction LINEAR(exp,var1). Elle teste exp pour voir si elle est de la forme A*var1+B où A et B ne contiennent pas var1 et A est non zéro. Les fonctions définies par DEFMATCH retournent (si la comparaison réussie) une liste d'équations dont les membres gauches sont les variables et paramètres du modèle et dont les membres droits sont les expressions correspondant aux variables et paramètres du modèle. Les variables du modèle, mais pas ses paramètres, (prennent les valeurs = are set to ???) des expressions correspondantes. Si la comparaison échoue, la fonction retourne FALSE. Ainsi LINEAR(3*Z+(Y+1)*Z+Y**2,Z) retournera [B=Y**2, A=Y+4, X=Z]. Toutes les variables non déclarées comme variables de modèle dans MATCHDECLARE ou comme paramètre dans DEFMATCH se trouvant dans modèle correspondront seulement à elles-mêmes, de sorte que si le troisième argument de DEFMATCH en (C4) avait été omis, LINEAR ne ferait correspondre que les expressions linéaires en X, pas en toute autre variable.

Un modèle qui ne contient pas de paramètres ni de variables de modèle retourne TRUE si la comparaison réussie. Faites EXAMPLE(DEFMATCH); pour d'autres exemples.

DEFRULE (*nomrègle, modèle, remplacement*) Fonction

définit et nomme une règle de remplacement pour le modèle donné. Si la règle nommée nomrègle est appliquée à une expression (par l'une des fonctions APPLY ci-dessous), toute sous-expression correspondant au modèle sera remplacée par le "remplacement". Toutes les variables du remplacement qui ont reçues des valeurs par correspondance avec le modèle reçoivent ces valeurs dans le remplacement qui est alors simplifié. Les règles elles-mêmes peuvent être traitées comme des fonctions qui transforment une expression en une opération de comparaison d'un modèle et remplacement. Si le modèle échoue, l'expression originale est retournée.

DISPRULE (*nomrègle1, nomrègle2, ...*) Fonction

affiche les règles de noms nomrègle1, nomrègle2, tels que donnés par DEFRULE, TELLSIMP, ou TELLSIMPAFTER ou par un modèle défini par DEFMATCH. Par exemple, la première règle modifiant SIN sera appelée SINRULE1. DISPRULE(ALL); affichera toutes les règles.

LET (*prod, repl, predname, arg1, arg2, ..., argn*) Fonction

définit une règle de substitution pour LETSIMP telle que prod est remplacé par repl. prod est un produit de puissances positives ou négatives des types suivants :

- (1) Des atomes que LETSIMP recherchera littéralement, sauf si avant d'appeler LETSIMP la fonction MATCHDECLARE a été utilisée pour associer un prédicat à l'atome. Dans ce cas LETSIMP fera correspondre à l'atome tout terme d'un produit satisfaisant le prédicat.
- (2) Des noyaux tels que SIN(X), N!, F(X,Y), etc. Comme avec les atomes LETSIMP recherchera une correspondance littérale à moins que MATCHDECLARE soit utilisée pour associer un prédicat à l'argument du noyau. Un terme d'une puissance positive ne correspondra qu'à un terme ayant au moins cette puissance dans l'expression soumise à LETSIMP. Un terme d'une puissance négative ne correspondra d'autre part qu'à un terme ayant une puissance au moins négative. Dans le cas des puissances négatives du "produit" le commutateur LETRAT devra être mis à TRUE (voir ci-dessous).

  Si un prédicat est inclus dans la fonction LET suivi d'une liste d'arguments, une tentative de correspondance (i.e. une qui serait acceptée si le prédicat était omis) sera acceptée seulement si predname(arg1',...,argn') s'évalue à TRUE, où les argi' sont les valeurs correspondant aux argi. argi peut être le nom de tout atome ou l'argument de tout noyau apparaissant dans prod. repl peut être toute expression rationnelle.

  Si certains de ces atomes ou arguments de prod apparaissent dans repl les substitutions appropriées sont effectuées.

Si LETRAT[FALSE] est FALSE, LETSIMP simplifiera le numérateur et le dénominateur de expr indépendamment et retournera le résultat. Des substitutions telle que N!/N donnant (N-1)! échoueront. Pour traiter de telles situations LETRAT devra être TRUE, alors le numérateur, le dénominateur, et leur quotient seront simplifiés dans cet ordre.

Ces fonctions de substitution vous permettent de travailler simultanément avec plusieurs packages de règles. Chacun d'eux peut contenir un nombre quelconque de

règles LET et est référé par un nom donné par l'utilisateur. Pour insérer une règle dans un nom de package de règle, faites LET([prod,repl,pred,arg1,...],nom). Pour appliquer les règles d'un nom de package, faites LETSIMP(expr, nom). La fonction LETSIMP(expr,name1,name2,...) est équivalente à LETSIMP(expr,name1) suivie de LETSIMP(%,name2) etc.

CURRENT_LET_RULE_PACKAGE est le nom du package de règle actuellement utilisé. L'utilisateur peut redéfinir cette variable sur le nom de tout package de règle précédemment défini avec la commande LET. Lorsque l'une des fonctions contenues dans le package let est appelée sans nom de package la valeur de CURRENT_LET_RULE_PACKAGE est utilisée. Si un appel tel que LETSIMP(expr,rule_pkg_name); est lancé, le package de règle rule_pkg_name est utilisé pour cette seule commande LETSIMP, i.e. la valeur de CURRENT_LET_RULE_PACKAGE n'est pas changée.

Il existe un DEFAULT_LET_RULE_PACKAGE qui est pris lorsqu'aucun autre nom n'est donné à aucune des fonctions. Dès qu'un LET inclut un nom de package de règle il est utilisé en tant que CURRENT_LET_RULE_PACKAGE.

LETRAT Variable

par défaut: [FALSE] - si FALSE, LETSIMP simplifiera le numérateur et le dénominateur de expr indépendamment et retournera le résultat. Les substitutions telle que N!/N donnant (N-1)! échoueront. Pour traiter de telles situations LETRAT devra être TRUE, alors le numérateur, le dénominateur, et leur quotient seront simplifiés dans cet ordre.

LETRULES () Fonction

affiche les règles du package de règle courant. LETRULES(nom) affiche les règles du package nommé. Le package de règle courant est la valeur de

```
CURRENT_LET_RULE_PACKAGE
```

La valeur initiale des règles est

```
DEFAULT_LET_RULE_PACKAGE
```

LETSIMP (*exp*) Fonction

applique en permanence les règles de substitution précédemment définies par la fonction LET jusqu'à ce qu'aucune autre modification ne soit faite à exp.

LETSIMP(expr,rule_pkg_name); fera utiliser le package de règle rule_pkg_name pour cette seule commande LETSIMP, i.e. la valeur de CURRENT_LET_RULE_PACKAGE n'est pas changée.

LET_RULE_PACKAGES Variable

par défaut:[DEFAULT_LET_RULE_PACKAGE] - La valeur de LET_RULE_PACKAGES est une liste de tous les package de règles let définis par l'utilisateur, plus le package spécial

```
DEFAULT_LET_RULE_PACKAGE
```

C'est le nom du package de règle utilisé lorsqu'aucun n'est explicitement donné par l'utilisateur.

MATCHDECLARE (*patternvar, predicate, ...*) Fonction

associe un prédicat à une variable de modèle de sorte que la variable ne fera correspondre que les expressions pour lesquelles le prédicat n'est pas FALSE (la concordance est établie par l'une des fonctions décrites ci-dessous). Par exemple, après l'exécution de

```
MATCHDECLARE(Q,FREEOF(X,%E))
```

Q correspondra à toute expression ne contenant pas X ou %E. Si la concordance réussie alors la variable est définie sur l'expression correspondante. Le prédicat (dans ce cas FREEOF) est écrit sans le dernier argument qui devrait être celui sur lequel la variable du modèle doit être testée. Notez que "patternvar" et les arguments du prédicat sont évalués en même temps que la correspondance. L'argument de rang impair peut aussi être une liste de variables de modèle, ayant toutes le même prédicat associé. Tout nombre pair d'arguments peut être donné.

Pour le (patttern matching filtrage???), les prédicats se réfèrent à des fonctions qui sont soit FALSE soit non FALSE (toute valeur non FALSE vaut TRUE). MATCHDECLARE(var,TRUE) permet à var de correspondre à n'importe quelle expression.

MATCHFIX Fonction

Les opérateurs MATCHFIX sont utilisés pour dénoter des fonctions d'un nombre quelconque d'arguments qui sont passés dans une liste à la fonction. Les arguments se trouvent entre l'opérateur principal et son délimiteur "correspondant". La fonction MATCHFIX("x",...) est une extension de syntaxe qui déclare x comme étant un opérateur MATCHFIX. La puissance liée par défaut est 180, et les ARGS à l'intérieur peuvent être quelconque.

```
(C1) matchfix("|","|");

(D1)                                    "|"
(C2) |a|+b;

(D2)                                  b + (|a|)
(C3) |(a,b)|;

(D3)                                     |b|
(C4) |[a,b]|;

(D4)                                  |[a, b]|

(C9) |x|:=IF NUMBERP(x) THEN ABS(x)
        ELSE (IF LISTP(x) AND APPLY("and",MAP(NUMBERP,x))
                  THEN SUM(x[i]^2,i,1,LENGTH(x))^0.5 ELSE BUILDQ([u:x],|u|))$

(C10) |[1,2,3]|;

(D10)                           3.741657386773941

(C18) |-7|;
```

```
(D18)                                  7
(C19) |[a,b]|;

(D19)                                |[a, b]|
```

REMLET (*prod, nom*) Fonction

supprime la règle de substitution, prod –> repl, la plus récemment définie par la fonction LET. Si nom est fourni la règle est supprimée du package de règle nom. REMLET() et REMLET(ALL) suppriment toutes les règles de substitution du package de règle courant. Si le nom d'un package de règle est donné, p. ex. REMLET(ALL,nom), le package, nom, est aussi supprimé. Si une substitution doit être changée, utilisant le même produit, REMLET n'a pas à être appelée, il suffit de redéfinir la substitution utilisant (littéralement) le même produit avec la fonction LET et le nouveau nom de remplacement et/ou du prédicat. Lorsque REMLET(produit) sera maintenant appelée la règle de substitution originale sera (rétablie revived???).

REMRULE (*fonction, nomrègle*) Fonction

enlève une règle de nom nomrègle de la fonction où elle avait été placée par DEFRULE, DEFMATCH, TELLSIMP, ou TELLSIMPAFTER. Si nomrègle est ALL, alors toutes les règles sont supprimées.

TELLSIMP (*modèle, remplacement*) Fonction

est semblable à TELLSIMPAFTER mais place la nouvelle information avant l'ancienne ce qui fait qu'elle est appliquée avant les règles de simplification intégrées. TELLSIMP est utilisée lorsqu'il est important de modifier l'expression avant que le simplificateur travaille sur elle, par exemple si le simplificateur "sait" quelque chose concernant l'expression, mais que ce qu'il retourne ne vous plait pas. Si le simplificateur "sait" quelque chose concernant l'opérateur principal de l'expression, mais n'en fait pas assez pour vous, vous voudrez probablement utiliser TELLSIMPAFTER. Le modèle peut ne pas être une somme, un produit, une variable unique, ou un nombre. RULES est une liste de noms auxquels des règles de simplification ont été ajoutées par DEFRULE, DEFMATCH, TELLSIMP, ou TELLSIMPAFTER. Faites EXAMPLE(TELLSIMP); pour les exemples.

TELLSIMPAFTER (*modèle, remplacement*) Fonction

définit un remplacement pour le modèle que le simplificateur de MACSYMA utilise après l'application des règles de simplification intégrées. Le modèle peut être quelconque mais pas une unique variable ou un nombre.

36 Listes

36.1 Introduction aux listes

Les listes sont les constructions de base pour maxima et lisp. Tous les types de donnée autres que les tableaux, les tables de hachage, les nombres, sont représentés par des listes lisp. Celles-ci ont la forme

```
((mplus) $A 2)
```

qui indique une expression `A+2`. Au niveau maxima on pourra voir la notation infixée `A+2`. Maxima a aussi des listes qui sont affichées comme

```
[1, 2, 7, x+y]
```

pour une liste de 4 éléments. En interne ceci correspond à une liste lisp de la forme

```
((mlist) 1  2  7  ((mplus)  $X $Y ))
```

Le drapeau qui dénote le type du champ de l'expression maxima est une liste elle-même, car après qu'elle soit passée par le simplificateur la liste deviendra

```
((mlist simp) 1 2 7 ((mplus simp) $X $Y))
```

36.2 Définitions pour les listes

APPEND (*list1, list2, ...*) Fonction

retourne une unique liste des éléments de list1 suivis de ceux de list2,... APPEND travaille aussi sur les expressions générales, p. ex. APPEND(F(A,B), F(C,D,E)); -> F(A,B,C,D,E). Faites EXAMPLE(APPEND); pour un exemple.

ATOM (*exp*) Fonction

est TRUE si exp est atomique (i.e. un nombre ou un nom), sinon FALSE. Ainsi ATOM(5) est TRUE alors que ATOM(A[1]) et ATOM(SIN(X)) sont FALSE (en supposant que A[1] et X ne sont pas liés).

CONS (*exp, list*) Fonction

retourne une nouvelle liste construite avec l'élément exp comme premier élément, suivi des éléments de "list". CONS travaille aussi sur d'autres expressions, p. ex. CONS(X, F(A,B,C)); -> F(X,A,B,C).

COPYLIST (*L*) Fonction

crée une copie de la liste L.

DELETE (*exp1, exp2*) Fonction

supprime toutes les occurrences de exp1 dans exp2. Exp1 peut être un terme de exp2 (si c'est une somme) ou un facteur de exp2 (si c'est un produit).

```
(C1)  DELETE(SIN(X),X+SIN(X)+Y);
(D1)                Y + X
```

DELETE(exp1, exp2, nombre) enlève les premières "nombre" occurrences de exp1 dans exp2. Bien sûr, s'il y en a moins que "nombre" alors toutes les occurrences seront supprimées.

ENDCONS (*exp, list*) Fonction

retourne une nouvelle liste constituée des éléments de "list" suivis par exp. ENDCONS fonctionne aussi sur les expressions générales, p. ex. ENDCONS(X, F(A,B,C)); -> F(A,B,C,X).

FIRST (*exp*) Fonction

renvoie la première partie de exp qui peut être le premier élément d'une liste, la première rangée d'une matrice, le premier terme d'une somme, etc. Notez que FIRST et ses fonctions apparentées, REST et LAST, travaillent sur la forme affichée de exp, pas sur la forme tapée en entrée. Cependant, si la variable INFLAG [FALSE] est mise à TRUE, ces fonctions examineront la forme interne de exp. Notez que le simplificateur ré-organise les expressions. Ainsi FIRST(X+Y) deviendra X si INFLAG est TRUE et Y si INFLAG est FALSE. (FIRST(Y+X) donne le même résultat).

GET (*a, i*) Fonction

récupère la propriété utilisateur indiquée par i associée à l'atome a ou retourne FALSE s'il n'a pas la propriété i.

```
(C1) PUT(%E,'TRANSCENDENTAL,'TYPE);
(D1)                              TRANSCENDENTAL
(C2) PUT(%PI,'TRANSCENDENTAL,'TYPE)$
(C3) PUT(%I,'ALGEBRAIC,'TYPE)$
(C4) TYPEOF(EXP) := BLOCK([Q],
                          IF NUMBERP(EXP)
                          THEN RETURN('ALGEBRAIC),
                          IF NOT ATOM(EXP)
                          THEN RETURN(MAPLIST('TYPEOF, EXP)),
                          Q : GET(EXP, 'TYPE),
                          IF Q=FALSE
                          THEN ERRCATCH(ERROR(EXP,"is not numeric.")) ELSE Q)$
(C5) TYPEOF(2*%E+X*%PI);
X is not numeric.
(D5)          [[TRANSCENDENTAL, []], [ALGEBRAIC, TRANSCENDENTAL]]
(C6) TYPEOF(2*%E+%PI);
(D6)             [TRANSCENDENTAL, [ALGEBRAIC, TRANSCENDENTAL]]
```

LAST (*exp*) Fonction

renvoie la dernière partie (terme, rangée, élément, etc.) de l'exp.

LENGTH (*exp*) Fonction

donne (par défaut) le nombre de parties dans la forme externe (affichée) de exp. Pour les listes c'est le nombre d'éléments, pour les matrices le nombre de rangées, et pour les sommes le nombre de termes (voir DISPFORM). La commande LENGTH est affectée par le commutateur INFLAG [par défaut FALSE]. Ainsi, p. ex. LENGTH(A/(B*C)); donne 2 si INFLAG est FALSE (en supposant que EXPTDISPFLAG est TRUE), mais 3 si INFLAG est TRUE (la représentation interne est essentiellement A*B^-1*C^-1).

LISTARITH Variable

par défaut: [TRUE] - si FALSE provoque la suppression de toutes les opérations arithmétiques sur les listes; si TRUE, les opérations liste-matrice sont contagieuses, provoquant la conversion des listes en matrices, donnant un résultat qui est toujours une matrice. Cependant, les opérations liste-liste devront retourner des listes.

LISTP (*exp*) Fonction

est TRUE si exp est une liste, sinon FALSE.

MAKELIST (*exp,var,lo,hi*) Fonction

retourne une liste comme valeur. MAKELIST peut être appelée avec MAKELIST(exp,var,lo,hi) ["lo" et "hi" doivent être des entiers], ou comme MAKELIST(exp,var,list). Dans le premier cas MAKELIST est analogue à SUM, alors que dans le second cas MAKELIST est semblable à MAP. Exemples:

```
MAKELIST(CONCAT(X,I),I,1,6) renvoie [X1,X2,X3,X4,X5,X6]
MAKELIST(X=Y,Y,[A,B,C]) renvoie [X=A,X=B,X=C]
```

MEMBER (*exp, list*) Fonction

retourne TRUE si exp est membre de "list" (pas "dans" un membre). Autrement retourne FALSE. MEMBER fonctionne aussi sur les expressions non-listes, p. ex. MEMBER(B, F(A,B,C)); -> TRUE.

REST (*exp, n*) Fonction

renvoie exp dont les n premiers éléments ont été supprimés si n est positif et ses -n derniers éléments si n est négatif. Si n est 1 il peut être omis. Exp peut être une liste, une matrice, ou une autre expression.

REVERSE (*list*) Fonction

inverse l'ordre des membres de la liste (pas les membres eux-mêmes). REVERSE travaille aussi sur les expressions générales, p. ex. REVERSE(A=B); donne B=A.

REVERSE par défaut: [FALSE] - dans les fonctions de tracé, si TRUE un système de coordonnées à gauche est supposé.

37 Définition de fonction

37.1 Introduction à la définition de fonction

37.2 Fonction

Pour définir une fonction dans MACSYMA vous utilisez l'opérateur :=. P. ex.

```
F(X):=SIN(X)
```

définit une fonction F. Des fonctions anonymes peuvent aussi être créées avec LAMBDA. Par exemple

```
lambda([i,j], ... )
```

peut être utilisée à la place de F où

```
F(I,J):=BLOCK([], ... );
MAP(LAMBDA([I],I+1),L)
```

retournera une liste avec 1 ajouté à chaque terme.

Vous pouvez aussi définir une fonction à nombre variable d'arguments, en donnant un argument final qui est affecté à la liste des arguments supplémentaires :

```
(C8) f([u]):=u;
(C9) f(1,2,3,4);
(D9)   [1, 2, 3, 4]
(C11) f(a,b,[u]):=[a,b,u];
(C12) f(1,2,3,4,5,6);
(D12)       [1, 2, [3, 4, 5, 6]]
```

Le membre droit d'une fonction est une expression. Ainsi, si vous voulez une suite d'expressions, vous faites

```
f(x):=(expr1,expr2,....,exprn);
```

et la valeur de exprn est celle retournée par la fonction.

Si vous désirez faire un `return` pour une certaine expression dans la fonction vous devez utiliser `block` et `return`.

```
block([],expr1,...,if(a>10) then return(a),...exprn)
```

est en elle-même une expression, et elle peut ainsi prendre la place du membre droit d'une définition de fonction. Ici il peut se faire que le retour se produise avant la dernière expression.

Le premier `[]` du bloc peut contenir une liste de variables et des affectations de variable, tel que `[a:3,b,c:[]]`, qui fait que les trois variables `a`,`b`, et `c` ne sont plus référencées par leurs valeurs globales, mais ont plutôt ces valeurs spéciales tant que le code exécuté dans le `block`, ou dans des fonctions appelées dans ce `block`. Ceci est appelé liaison *dynamique*, car les variables vivent du début du bloc jusqu'à sa sortie. Une fois sorti, ou ejecté, du `block`, les anciennes valeurs (si elles existent) des variables sont restaurées. C'est certainement une bonne idée de protéger vos variables de cette façon. Notez que les affectations des variables du bloc sont faites en parallèle. Ce qui signifie que si vous aviez utilisé `c:a` ci-dessus, la valeur de `c` aurait été celle de `a` au moment où vous êtes entré dans le bloc, mais avant que `a` soit lié. Ainsi faire quelque chose comme

```
block([a:a],expr1,... a:a+3,...exprn)
```

empêchera la valeur externe de `a` d'être altérée, mais vous laissera y accéder quelle que soit sa valeur. Ainsi le membre droit des affectations est évalué dans le contexte entrant, avant que les liaisons se produisent. Utiliser juste `block([x],..` fera que x aura lui-même pour valeur, exactement comme celle qu'il aurait si vous entriez dans une nouvelle session **MAXIMA**.

Les arguments réels de la fonction sont traités exactement de la même façon que les variables dans un bloc. Ainsi, dans

```
f(x):=(expr1,...exprn);
et
f(1);
```

nous aurions pour l'évaluation des expressions un contexte semblable à celui que nous aurions avec

```
block([x:1],expr1,...exprn)
```

à l'intérieur des fonctions, lorsque le membre droit d'une définition peut être calculé à l'exécution, il est utile d'utiliser `define` et si possibly `buildq`.

37.3 Macros

BUILDQ([varlist],expression); Fonction

EXPRESSION est toute expression MAXIMA unique et VARLIST est une liste des éléments de la forme `<atome>` ou `<atome>:<valeur>`

37.3.1 Sémantiques

Les <valeur>s dans la <varlist> sont évaluées de gauche à droite (la syntaxe <atome> est équivalente à <atome>:<atome>), puis ces valeurs sont substituées dans <expression> en parallèle. Si un quelconque <atome> apparaît comme unique argument d'une forme spéciale SPLICE (i.e. SPLICE(<atome>)) dans <expression>, alors la valeur associée à cet <atome> devra être une liste macsyma, et elle est (spliced = découpée ???) en <expression> plutôt que d'être substituée.

37.3.2 Simplification

Les arguments de BUILDQ doivent être protégés de toute simplification jusqu'à ce que les substitutions aient été effectuées. Ce code devra y pourvoir en utilisant `'`.

`buildq` peut être utile pour construire des fonctions à la volée. L'une des choses les plus puissantes concernant **MAXIMA** est que vos fonctions peuvent définir d'autres fonctions pour aider à résoudre le problème. Plus loin dans ce chapitre nous discuterons de la construction d'une fonction récursive, pour la solution d'une suite. Cette façon de définir des fonctions à l'intérieur de fonctions utilise en général `define`, qui évalue ses arguments. Nombre d'exemples sont inclus sous `splice`.

SPLICE (*atome*) Fonction

Est utilisé avec buildq pour construire une liste. Pratique pour faire des listes d'arguments, conjointement avec BUILDQ :

```
MPRINT([X]) ::= BUILDQ([U : x],
  if (debuglevel > 3) print(splice(u)));
```

Inclure un appel tel que

```
MPRINT("la matrice est ",MAT,"de longueur ",LENGTH(MAT))
```

équivaut à placer la ligne

```
IF DEBUGLEVEL > 3
  THEN PRINT("la matrice est ",MAT,"de longueur ",
              LENGTH(MAT))
```

Un exemple moins trivial serait l'affichage des valeurs des variables ET de leur noms.

```
MSHOW(A,B,C)
```

deviendra

```
PRINT('A,"=",A,",",'B,"=",B,", and",'C,"=",C)
```

de sorte que si cette ligne se trouve dans un programme nous pouvons afficher les valeurs.

```
(C101) foo(x,y,z):=mshow(x,y,z);
(C102) foo(1,2,3);
X = 1 , Y = 2 , et Z = 3
```

La définition réelle de mshow est la suivante. Notez comment buildq vous permet de construire une structure 'QUOTED', de sorte que le `'u` vous permet d'obtenir le nom de la variable. Notez aussi que dans les macros, le RESULT est un morceau de code qui sera remplacé par la macro et évalué.

```
MSHOW([lis])::=BLOCK([ans:[],N:LENGTH(lis)],
   FOR i THRU N DO
       (ans:APPEND(ans,
   BUILDQ([u:lis[i]],
  ['u,"=",u])),
IF i < N
    THEN ans
    :APPEND(ans,
    IF i < N-1 THEN [","]
        ELSE [", and"])),
   BUILDQ([U:ans],PRINT(SPLICE(u))))
```

Le splice travaille aussi pour mettre les arguments dans les opérations algébriques :

```
(C108) BUILDQ([A:'[B,C,D]],+SPLICE(A));
(D108) D+C+B
```

Notez comment la simplification ne se produit qu'APRèS la substitution. L'opération appliquée au splice est dans le premier cas le `+`, alors que dans le second c'est le `*`, pourtant vous pourriez penser que logiquement `splice(a)+splice(A)` pourrait être remplacé par `2*splice(A)`. Aucune simplification n'a lieu avec buildq. Pour comprendre ce que fait SPLICE avec l'algèbre vous devez comprendre que pour MAXIMA, une formule et une opération comme `A+B+C` est en fait semblable en interne à `+(A,B,C)`, et de même pour la multiplication. Ainsi `*(2,B,C,D)` est `2*B*C*D`

```
(C114) BUILDQ([A:'[B,C,D]],+SPLICE(A));
(D114) D+C+B
```

```
(C111) BUILDQ([A:'[B,C,D]],SPLICE(A)+SPLICE(A));
(D111) 2*D+2*C+2*B
  but
(C112) BUILDQ([A:'[B,C,D]],2*SPLICE(A));
(D112) 2*B*C*D
```

Enfin le buildq est sans prix pour construire des fonctions récursives. Supposons que votre programme résoud une équation différentielle en utilisant la méthode des séries, et qu'il a déterminé qu'il a besoin de construire une relation de récurrence

```
F[N]:=(-((N^2-2*N+1)*F[N-1]+F[N-2]+F[N-3])/(N^2-N))
```

et qu'il doit le faire à la volée dans votre fonction. Vous aimeriez vraiment maintenant ajouter `expand`,

```
F[N]:=EXPAND((-((N^2-2*N+1)*F[N-1]+F[N-2]+F[N-3])
  /(N^2-N)));
```

mais comment créer ce code ? Vous voulez que `expand` se produise chaque fois que la fonction s'exécute, PAS avant.

```
kill(f),
val:(-((N^2-2*N+1)*F[N-1]+F[N-2]+F[N-3])/(N^2-N)),
define(f[n],buildq([u:val],expand(u))),
```

fait le travail. Ce peut être utile, puisque lorsque vous faites

```
avec Expand
(C28) f[6];
(D28) -AA1/8-13*AA0/180
```

où sans lui il reste non simplifié, et même après 6 termes il devient :

```
(C25) f[6];
(D25) (5*(-4*(-3*(-2*(AA1+AA0)+AA1+AA0)/2
    -(AA1+AA0)/2+AA1)
/3
-(-2*(AA1+AA0)+AA1+AA0)/6+(-AA1-AA0)/2)
      /4
      +(-3*(-2*(AA1+AA0)+AA1+AA0)/2
       -(AA1+AA0)/2+AA1)
       /12-(2*(AA1+AA0)-AA1-AA0)/6)
      /30
```

L'expression devient vite compliquée si elle n'est pas simplifiée à chaque étape, donc la simplification doit faire partie de la définition. Par conséquent le `buildq` est utile pour construire la forme.

37.4 Optimisation

Lors de l'utilisation de TRANSLATE et en générant du code avec MACSYMA, il y a nombre de techniques qui peuvent gagner du temps et être utiles. Faites DEMO("optimu.dem") pour une démonstration. En particulier, la fonction FLOAT-DEFUNK du TRANSL;OPTIMU FASL, crée une définition de fonction à partir d'une expression mathématique, mais elle l'optimise (avec OPTIMIZE) et la place en MODE_DECLARE nécessaire pour la compiler correctement (ceci peut être fait à la main, évidemment). La démonstration ne s'exécutera que dans un nouveau macsyma.

37.5 Définitions pour la définition de fonction

APPLY (*fonction, liste*) Fonction

donne le résultat de l'application de la fonction à la liste de ses arguments. Utile lorsque l'on veut calculer les arguments d'une fonction avant d'appliquer cette fonction. Par exemple, si L est la liste [1, 5, -10.2, 4, 3], alors APPLY(MIN,L) donne -10.2. APPLY est aussi utile pour appeler des fonctions dont les arguments ne sont pas évalués si l'on veut connaître leur évaluation. Par exemple, si FILESPEC est une variable liée à la liste [TEST,CASE] alors APPLY(CLOSEFILE,FILESPEC) est équivalente à CLOSEFILE(TEST,CASE). En général le premier argument de APPLY devra être précédé d'une ' pour l'obliger à s'évaluer lui-même. Comme certaines variables atomiques ont le même nom que certaines fonctions, les valeurs de la variable seront utilisées plutôt que la fonction car le premier argument de APPLY est évalué, ainsi que son second.

BINDTEST (*ai*) Fonction

ai signale une erreur s'il est utilisé dans un calcul non lié.

BLOCK ([*v1,...,vk*], *statement1,...,statementj*) Fonction

Les blocs dans MACSYMA sont quelque peu analogues aux sous-routines du FORTRAN ou aux procédures de l'ALGOL ou du PL/I. Les blocs sont comme des instructions composées mais permettent aussi à l'utilisateur d'étiqueter les instructions dans le bloc et d'affecter des variables "factices" (ou formelles???) aux valeurs qui sont locales au bloc. Les vi sont des variables qui sont locales au BLOCK et les stmti sont des expressions MACSYMA quelconques. Si aucune des variables n'est locale alors la liste peut être omise. Un bloc utilise ces variables locales pour éviter des conflits avec les variables ayant les mêmes noms utilisées hors du bloc (i.e. globales au bloc). Dans ce cas, à l'entrée dans le bloc, les valeurs globales sont sauvées sur une pile et sont inaccessibles pendant l'exécution du bloc. Les variables locales sont alors dé-liées de sorte qu'elles s'évaluent à elles-mêmes. Elles peuvent être liées à des valeurs arbitraires dans le bloc mais à la sortie du bloc les valeurs sauvées sont restituées à ces variables. Les valeurs créées dans le bloc pour ces variables locales sont perdues. Si une variable utilisée dans une bloc 'est pas sur la liste des variables locales de ce bloc elle sera identifiée à la variable utilisée hors du bloc.

Si on veut sauver et restaurer d'autres propriétés locales que VALEUR, par exemple ARRAY (sauf les tableaux complets), FUNCTION, DEPENDENCIES, ATVALUE, MATCHDECLARE, ATOMGRAD, CONSTANT, et NONSCALAR alors la fonction LOCAL devra être utilisée à l'intérieur du bloc avec pour arguments les noms des variables.

La valeur d'un bloc est celle de la dernière instruction ou bien la valeur de l'argument de la fonction RETURN qui peut être utilisée pour sortir explicitement du bloc. La fonction GO peut être utilisée pour transférer le contrôle à l'instruction du bloc qui est marquée comme argument de GO. Pour marquer une instruction, précédez-la d'un argument atomique comme une autre instruction dans le BLOCK. Par exemple: BLOCK([X],X:1,LOOP,X:X+1,...,GO(LOOP),...). L'argument de GO doit être

le nom d'une marque apparaissant dans le BLOCK. On ne peut utiliser GO pour transférer vers une marque d'un BLOCK autre que celui contenant le GO.

Les blocs apparaissent typiquement dans le membre droit d'une définition de fonction mais peuvent être utilisés ailleurs aussi bien.

BREAK (*arg1, ...*) Fonction

va évaluer et afficher ses arguments puis provoquer un (MACSYMA-BREAK) permettant en ce point à l'utilisateur d'examiner et changer son environnement. L'exécution repart en tapant EXIT;.

Contrôle-A (^A) entre un MACSYMA-BREAK depuis tout point, interactivement. EXIT; continuera le calcul. Contrôle-X peut être utilisé à l'intérieur du MACSYMA-BREAK pour sortie localement, sans quitter le calcul principal.

BUILDQ Macro

Voir DESCRIBE(MACROS); .

CATCH (*exp1,...,expn*) Fonction

évalue ses arguments un à un; si la structure de expi mène à une évaluation d'une expression de la forme THROW(arg), alors la valeur du CATCH est celle de THROW(arg). Ce "retour non local" parcourt ainsi toute profondeur d'imbrication jusqu'au CATCH de fermeture le plus proche. Il doit y avoir un CATCH correspondant à un THROW, sinon une erreur est générée. Si l'évaluation de expi ne mène pas à celle d'un THROW alors la valeur du CATCH est celle de expn.

```
(C1) G(L):=CATCH(MAP(LAMBDA([X],
       IF X<0 THEN THROW(X) ELSE F(X)),L));
(C2) G([1,2,3,7]);
(D2)                    [F(1), F(2), F(3), F(7)]
(C3) G([1,2,-3,7]);
(D3)                                 - 3
```

La fonction G retourne une liste de F de chaque élément de L si L consiste seulement en nombres non négatifs; autrement, G "attrape" le premier élément négatif de L et le "rejette".

COMPFILE ([*filespec*], *f1, f2, ..., fn*) Fonction

Compile les fonctions fi dans le fichier "filespec". Par commodité, voyez la fonction COMPILE.

COMPGRIND Variable

par défaut: [FALSE] - lorsque TRUE les définitions de fonction créées par COMPFILE sont mises en forme.

COMPILE (*f*) Fonction

La commande COMPILE est une propriété commode de macsyma. Elle traite les appels de la fonction COMPFILE, qui traduit les fonctions macsyma en lisp, les appels du compilateur lisp sur le fichier produit par COMPFILE, et le chargement de la sortie du compilateur, connue comme fichier FASL, dans macsyma. Elle vérifie aussi

le fichier de sortie des commentaires du compilateur pour certaines erreurs communes. Faites PRINTFILE(MCOMPI,DOC,MAXDOC); pour plus de détails.

COMPILE(); fait demander des arguments à macsyma.

COMPILE(fonction1,fonction2,...); compile les fonctions, elle utilise le nom de fonction1 comme premier nom du fichier où placer la sortie de lisp.

COMPILE(ALL); ou COMPILE(FUNCTIONS); compilera toutes les fonctions.

COMPILE([nom-fichier],fonction1,fonction2,...); N.B. tous les arguments sont évalués, comme pour une fonction normale (c'est une fonction normale !). Par conséquent, si vous avez des variables avec le même nom comme partie du fichier vous ne pouvez ignorer ce fait.

COMPILE_LISP_FILE ("*input nom-fichier*") Fonction

prend un second argument optionnel de "output nom-fichier", peut être utilisé conjointement avec

```
TRANSLATE_FILE("nom-fichier").
```

Par commodité vous pourriez définir

```
Compile_and_load(FILENAME):=
    LOAD(COMPILE_LISP_FILE(TRANSLATE_FILE(FILENAME)[2]))[2]);
```

Ces commandes orientées fichier doivent être préférées à l'utilisation de COMPILE, COMPFILE, et de la combinaison TRANSLATE SAVE.

DEFINE (*f*(*x1, ...*), *body*) Fonction

est équivalente à f(x1,...):="corps mais lorsqu'utilisée à l'intérieur des fonctions elle se produit au moment de l'exécution plutôt que pendant la définition de la fonction qui la contient.

DEFINE_VARIABLE Fonction

(*nom,default-binding,mode,optional-documentation*)

introduit une variable globale dans l'environnement MACSYMA. Destiné aux packages écrits par l'utilisateur, qui sont souvent traduits ou compilés. Ainsi

```
DEFINE_VARIABLE(FOO,TRUE,BOOLEAN);
```

fait ce qui suit :

(1) MODE_DECLARE(FOO,BOOLEAN); le définit pour le traducteur.

(2) Si la variable n'est pas liée, il le fait : FOO:TRUE.

(3) DECLARE(FOO,SPECIAL); le déclare comme étant spécial.

(4) Définit pour lui une propriété d'affectation pour être sûr qu'il ne lui sera jamais donné une valeur d'un mode incorrect. P. ex. FOO:44 serait une erreur si FOO est déclaré comme BOOLEAN.

Voyez DESCRIBE(MODE_DECLARE); pour une liste des "modes" possibles.

Le quatrième argument optionnel est une chaîne de documentation. Lorsque TRANSLATE_FILE est utilisé sur un package qui inclut des chaînes de documentation, un second fichier est créé, en plus du fichier LISP, qui contiendra les chaînes de documentation, convenablement formatées pour leur incorporation dans les manuels, les

fichiers d'utilisation, ou (par exemple) DESCRIBE. Avec toute variable qui a été définie par DEFINE_VARIABLE avec un mode autre que ANY, vous pouvez donner une propriété VALUE_CHECK, qui est une fonction à un argument appelée sur la valeur que l'utilisateur est en train d'affecter à la variable.

```
PUT('G5,LAMBDA([U],IF U#'G5 THEN ERROR("Don't set G5")),
       'VALUE_CHECK);
```

Utilisez DEFINE_VARIABLE(G5,'G5,ANY_CHECK, "ce n'est pas supposé être défini par quelqu'un d'autre que moi.")

ANY_CHECK est un mode qui signifie la même chose qu'ANY, mais qui empêche DEFINE_VARIABLE d'optimiser la propriété affectée.

DISPFUN (*f1, f2, ...*) Fonction

affiche la définition des fonctions f1, f2, ... définies par l'utilisateur qui peuvent aussi être les noms de fonctions de tableaux associés, de fonctions indexées, ou de fonctions à indices constants qui sont les mêmes que ceux utilisés lorsque les fonctions ont été définies.

DISPFUN(ALL) affichera toutes les fonctions définies par l'utilisateur telles que données dans les listes FUNCTIONS et ARRAYS sauf les fonctions indexées par indices constants. P. ex. si l'utilisateur a défini une fonction F(x), DISPFUN(F); affichera la définition.

FUNCTIONS Variable

par défaut: [] - toutes les fonctions définies par l'utilisateur (à l'aide de f(x):=...).

FUNDEF (*nom-fonction*) Fonction

retourne la définition de fonction associée à "nom-fonction". FUNDEF(fnname); est semblable à DISPFUN(fnname);, sauf que FUNDEF n'invoque pas l'affichage.

FUNMAKE (*nom,[arg1,...,argn]*) Fonction

retourne nom(arg1,...,argn) sans appeler la fonction "nom".

LOCAL (*v1, v2, ...*) Fonction

rend les variables v1,v2,... locales par rapport à toutes les propriétés de l'instruction dans laquelle cette fonction est utilisée. LOCAL ne peut être utilisée que dans les BLOCKs, dans le corps d'une définition de fonction ou d'expressions LAMBDA, ou dans la fonction EV et une seule occurrence est permise dans chacune d'elles. LOCAL est indépendante de CONTEXT.

MACROEXPANSION Variable

par défaut:[FALSE] - Contrôle les particularités avancées qui affectent l'efficacité des macros. Paramètrage possible :

FALSE – Les macros sont développées normalement chaque fois qu'elles sont appelées.

EXPAND – La première fois qu'un appel particulier est évalué, le développement est "conservé" en interne, de sorte qu'il ne sera pas recalculé lors des appels suivants, les rendant ainsi plus rapides. La macro peut encore utiliser GRIND et DISPLAY

normalement, cependant de la mémoire supplémentaire est requise pour se "souvenir" de tous les développements.

DISPLACE – La première fois qu'un appel particulier est évalué, le développement remplace l'appel. Ceci requiert un peu moins d'espace de stockage que lorsque EXPAND est l'argument de MACROEXPANSION et est juste aussi rapide, mais a le désavantage que l'appel de la macro d'origine n'est plus conservé et donc que le développement sera vue si DISPLAY ou GRIND est appelée. Voir la documentation pour TRANSLATE et MACROS pour plus de détails.

MODE_CHECKP Variable

par défaut: [TRUE] - Si TRUE, MODE_DECLARE vérifie les modes des variables liées.

MODE_CHECK_ERRORP Variable

par défaut: [FALSE] - Si TRUE, MODE_DECLARE appelle "error".

MODE_CHECK_WARNP Variable

par défaut: [TRUE] - Si TRUE, les erreurs de mode sont décrites.

MODE_DECLARE (*y1, mode1, y2, mode2, ...*) Fonction

MODEDECLARE en est un synonyme. MODE_DECLARE est utilisée pour déclarer les modes des variables et fonctions pour la traduction ou la compilation de fonctions ultérieures. Ses arguments sont des paires formées d'une variable yi, et d'un mode qui est BOOLEAN, FIXNUM, NUMBER, RATIONAL, ou FLOAT. Chaque yi peut aussi être une liste de variables, toutes étant déclarées comme ayant le modei. Si yi est un tableau, et si chaque élément du tableau qui est référencé a une valeur alors ARRAY(yi, COMPLETE, dim1, dim2, ...) plutôt que

```
ARRAY(yi, dim1, dim2, ...)
```

devra être utilisé dès la déclaration des bornes du tableau. Si tous les éléments du tableau ont le mode FIXNUM (FLOAT), utilisez FIXNUM (FLOAT) au lieu de COMPLETE. Et si tous les éléments du tableau ont même mode, disons m, alors

```
MODE_DECLARE(COMPLETEARRAY(yi),m))
```

devra être utilisé pour une traduction efficace. Du code numérique utilisant des tableaux peut aussi être accéléré en déclarant la taille attendue du tableau, comme ceci :

```
MODE_DECLARE(COMPLETEARRAY(A[10,10]),FLOAT)
```

pour un tableau de nombres en virgule flottante de 10 x 10. On peut de plus déclarer le mode du résultat d'une fonction en utilisant FUNCTION(F1,F2,...) comme argument; ici F1,F2,... sont les noms de fonctions. Par exemple l'expression,

```
MODE_DECLARE([FUNCTION(F1,F2,...),X],FIXNUM,Q,
                                    COMPLETEARRAY(Q),FLOAT)
```

déclare que X et les valeurs retournées par F1,F2,... sont des entiers d'un seul mot et que Q est un tableau de nombres en virgule flottante.

MODE_DECLARE est utilisé soit immédiatement à l'intérieur d'une définition de fonction, soit au niveau haut pour les variables globales. Faites PRINTFILE(MCOMPI,DOC,MAXDOC); pour quelques exemples de l'utilisation de MODE_DECLARE en traduction et compilation.

MODE_IDENTITY (*arg1,arg2*) Fonction

Une forme spéciale utilisée avec MODE_DECLARE et MACROS pour déclarer, p. ex., une liste de listes de "flonums" (nombres "flottants"), ou d'autres objets de donnée composés. Le premier argument de MODE_IDENTITY est une valeur primitive du nom de mode telle que donnée à MODE_DECLARE (i.e. [FLOAT,FIXNUM,NUMBER, LIST,ANY]), et le second argument est une expression qui est évaluée et retournée comme valeur de MODE_IDENTITY. Cependant, si la valeur retournée n'est pas allouée par le mode déclaré dans le premier argument, une erreur ou un avertissement sera signalé. La chose importante est que le MODE de l'expression tel que déterminé par MACSYMA vers le traducteur Lisp, sera celui donné comme premier argument, indépendamment de ce qui se trouve dans le second argument.

P. ex. X:3.3; MODE_IDENTITY(FIXNUM,X); est une erreur. MODE_IDENTITY(FLONUM,X) retourne 3.3.

Ceci a nombre d'utilisations, p. ex., si vous savez que FIRST(L) retournera un nombre alors vous pouvez écrire MODE_IDENTITY(NUMBER,FIRST(L)). Cependant, une façon plus efficace de la faire serait de définir une nouvelle primitive,

```
FIRSTNUMB(X)::=BUILDQ([X],MODE_IDENTITY(NUMBER,X));
```

et d'utiliser FIRSTNUMB chaque fois que vous prenez le premier d'une liste de nombres.

TRANSBIND Variable

par défaut: [FALSE] - si TRUE enlève les déclarations globales du contexte local. Ceci s'applique aux variables qui sont des paramètres formels de l'une des fonctions de TRANSLATE qui traduisent du code MACSYMA en LISP.

TRANSCOMPILE Variable

par défaut:[FALSE] - si TRUE, TRANSLATE génère les déclarations nécessaires pour rendre possible la compilation. La commande COMPFILE utilise TRANSCOMPILE:TRUE;.

TRANSLATE (*f1, f2, ...*) Fonction

traduit les fonctions définies par l'utilisateur f1,f2,... du langage MACSYMA en LISP (i.e. elle en fait des EXPR). Ceci produit un gain de vitesse lorsqu'elles sont appelées. Il existe maintenant une version de macsyma avec un traducteur macsyma vers lisp intégré. Il est disponible en tapant :TM (pour TranslateMacsyma) au niveau DDT. Si un nom de fichier lui est donné, p. ex. :TM GJC;TMTEST >, il envoie ce fichier à la fonction TRANSLATE_FILE, et le traite sans autre intervention de l'utilisateur. Si aucun nom de fichier n'est donné, :TM donne une ligne macsyma régulière "(C1)". N.B. : un fichier utilisateur init de second nom "TM" sera chargé s'il existe. Vous pouvez juste vouloir lier celui-ci à votre fichier macsyma init.

Les fonctions à traduire devront contenir si possible au début un appel à MODE_DECLARE, afin de produire du code plus efficace. Par exemple:

```
F(X1,X2,...):=BLOCK([v1,v2,...],
        MODE_DECLARE(v1,mode1,v2,mode2,...),...)
```

où les X1,X2,... sont les paramètres de la fonction et les v1,v2,... les variables locales. Les noms des fonctions traduites sont enlevés de la liste FUNCTIONS si SAVEDEF est FALSE (voir ci-dessous) et sont ajoutés aux listes PROPS. Les fonctions ne devront pas être traduites à moins qu'elles soient complètement déboguées. De plus, les expressions sont supposées simplifiées ; si elles ne le sont pas, du code correct mais non optimal est créé. Ainsi, l'utilisateur ne devra pas mettre le commutateur SIMP à FALSE, ce qui empêcherait la simplification des expressions à traduire.

Le commutateur TRANSLATE, par défaut: [FALSE], si TRUE, provoque la traduction automatique des fonctions utilisateur en LISP. Notez que les fonctions traduites peuvent ne pas s'exécuter comme elles le faisaient avant la traduction car certaines incompatibilités peuvent exister entre les versions de LISP et de MACSYMA. En particulier, la fonction RAT avec plus d'un argument et la fonction RATVARS ne devraient pas être utilisées si des variables sont MODE_DECLAREd CRE. Et le paramétrage PREDERROR:FALSE ne sera pas traduit.

SAVEDEF[TRUE] - si TRUE provoque la sauvegarde de la version MACSYMA d'une fonction utilisateur lorsqu'elle est TRANSLATE. Ce qui permet à la définition d'être affichée par DISPFUN et à la fonction d'être éditée.

TRANSRUN[TRUE] - si FALSE provoque l'exécution de la version interprétée de toutes les fonctions (pourvu qu'elles soient encore là) plutôt que celle de la version traduite.

On peut traduire des fonctions stockées dans un fichier en donnant à TRANSLATE un argument qui est une spécification de fichier. C'est une liste de la forme [fn1,fn2,DSK,dir] où fn1 fn2 est le nom du fichier des fonctions MACSYMA, et dir est le nom du répertoire du fichier. Le résultat retourné par TRANSLATE est une liste des noms des fonctions traitées par TRANSLATE. Dans le cas d'un fichier de traduction l'élément correspondant de la liste est une liste des premier et second nouveaux noms de fichier contenant le code LISP résultant de la traduction. Ce sera fn1 LISP dans le répertoire sur le disque. Le fichier de code LISP peut être lu dans MACSYMA avec la fonction LOADFILE.

TRANSLATE_FILE (*file*) Fonction

traduit un fichier de code MACSYMA en un fichier de code LISP. Elle prend un ou deux arguments. Le premier argument est le nom du fichier MACSYMA, et le second argument optionnel celui du fichier LISP à produire. Le second argument est par défaut le nom du premier argument avec pour second nom de fichier la valeur de TR_OUTPUT_FILE_DEFAULT qui est par défaut TRLISP. Par exemple: TRANSLATE_FILE("test.mc")); traduira "test.mc" en "test.LISP". Un fichier de messages d'avertissement de traduction est aussi produit, de divers degrés de sévérité. Le second nom de fichier est toujours UNLISP. Ce fichier contient des informations valables (bien que quelquefois obscures) pour tracer les bogues du code traduit.

Faites APROPOS(TR_) pour avoir une liste de commutateurs TR (pour TRANSLATE).

En résumé, TRANSLATE_FILE("foo.mc"), LOADFILE("foo.LISP") est "=" à BATCH("foo.mc") modulo certaines restrictions (l'utilisation de " et % par exemple).

TRANSRUN Variable

par défaut: [TRUE] - si FALSE provoque l'exécution de la version interprétée de toutes les fonctions (pourvu qu'elles soient encore là) plutôt que de la version traduite.

TR_ARRAY_AS_REF Variable

par défaut: [TRUE] - si TRUE le code d'exécution utilise la valeur de la variable comme tableau.

TR_BOUND_FUNCTION_APPLYP Variable

par défaut: [TRUE] - Donne un avertissement si une variable liée se trouve utilisée comme une fonction.

TR_FILE_TTY_MESSAGESP Variable

par défaut: [FALSE] - Détermine si les messages générés par TRANSLATE_FILE pendant la traduction d'un fichier seront envoyés au TTY. Si FALSE (par défaut), les messages concernant la traduction du fichier seront seulement insérés dans le fichier UNLISP. Si TRUE, les messages sont envoyés au TTY et sont aussi insérés dans le fichier UNLISP.

TR_FLOAT_CAN_BRANCH_COMPLEX Variable

par défaut: [TRUE] - Spécifie si les fonctions arc doivent retourner des résultats complexes. Les fonctions arc sont SQRT, LOG, ACOS, etc. P. ex., si elle est TRUE alors ACOS(X) sera en mode ANY même si X est en mode FLOAT. Si FALSE alors ACOS(X) sera en mode FLOAT si et seulement si X est en mode FLOAT.

TR_FUNCTION_CALL_DEFAULT Variable

par défaut: [GENERAL] - FALSE signifie abandonner et appeler MEVAL, EXPR signifie présumer que Lisp fixe l'argument de la fonction. GENERAL, le défaut, donne du code convenant à MEXPRS et MLEXPRS mais pas à MACROS.

GENERAL assure que les liens des variable sont corrects dans le code compilé. En mode GENERAL, lors de la traduction de F(X), si F est une variable liée, alors il est supposé que APPLY(F,[X]) est signifié, et traduit comme tel, avec un avertissement approprié. Désactiver ceci est inutile. Avec les paramétres par défaut, aucun message d'avertissement n'implique une complète compatibilité entre le code traduit et compilé avec l'interpréteur de macsyma.

TR_GEN_TAGS Variable

par défaut: [FALSE] - Si TRUE, TRANSLATE_FILE génère un fichier TAGS pour utilisation avec l'éditeur de texte.

TR_NUMER Variable

par défaut: [FALSE] - Si TRUE les propriétés numériques sont utilisées pour les atomes les possédant, p. ex. %PI.

TR_OPTIMIZE_MAX_LOOP Variable

par défaut: [100] - Le nombre maximum de fois que la phase de développement macro et d'optimisation du traducteur bouclera en considérant une forme. C'est pour trouver les erreurs de développement MACRO, et les propriétés d'optimisation ne se terminant pas.

TR_OUTPUT_FILE_DEFAULT Variable

par défaut: [TRLISP] - C'est le second nom de fichier à être utilisé pour traduire la sortie lisp.

TR_PREDICATE_BRAIN_DAMAGE Variable

par défaut: [FALSE] - Si TRUE, sort les multiples évaluations possibles en tentant de s'interfacer avec le package COMPARE.

TR_SEMICOMPILE Variable

par défaut: [FALSE] - Si TRUE TRANSLATE_FILE et COMPFILE sortent des formes qui seront développées mais non compilées en code machine par le compilateur lisp.

TR_STATE_VARS Variable

par défaut :

```
[TRANSCOMPILE, TR_SEMICOMPILE,
TR_WARN_UNDECLARED, TR_WARN_MEVAL, TR_WARN_FEXPR, TR_WARN_MODE,
TR_WARN_UNDEFINED_VARIABLE, TR_FUNCTION_CALL_DEFAULT,
 TR_ARRAY_AS_REF,TR_NUMER]
```

La liste des commutateurs qui affectent la forme de la sortie traduite. Cette information est utile pour les ingénieurs système lorsqu'ils essaient de déboguer le traducteur. En comparant le produit traduit à ce qui aurait dû être obtenu pour un état donné, il est possible de traquer les bogues.

TR_TRUE_NAME_OF_FILE_BEING_TRANSLATED Variable

par défaut : [FALSE] est liée à la chaîne entre guillemets qui forme le nom réel du fichier le plus récemment traduit par TRANSLATE_FILE.

TR_VERSION Variable

Le numéro de version du traducteur.

TR_WARNINGS_GET () Fonction

Affiche une liste d'avertissements qui ont été donnés par le traducteur pendant la traduction actuelle.

TR_WARN_BAD_FUNCTION_CALLS Variable

par défaut: [TRUE] - Donne un avertissement lorsque des appels de fonction sont faits qui peuvent être incorrects à cause de déclarations impropres faites au moment de la traduction.

TR_WARN_FEXPR Variable

par défaut: [COMPFILE] - Donne un avertissement si des FEXPR sont rencontrées. Les FEXPR ne devraient pas normalement être en sortie du code traduit, toutes les formes légitimes spéciales du programme sont traduites.

TR_WARN_MEVAL Variable

par défaut: [COMPFILE] - Donne un avertissement si la fonction MEVAL est appelée. Si MEVAL est appelée ceci indique des problèmes avec la traduction.

TR_WARN_MODE Variable

par défaut: [ALL] - Donne un avertissement lorsque des variables reçoivent des valeurs inappropriées à leur mode.

TR_WARN_UNDECLARED Variable

par défaut: [COMPILE] - Détermine quand envoyer au TTY des avertissements à propos de variables non déclarées.

TR_WARN_UNDEFINED_VARIABLE Variable

par défaut: [ALL] - Donne un avertissement lorsque des variables globales non définies sont rencontrées.

TR_WINDY Variable

par défaut: [TRUE] - Génère des commentaires "utiles" et des astuces de programmation.

UNDECLAREDWARN Variable

par défaut: [COMPFILE] - Un commutateur dans le traducteur. Il y a quatre paramètres s'y rapportant :

FALSE : n'affiche jamais les messages d'avertissement.

COMPFILE : avertir lorsque dans COMPFILE

TRANSLATE : avertir lorsque dans TRANSLATE et que TRANSLATE:TRUE

ALL : avertir lorsque dans COMPFILE et TRANSLATE

Faites MODE_DECLARE(<variable>,ANY) pour déclarer une variable comme étant une variable macsyma générale (i.e. non limitée à FLOAT ou FIXNUM). Le travail supplémentaire en déclarant toutes vos variables en code à compiler devrait payer.

COMPILE_FILE (*nom-fichier*,&optional-outfile) Fonction

Prend nom-fichier qui contient du code macsyma, et le traduit en lisp puis compile le résultat. Elle retourne une liste de quatre fichiers (l'original, le traduit, les notes sur la traduction et le code compilé).

DECLARE_TRANSLATED (*FN1,FN2..*) Fonction

Lors de la traduction d'un fichier de code macsyma en lisp, il est important pour le traducteur de savoir quelles fonctions du fichier sont des fonctions à traduire ou à compiler, et lesquelles sont juste des fonctions macsyma ou indéfinies. Placer cette déclaration au début du fichier, lui fait savoir qu'un symbole n'ayant pas encore une valeur de fonction lisp, en aura une au moment de l'appel. (MFUNCTION-CALL fn arg1 arg2..) est générée lorsque le traducteur ne sait pas si fn sera une fonction lisp.

38 Flot du programme

38.1 Déroulement du programme

MACSYMA fournit une boucle DO pour l'itération, ainsi que des constructions plus primitives telle que GO.

38.2 Définitions

BACKTRACE Variable

par défaut: [] (lorsque DEBUGMODE:ALL a été exécutée) a comme valeur une liste de toutes les fonctions actuellement entrées.

DO Opérateur spécial

L'instruction DO est utilisée pour accomplir une itération. à cause de sa grande généralité l'instruction DO sera décrite en deux parties : en premier, la forme donnée en général, analogue à celle utilisée dans plusqieurs autres langages de programmation (FORTRAN, ALGOL, PL/I, etc.); puis les autres particularités seront ensuite mentionnées.

1. Il existe trois variantes de cette forme qui diffèrent seulement en leur conditions de terminaison. Ce sont :

- (a) FOR variable : valeur-initiale STEP incrément THRU limite DO corps
- (b) FOR variable : valeur-initiale STEP incrément WHILE condition DO corps
- (c) FOR variable : valeur-initiale STEP incrément UNLESS condition DO corps

(Alternativement, le STEP peut être donné après la condition de terminaison ou la limite).

La valeur-initiale, l'incrément, la limite, et le corps peuvent être des expressions quelconques. Si l'incrément est 1 alors "STEP 1" peut être omis.

L'exécution de l'instruction DO commence par affecter la valeur-initiale à la variable (désormais appelée variable de contrôle). Puis: (1) Si la variable de contrôle a dépassé la limite de la spécification THRU, ou si la condition du UNLESS est TRUE, ou si la condition du WHILE est FALSE alors le DO se termine. (2) Le corps est évalué. (3) L'incrément est ajouté à la variable de contrôle. Le processus de (1) à (3) est répété jusqu'à ce que la condition de terminaison soit satisfaite. On peut aussi donner plusieurs conditions de terminaison, auquel cas le DO se termine lorsque l'une d'elles est satisfaite.

En général le test THRU est satisfait lorsque la variable de contrôle est plus grande que la limite si l'incrément est non négatif, ou lorsque la variable de contrôle est inférieure à la limite si l'incrément est négatif. Incrément et limite peuvent être des expressions non numériques tant que cette inégalité peut être déterminée. Cependant, à moins que l'incrément soit syntaxiquement négatif (p. ex. soit un nombre négatif) au moment de l'entrée de instruction DO, MACSYMA suppose qu'il sera positif

lorsque le DO sera exécuté. S'il ne l'est pas (positif), alors le DO peut ne pas se terminer proprement.

Notez que la limite, l'incrément, et la condition de terminaison sont évalués à chaque fois dans la boucle. Ainsi, si l'un de ceux-ci implique un calcul et renvoie un résultat qui ne change pas pendant toutes les exécutions du corps, il est alors plus efficace de définir une variable de cette valeur avant le DO et d'utiliser cette variable dans la forme DO.

La valeur normalement retournée par une instruction DO est l'atome DONE, car toute instruction dans MACSYMA retourne une valeur. Cependant, la fonction RETURN peut être utilisée dans le corps pour sortir du DO prématurément et lui donner une valeur désirée. Notez pourtant qu'un RETURN dans un DO se produisant dans un BLOCK sortira seulement du DO et pas du BLOCK. Notez aussi que la fonction GO ne peut être utilisée pour sortir d'un DO situé dans un BLOCK.

La variable de contrôle est toujours locale dans le DO et toute variable peut ainsi être utilisée sans affecter la valeur d'une variable de même nom en dehors du DO. La variable de contrôle n'est plus liée quand le DO se termine.

```
(C1)   FOR A:-3 THRU 26 STEP 7 DO LDISPLAY(A)$
(E1)           A = -3
(E2)           A =  4
(E3)           A = 11
(E4)           A = 18
(E5)           A = 25
```

La fonction LDISPLAY génère des étiquettes intermédiaires; DISPLAY ne le fait pas.

```
(C6)   S:0$
(C7)   FOR I:1 WHILE I<=10 DO S:S+I;
(D7)           DONE
(C8)   S;
(D8)           55
```

Notez que la condition en C7 est équivalente à UNLESS I > 10 et aussi à THRU 10

```
(C9)   SERIES:1$
(C10)  TERM:EXP(SIN(X))$
(C11)  FOR P:1 UNLESS P>7 DO
          (TERM:DIFF(TERM,X)/P,
          SERIES:SERIES+SUBST(X=0,TERM)*X^P)$
(C12)   SERIES;
                 7    6     5    4    2
(D12)           X    X     X    X    X
                -- - --- - -- - -- + -- + X + 1
                96   240   15   8    2
```

```
qui donne 8 termes de la série de Taylor pour e^sin(x).
(C13) POLY:0$
(C14) FOR I:1 THRU 5 DO
        FOR J:I STEP -1 THRU 1 DO
           POLY:POLY+I*X^J$
(C15) POLY;
              5      4       3       2
```

```
(D15)      5 X  + 9 X  + 12 X  + 14 X  + 15 X
(C16) GUESS:-3.0$
(C17) FOR I:1 THRU 10 DO (GUESS:SUBST(GUESS,X,.5*(X+10/X)),
         IF ABS(GUESS^2-10)<.00005 THEN RETURN(GUESS));
(D17)                  - 3.1622807
```

Cet exemple calcule la racine carrée négative de 10 en utilisant 10 fois au plus l'itération de Newton- Raphson. Si le critère de convergence n'avait pas été fixé, la valeur retournée aurait été DONE.

38.2.1 Formes supplémentaires de l'instruction DO.

Au lieu de toujours additionner une quantité à la variable de contrôle on peut quelquefois vouloir la changer d'une autre façon à chaque itération. Pour cela on peut utiliser "NEXT expression" au lieu de "STEP incrément". Ceci donnera à la variable de contrôle le résultat de l'évaluation de l'expression à chaque boucle d'itération.

```
(C1)  FOR COUNT:2 NEXT 3*COUNT THRU 20
         DO DISPLAY(COUNT)$
                    COUNT = 2
                    COUNT = 6
                    COUNT = 18
```

Comme alternative à FOR variable:valeur ...DO... la syntaxe FOR variable FROM valeur ...DO... peut être utilisée. Ceci permets au "FROM valeur" d'être placé après le STEP ou NEXT valeur ou après la condition de terminaison. Si "FROM valeur" est omis alors 1 est utilisé comme valeur initiale.

On peut quelquefois être intéressé par l'exécution d'une itération où la variable de contrôle n'est en fait jamais utilisée. Il est ainsi possible de donner seulement les conditions de terminaison en omettant l'initialisation et la mise à jour comme dans l'exemple suivant, qui calcule la racine carrée de 5 en utilisant une estimation initiale faible.

```
(C1) X:1000;
(C2)  THRU 10 WHILE X#0.0 DO X:.5*(X+5.0/X)$
(C3) X;
(D3)               2.236068
```

On peut même au besoin omettre entièrement les conditions de terminaison et donner juste "DO corps" qui continuera à évaluer le corps indéfiniment. Dans ce cas la fonction RETURN devra être utilisée pour terminer l'exécution du DO.

```
(C1) NEWTON(F,GUESS):=
  BLOCK([NUMER,Y],
        LOCAL(DF),
        NUMER:TRUE,
        DEFINE(DF(X),DIFF(F(X),X)),
        DO (Y:DF(GUESS),
            IF Y=0.0 THEN ERROR("Derivative at:",GUESS," is zero."),
            GUESS:GUESS-F(GUESS)/Y,
            IF ABS(F(GUESS))<5.0E-6 THEN RETURN(GUESS)))$
(C2) SQR(X):=X^2-5.0$
```

```
(C3) NEWTON(SQR,1000);
(D3)                    2.236068
```

Notez que RETURN, lorsqu'exécuté, retourne la valeur courante de GUESS comme valeur de DO. Le BLOCK est quitté et cette valeur du DO est retournée comme étant celle du BLOCK car le DO est la dernière instruction dans le bloc.

Une autre forme de DO est disponible dans MACSYMA. La syntaxe en est :

```
FOR variable IN liste [end-tests] DO corps
```

Les membres de la liste sont des expressions quelconques qui seront successivement affectées à la variable à chaque itération du corps. Le "end-tests" optionnel peut être utilisé pour terminer l'exécution du DO; autrement il se terminera lorsque la liste sera épuisée ou lorsqu'un RETURN sera exécuté dans le corps. En fait, la liste peut être toute expression non atomique, ses éléments successifs étant pris tour à tour.

```
(C1)  FOR F IN [LOG, RHO, ATAN] DO LDISP(F(1))$
(E1)                                  0
(E2)                                RHO(1)
                                     %PI
(E3)                                 ---
                                      4
(C4) EV(E3,NUMER);
(D4)                           0.78539816
```

ERRCATCH (*exp1, exp2, ...*) Fonction

évalue ses arguments un par un et retourne une liste de la valeur du dernier si aucune erreur ne s'est produite. Si une erreur se produit dans l'évaluation d'un des arguments, ERRCATCH "attrape" l'erreur et retourne immédiatement [] (la liste vide). Cette fonction est utile dans les fichiers BATCH où on suspecte qu'une erreur peut se produire qui autrement aurait terminé le BATCH si elle n'était pas prise en compte.

ERREXP Variable

par défaut: [ERREXP] - Lorsqu'une erreur se produit au cours d'un calcul, MACSYMA affiche un message d'erreur et termine le calcul. ERREXP est pointée sur l'expression en faute et le message "ERREXP contains the offending expression" est affiché. L'utilisateur peut alors taper ERREXP; pour le voir et, on l'espère, corriger le problème.

ERROR (*arg1, arg2, ...*) Fonction

va évaluer et afficher ses arguments puis provoque un retour au niveau haut de MACSYMA ou au plus proche ERRCATCH imbriqué. C'est utile pour sortir de fonctions imbriquées si une condition d'erreur est détectée, ou si on ne peut taper contrôle-^.

La variable ERROR est pointée sur une liste décrivant l'erreur, le premier élément étant une chaîne de texte, et le reste les objets en question.

ERRORMSG(); est la méthode préférée pour voir le dernier message d'erreur.

ERRORFUN, par défaut: [FALSE] - si définie sur le nom d'une fonction sans argument provoque l'exécution de cette fonction dès qu'une erreur se produit. C'est utile

dans les fichiers BATCH où l'utilisateur peut vouloir que son MACSYMA se termine ou sortir de son terminal en cas d'erreur. Dans ces cas ERRORFUN sera pointée sur QUIT ou LOGOUT.

ERRORFUN Variable

par défaut: [FALSE] - si elle contient le nom d'une fonction sans argument, provoque l'exécution de cette fonction dès qu'une erreur se produit. Utile dans les fichiers BATCH où l'utilisateur peut vouloir "tuer" son MACSYMA ou sortir de son terminal si une erreur se produit. Dans ces cas ERRORFUN sera QUIT ou LOGOUT.

ERRORMSG () Fonction

ré-affiche le dernier message d'erreur. Très utile si vous utiliser une console d'affichage et que le message est sorti de l'écran. La variable ERROR pointe sur une liste décrivant l'erreur, son premier élément est une chaîne de texte, et le reste les objets en question.

TTYINTFUN:LAMBDA([],ERRORMSG(),PRINT(""))$ définit le caractère d'interruption (^U) de l'utilisateur pour ré-afficher le message.

FOR Opérateur spécial

Utilisé en itérations, faites DESCRIBE("DO"); pour une description des possibilités d'itération de MACSYMA.

GO (*tag*) Fonction

est utilisé dans un BLOCK pour transférer le contrôle à l'instruction du bloc qui est marquée avec l'argument de GO. Pour marquer une instruction, faites-la précéder d'un argument atomique comme autre instruction du BLOCK. Par exemple:

```
BLOCK([X],X:1,LOOP,X+1,...,GO(LOOP),...)
```

L'argument de GO doit être le nom de la marque apparaissant dans le même BLOCK. On ne peut utiliser GO pour transférer vers une marque d'un BLOCK autre que celui contenant le GO.

IF Opérateur spécial

L'instruction IF est utilisée pour l'exécution conditionnelle. La syntaxe est :

```
IF condition THEN expression1 ELSE expression2.
```

Le résultat d'une instruction IF est expression1 si condition est vraie et expression2 si elle est fausse. expression1 et expression2 sont des expressions MACSYMA quelconques (y compris des instructions IF imbriquées), et condition est une expression qui s'évalue à TRUE ou FALSE et est composée d'opérateurs relationnels et logiques qui sont les suivants :

Nom de l'opérateur	Symbol	Type
plus grand que	>	infixe relationnel
égal à	= , EQUAL	" "
non égal à	#	" "
inférieur à	<	" "

```
plus grand que          >=
  ou égal à                          "  "
inférieur à             <=
  ou égal à                          "  "
et                     AND          infixe logique
ou                      OR              "  "
non                     NOT          préfixe logique
```

LISPDEBUGMODE () Fonction

L'utilisateur dispose des possibilités de débogage LISPDEBUGMODE(); DEBUGPRINTMODE(); et DEBUG(); utilisées par les programmeurs système. Ces outils sont puissants, et bien que quelques conventions soient différentes du niveau usuel de macsyma, on pense que leur utilisation est très intuitive. [Certains affichages peuvent être bavards sur des terminaux lents, mais des commutateurs contrôlent cela]. Ces commandes ont été conçues pour l'utilisateur qui doit déboguer du code macsyma traduit, comme telles elles sont une aubaine. Voyez MACDOC;TRDEBG USAGE pour plus d'information. Pour plus d'aide, consultez GJC.

MAP (*fn, exp1, exp2, ...*) Fonction

retourne une expression dont l'opérateur de tête est le même que celui des expi mais dont les sous-parties résultent de l'application de fn aux sous-parties correspondantes des expi. Fn est soit le nom d'une fonction de n arguments (où n est le nombre de expi), soit une forme LAMBDA de n arguments.

MAPERROR[TRUE] - si FALSE provoque l'application des fonctions : (1) s'arrête lorsqu'elles atteignent le plus court des expi si tous les expi n'ont pas la même longueur; (2) appliquent fn à [exp1, exp2,...] si les expi ne sont pas le même type d'objet. Si MAPERROR est TRUE alors un message d'erreur sera envoyé dans ces deux cas.

L'une des utilisation de cette fonction est d'appliquer MAP à une fonction (e.g. PARTFRAC) sur chaque terme d'une très grande expression où il ne serait en général pas possible d'utiliser la fonction sur l'expression toute entière à cause de l'épuisement de l'espace de stockage de liste lors du calcul.

```
(C1) MAP(F,X+A*Y+B*Z);
(D1)                        F(B Z) + F(A Y) + F(X)
(C2) MAP(LAMBDA([U],PARTFRAC(U,X)),X+1/(X^3+4*X^2+5*X+2));
                             1       1        1
(D2)                       ----- - ----- + -------- + X
                           X + 2   X + 1          2
                                             (X + 1)
(C3) MAP(RATSIMP, X/(X^2+X)+(Y^2+Y)/Y);
                                        1
(D3)                              Y + ----- + 1
                                    X + 1
(C4) MAP("=",[A,B],[-0.5,3]);
(D4)                           [A = - 0.5, B = 3]
```

MAPATOM (*expr*) Fonction

est TRUE si et seulement si expr est traitée par les routines de MAPping comme un "atome", une unité. "Mapatoms" sont des atomes, des nombres (y compris les nombres rationnels), et des variables indicées.

MAPERROR Variable

par défaut: [TRUE] - si FALSE provoque l'application à toutes les fonctions, par exemple

```
MAP(fn,exp1,exp2,...))
```

(1) s'arrête lorsqu'elles en ont terminé avec le plus court des expi si tous les expi n'ont pas la même longueur et (2) applique fn à [exp1, exp2,...] si les expi ne sont pas tous du même type d'objet. Si MAPERROR est TRUE alors un message d'erreur est envoyé dans les deux cas précédents.

MAPLIST (*fn, exp1, exp2, ...*) Fonction

retourne une liste des applications de fn aux parties des expi. Ceci diffère de MAP(fn,exp1,exp2,...), qui retourne une expression avec le même opérateur principal que celui des expi (sauf pour les simplifications et dans le cas où MAP fait un APPLY). Fn a la même forme que dans MAP.

PREDERROR Variable

par défaut: [TRUE] - Si TRUE, un message d'erreur est envoyé dès que le prédicat d'une instruction IF ou une fonction IS échoue à évaluer TRUE ou FALSE. Si FALSE, UNKNOWN est retourné dans ce cas. Le mode PREDERROR:FALSE n'est pas supporté dans le code traduit.

RETURN (*valeur*) Fonction

peut être utilisé pour sortir explicitement d'un BLOCK, en apportant son argument. Faites DESCRIBE(BLOCK); pour plus d'information.

SCANMAP (*fonction,exp*) Fonction

applique récursivement une fonction à exp, d'une manière "du haut en bas". Surtout utile lorsqu'une factorisation "complète" est envisagée, par exemple:

```
(C1) EXP:(A^2+2*A+1)*Y + X^2$
(C2) SCANMAP(FACTOR,EXP);
                                      2      2
(D2)                          (A + 1)  Y + X
```

Notez la façon par laquelle SCANMAP applique la fonction donnée FACTOR aux sous-expressions constituant exp; si une autre forme de exp est présentée à SCANMAP alors le résultat pourra être différent. Ainsi, D2 n'est pas récupéré lorsque SCANMAP est appliquée à la forme développée de exp:

```
(C3) SCANMAP(FACTOR,EXPAND(EXP));
                              2                    2
(D3)                         A  Y + 2 A Y + Y + X
```

Voici un autre exemple de la façon dont SCANMAP applique récursivement une fonction donnée à toutes les sous-expressions, y compris les exposants :

```
(C4) EXPR : U*V^(A*X+B) + C$
(C5) SCANMAP('F, EXPR);
                      F(F(F(A) F(X)) + F(B))
(D5) F(F(F(U) F(F(V                         )) + F(C))
```

SCANMAP(fonction,expression,BOTTOMUP) applique fonction à exp de manière "bas vers haut". P. ex., pour F indéfinie,

```
SCANMAP(F,A*X+B) ->
   F(A*X+B) -> F(F(A*X)+F(B)) -> F(F(F(A)*F(X))+F(B))
SCANMAP(F,A*X+B,BOTTOMUP) -> F(A)*F(X)+F(B)
    -> F(F(A)*F(X))+F(B) ->
     F(F(F(A)*F(X))+F(B))
```

Dans ce cas, vous avez la même réponse des deux façons.

THROW (*exp*) Fonction

évalue exp et renvoie la valeur vers le plus récent CATCH. THROW est utilisé avec CATCH comme mécanisme structuré de sortie non local.

39 Débogage

39.1 Débogage au niveau source

Maxima a des possibilités au niveau source. Un utilisateur peut placer un point d'arrêt sur une ligne d'un fichier, puis le parcourir ligne par ligne à partir de là. La pile des appels peut être examinée, ainsi que les variables liées à ce niveau. Si l'utilisateur a lancé le code sous GNU emacs dans une fenêtre du shell (dbl shell), ou s'il est en train d'exécuter `xmaxima`, la version d'interface graphique, dès qu'il atteint un point d'arrêt, il y verra sa position courante dans le fichier source, qui est affiché dans l'autre moitié de la fenêtre, mis en valeur en rouge, ou avec une petite flèche pointant sur la bonne ligne. Il peut avancer une ligne à la fois en tapant M-n (Alt-n) ou alternativement en entrant `:n`. Pour voir les noms des raccourcis des commandes tapez :help (ou :h). En général les commandes peuvent être abrégées si l'abréviation est unique. Si ce n'est pas le cas une liste donnera les alternatives.

Sous Emacs vous devriez exécuter un shell `dbl`, qui requiert le fichier dbl.el dans le répertoire elisp. Assurez-vous d'installer les fichiers elisp ou d'ajouter le répertoire maxima elisp à votre chemin de recherche : par exemple ajoutez ce qui suit à votre fichier '`.emacs`' ou le `site-init.el`

```
(setq load-path (cons "/usr/local/maxima-5.5/elisp" load-path))
(autoload 'dbl "dbl")
```

puis dans emacs

```
M-x dbl
```

ce qui devrait démarrer une fenêtre shell dans laquelle vous pouvez lancer des programmes, par exemple maxima, gcl, gdb etc. Cette fenêtre shell connaît aussi le débogage au niveau source, et affiche le code source dans l'autre fenêtre.

```
maxima
Maxima 5.5 Wed Apr 18 19:02:00 CDT 2001 (with enhancements by W. Schelter).
Licensed under the GNU Public License (see file COPYING)
(C1) batchload("/tmp/joe.mac");
(D1)                              /tmp/joe.mac
(C2) :br joe
Turning on debugging debugmode(true)
Bkpt 0 for joe (in /tmp/joe.mac line 8)
(C2) foo(2,3);
Bkpt 0:(joe.mac 8)
(dbm:1) :bt                       <-- :bt typed here gives a backtrace
#0: joe(y=5)(joe.mac line 8)
#1: foo(x=2,y=3)(joe.mac line 5)
(joe.mac 9)                       <-- Here type M-n to advance line
(joe.mac 10)                      <-- Here type M-n to advance line
                                  In the other buffer the source code
                                  appears with an arrow.
(dbm:1) u;                        Investigate value of 'u
28
(dbm:1) u:33;                     Alter it to be 33
(dbm:1) :r                        :r Resumes the computation
```

```
(D3)                                  1094
```

Le fichier /tmp/joe.mac est en fait :

```
foo(x,y):=(
  x:x+2,
  y:y+2,
  x:joe(y),
  x+y);

joe(y):=block([u:y^2],
  u:u+3,
  u:u^2,
   u);
```

Si vous êtes dans Gnu Emacs et que vous examinez le fichier joe.mac, vous pouvez placer un point d'arrêt sur une certaine ligne de ce fichier en tapant `C-x espace`. Ceci vous indiquera dans quelle fonction se trouve votre curseur, puis sur quelle ligne de cette fonction vous êtes. Si vous êtes sur la ligne 2 de joe, alors la commande `:br joe 2` sera insérée dans l'autre fenêtre, qui stoppera joe sur sa seconde ligne. Pour que ceci soit disponible, vous devez avoir activé maxima-mode.el dans la fenêtre que le fichier joe.mac est en train de visiter.

D'autres commandes supplémentaires sont disponibles dans cette fenêtre du fichier, telle que l'évaluation de la fonction dans maxima, en tapant `Alt-Contrôle-x`

39.2 Commandes par mot-clé

Les commandes d'arrêt commencent par ':'. Ainsi pour évaluer une forme lisp vous pouvez taper `:lisp` suivi de l'argument, qui est la forme à évaluer.

```
(C3) :lisp (+ 2 3)
5
```

Le nombre d'arguments dépend de la commande particulière. Vous n'avez pas besoin de taper la commande tout entière, juste assez pour qu'elle soit unique parmi les raccourcis. Ainsi `:br` suffira pour `:break`.

Les commandes sont actuellement :

`:break` Place un point d'arrêt dans la FUNCTION spécifiée sur la LINE spécifiée par le décalage depuis le début de la fonction. Si FUNCTION est donnée sous forme d'une chaîne, elle est alors supposée être un fichier FILE et LINE est le décalage depuis le début du fichier.

`:bt` Non documentée

`:continue`
Continue le calcul.

`:delete` Supprime tous les points d'arrêt, ou si des arguments sont donnés, supprime les points d'arrêt spécifiés.

`:disable` Désactive les points d'arrêt spécifiés, ou tous si aucun n'est spécifié.

`:enable` Active les points d'arrêt spécifiés, ou tous si aucun n'est spécifié.

`:frame` Avec un argument affiche la pile de (frame ???) sélectionnée. Sinon la (frame???) courante (active??).

`:help` Affiche l'aide sur une commande break ou sans argument sur toutes les commandes break.

`:info` Non documentée

`:lisp` évalue la forme lisp suivante sur la ligne

`:lisp-quiet`
évalue son argument comme forme lisp sans afficher d'invite.

`:next` Comme :step, sauf que les appels de sous-routine sont sautés.

`:quit` Quitte ce niveau.

`:resume` Continue le calcul..

`:step` Fait avancer le programme jusqu'à ce qu'il atteigne une nouvelle ligne source.

`:top` Revient au niveau haut.

39.3 Définitions pour le débogage

REFCHECK Variable

par défaut: [FALSE] - si TRUE provoque l'affichage d'un message chaque fois qu'une variable liée est utilisée pour la première fois dans un calcul.

REMTRACE () Fonction

Cette fonction n'est plus utilisée dans le nouveau package TRACE.

SETCHECK Variable

par défaut: [FALSE] - Si définie comme liste de variables (qui peuvent être indicées) provoque une sortie (printout imprimée ???) dès que les variables, ou leurs occurrences indicées, sont liées (par : ou :: ou une fonction). La sortie consiste en la variable et la valeur à laquelle elle est liée. SETCHECK peut être ALL ou TRUE ce qui inclut toutes les variables. Note: aucune sortie n'est créée lorsqu'une variable SETCHECK est liée à elle-même, p. ex. X:'X.

SETCHECKBREAK Variable

par défaut: [FALSE] - si TRUE un (MACSYMA-BREAK) se produira dès que les variables sur la liste SETCHECK seront liées. L'arrêt se produit avant que la liaison soit créée. Á ce point, SETVAL contient la valeur à laquelle la variable va être affectée. Par conséquent, on peut changer cette valeur en redéfinissant SETVAL.

SETVAL Variable

contient la valeur à laquelle une variable va être affectée lorsque SETCHECKBREAK se produit. Par conséquent, on peut changer cette valeur en redéfinissant SETVAL. (Voir SETCHECKBREAK).

TIMER (*F*) Fonction

place un compteur de temps sur la fonction F, situé dans le package TRACE, i.e. elle affichera le temps passé pour calculer F.

TIMER_DEVALUE Variable

par défaut: [FALSE] - si TRUE le temps d'une fonction est celui passé dynamiquement dans la fonction moins le temps passé dans d'autres fonctions chronométrées.

TIMER_INFO (*F*) Fonction

affichera l'information de durée stockée par GET('F,'CALLS); GET('F,'RUNTIME); et GET('F,'GCTIME); . C'est une fonction du package TRACE.

TRACE (*nom1, nom2, ...*) Fonction

donne une trace imprimée dès que les fonctions mentionnées sont appelées. TRACE() affiche une liste des fonctions actuellement sous TRACE. Sur MC voir MACDOC;TRACE USAGE pour plus d'information. Et aussi, DEMO("trace.dem");. Pour supprimer la trace, voir UNTRACE.

TRACE_OPTIONS (*F,option1,option2,...*) Fonction

donne à la fonction F les options indiquées. Une option est soit un mot-clé soit une expression.

Les mots-clé possibles sont :

Mot-clé	Sens de la valeur retournée
NOPRINT	Si TRUE ne pas imprimer.
BREAK	Si TRUE donne un point d'arrêt.
LISP_PRINT	Si TRUE utiliser l'impression lisp.
INFO	Info supplémentaire sur print.
ERRORCATCH	Si TRUE les erreurs sont repérées.

Un mot-clé signifie que l'option est activée. Utiliser un mot-clé comme expression, p. ex. NOPRINT(fonction_prédicat) signifie : appliquer la fonction_prédicat (qui est définie par l'utilisateur) à certains arguments pour déterminer si l'option est active. La liste d'arguments de cette fonction_prédicat est toujours [LEVEL, DIRECTION, FUNCTION, ITEM] où LEVEL est le niveau de récursion pour cette fonction. DIRECTION est ENTER ou EXIT. FUNCTION est le nom de la fonction. ITEM est soit la liste d'arguments soit la valeur de retour. Sur MC voir DEMO("trace.dem"); pour plus de détails.

UNTRACE (*nom1, ...*) Fonction

supprime le "tracing" invoqué par la fonction TRACE, pour toutes les fonctions.

40 Index

Appendice A Index des fonctions et variables

%

?

[

"

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Sommaire

Table des matières