Chapitre 7

MODULE DU CALCUL DUAL

7.1 Introduction

Nous avons écrit le module `dualstruc.map` dans le but de disposer d'un ensemble d'outils permettant d'effectuer des calculs sur des données de type duale (nombres duaux, vecteurs duaux, matrices duales, fonctions de variables duale, construction automatique des prolongements canoniques, ...etc). Actuellement, ce module compte un grand nombre de fonctions *auxiliaires* et un nombre plus restreint de fonctions *principales*. Entièrement implémentées dans le langage Maple, elles peuvent être combinées aux fonctionnalités de base du système Maple pour étendre les possibilités du logiciel à la résolution d'un nouveau type de problèmes (utilisant des données duales). Dans la fenêtre (Maple V Release 2) de la figure ci-après, on présente une session de travail sur Maple où l'on expérimente certaines fonctions développées dans le module `dualstruc.map`. Ce chapitre est consacré à présenter les fonctions *principales* définies dans ce module.

7.2 Présentation des constructeurs duaux

Afin de donner un avant goût de la programmation avec les outils définis dans le module `dualstruc.map`, nous allons présenter des exemples simples, mais qui illustrent déjà quelques différences de style avec le calcul classique sur le système Maple. Les fonctions du module - `dualstruc.map` doivent explicitement être chargées avant utilisation.

```
    |\^/|     MAPLE V
._|\|   |/|_.  Copyright (c) 1981-1990 by the University of Waterloo.
 \  MAPLE  /   All rights reserved.  MAPLE is a registered trademark of
 <____ ____>   Waterloo Maple Software.
      |        Type ? for help.

> read `dualstruc.map`:
```

Comme le module `defstruc.map`, le module `dualstruc.map` dispose d'un constructeur d'objets

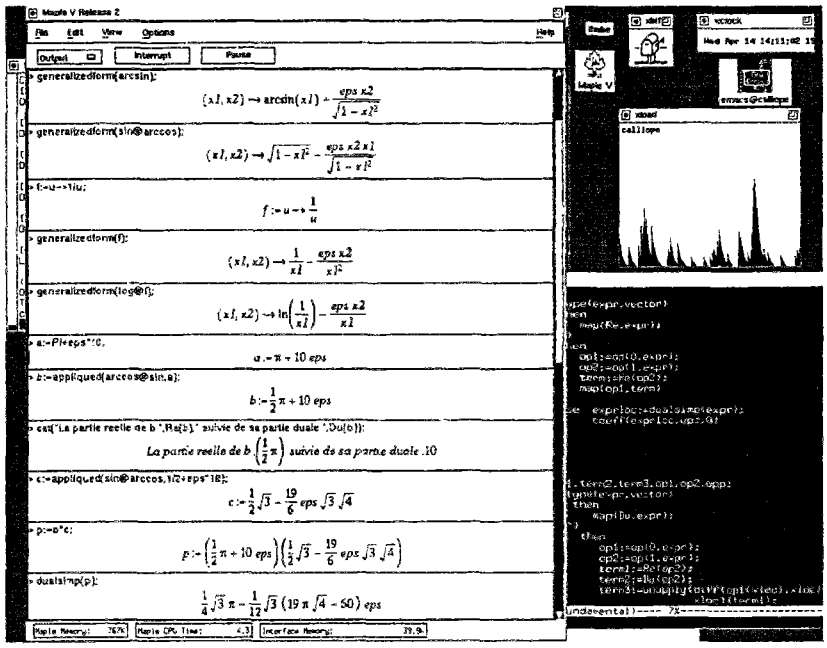

Fig. 7.1: Session de travail sur Maple

matriciels, vectoriels et scalaires. La définition des constructeurs duaux est conforme aux idées développées dans le module `defstruc.map` à la différence près que dans le présent module ces constructeurs permettent de définir de façon autonome des objets de type dual. C'est à dire qu'au niveau morphologique de l'objet, les champs sont des symboles qui représentent des scalaires (i.e. entités élémentaires) de type dual.

Exemple

La manière la plus simple pour construire une matrice duale est de spécifier toutes ses composantes. L'opération principale permettant une telle définition est l'affectation:

```
> Dmat:=array([[1.+7.*eps,e],[3,a+eps*b]]);

                    [ 1. + 7. eps            e          ]
              Dmat := [                                   ]
                    [      3        ra + eps da + eps b ]
```

où le symbole eps désigne l'unité duale ε.

7.2.1 Constructeur de classes

Comme nous l'avons déjà évoqué au chapitre précédent, un constructeur est une fonction d'initialisation que l'on peut appeler (avec des sélecteurs spécifiques) chaque fois que l'on souhaiterait créer une instance d'une classe donnée. Grâce à son constructeur de classes `make_duobj`, le module `dualstruc.map` offre la possibilité de définir des entités duales de façon plus autonome. Le constructeur `make_duobj` permet de créer un objet à partir de son nom et de la spécification de sa nature (classe d'appartenance) et de sa dimension.

La syntaxe d'un tel constructeur est: `make_duobj(iden,struc,dim)`
où:

`iden` : désigne l'identificateur de l'objet qu'on souhaite créer,

`struc` : désigne la structure qu'on souhaite créer,

`dim` : désigne la dimension de l'objet.

La déclaration de la dimension `dim` est inutile lorsque la structure `struc` désigne le type scalaire.

Exemples

```
> make_duobj(A,matrix,3);
                          Class: dual matrix(3,3)

> object(A);
 [ rA[1, 1] + eps dA[1, 1]  rA[1, 2] + eps dA[1, 2]  rA[1, 3] + eps dA[1, 3] ]
 [                                                                           ]
 [ rA[2, 1] + eps dA[2, 1]  rA[2, 2] + eps dA[2, 2]  rA[2, 3] + eps dA[2, 3] ]
 [                                                                           ]
 [ rA[3, 1] + eps dA[3, 1]  rA[3, 2] + eps dA[3, 2]  rA[3, 3] + eps dA[3, 3] ]

> make_duobj(V,vector,2);
                           Class: dual vector(2)

> object(V);
                  [ rV[1] + eps dV[1], rV[2] + eps dV[2] ]

> make_duobj(a,scalar);
                             Class: dual scalar

> object(a);
                                ra + eps da
```

La fonction `object` permet ainsi de visualiser la structure morphologique d'un objet construit à partir de `make_duobj`. Les objets duaux, ainsi créés, comportent une partie réelle dont le nom est un symbole racine préfixée par la lettre `r` et une partie duale préfixée par la lettre `d`.

7.2.2 Opérateur ad d'un torseur

A partir de la donnée des coordonnées duales `X` d'un torseur X, l'opérateur `ad` permet de calculer la matrice duale associée à l'opérateur $ad\,X$ et telle que: $ad\,X\,Y = [X,Y]$ pour tout torseur Y de $\mathfrak{D}$.

Exemples

```
> make_duobj(X,vector,3);
                          Class: dual vector(3)

> EE:=ad(X);
          [          0          - rX[3] - eps dX[3]   rX[2] + eps dX[2]  ]
          [                                                              ]
    EE := [  rX[3] + eps dX[3]           0           - rX[1] - eps dX[1] ]
          [                                                              ]
          [ - rX[2] - eps dX[2]   rX[1] + eps dX[1]           0          ]
```

7.2.3 Déplacements autour des axes d'un repère

La matrice d'un déplacement cylindrique exprimée relativement à une famille fondamentale $\mathfrak{f} = (\xi, \eta, \zeta)$ de l'algèbre de Lie $\mathfrak{D}$ dont l'un des axes est celui du déplacement, s'écrit de façon relativement simple. Le module `dualstruc.map` offre la possibilité de définir de telles matrices. Les fonctions `RD1`, `RD2` et `RD3` permettent de construire les déplacements cylindriques respectivement suivant les premier, second et troisième axes de la famille $\mathfrak{f}$.

Exemples

Si a est le scalaire dual défini dans l'exemple précédent, la *rotation duale*, d'*angle dual* a et d'axe Λ_ξ s'exprime dans la famille fondamentale $\mathfrak{f}$ par:

```
> Dep1:=RD1(a);
              [ 1              0                           0             ]
              [                                                          ]
      Dep1 := [ 0  cos(ra) - eps da sin(ra)  - sin(ra) - eps da cos(ra) ]
              [                                                          ]
              [ 0  sin(ra) + eps da cos(ra)   cos(ra) - eps da sin(ra)  ]

> Dep2:=RD3(Pi/2+eps*c);
                                [ - eps c     -1     0 ]
                                [                      ]
                      Dep2 := [    1     - eps c  0 ]
                                [                      ]
                                [    0        0     1 ]
```

La matrice de tout autre déplacement peut être obtenue à partir des trois matrices `RD1`, `RD2` et `RD3` (grâce à différentes cartes du groupe de Lie $\mathbb{D}$).

7.3 Simplificateurs

Pour mener un processus de calcul formel, étant donné que le but de l'implémentation du module `dualstruc.map` est de pouvoir utiliser, aussi bien des fonctions de base de Maple que des fonctions définies dans ce module, il était nécessaire de concevoir des simplificateurs qui permettent d'appliquer en plus des règles de simplification définies au toplevel, des règles de transformation de type $\varepsilon^2 \longrightarrow 0$. Les simplificateurs les plus importants sont `dualsimp` pour opérer des simplifications sur les nombres duaux, `objdualsimp` pour la simplification des objets duaux, `duwithopt` pour la simplification et l'optimisation des expressions scalaires duales et `objwithopt` pour la simplification et l'optimisation des expressions des champs d'un objet dual quelconque.

Exemples

```
> AEE:=multiply(A,EE);
AEE :=
                                                                    2
[rA[1, 2] rX[3] + rA[1, 2] eps dX[3] + eps dA[1, 2] rX[3] + eps  dA[1, 2] dX[3]

     - rA[1, 3] rX[2] - rA[1, 3] eps dX[2] - eps dA[1, 3] rX[2]

          2
     - eps  dA[1, 3] dX[2], - rA[1, 1] rX[3] - rA[1, 1] eps dX[3]

                                2
     - eps dA[1, 1] rX[3] - eps  dA[1, 1] dX[3] + rA[1, 3] rX[1]

                                                          2
     + rA[1, 3] eps dX[1] + eps dA[1, 3] rX[1] + eps  dA[1, 3] dX[1],

    rA[1, 1] rX[2] + rA[1, 1] eps dX[2] + eps dA[1, 1] rX[2]

          2
     + eps  dA[1, 1] dX[2] - rA[1, 2] rX[1] - rA[1, 2] eps dX[1]

                                2
     - eps dA[1, 2] rX[1] - eps  dA[1, 2] dX[1]]

                                                                    2
[rA[2, 2] rX[3] + rA[2, 2] eps dX[3] + eps dA[2, 2] rX[3] + eps  dA[2, 2] dX[3]

     - rA[2, 3] rX[2] - rA[2, 3] eps dX[2] - eps dA[2, 3] rX[2]

          2
     - eps  dA[2, 3] dX[2], - rA[2, 1] rX[3] - rA[2, 1] eps dX[3]

                                2
     - eps dA[2, 1] rX[3] - eps  dA[2, 1] dX[3] + rA[2, 3] rX[1]

                                                          2
     + rA[2, 3] eps dX[1] + eps dA[2, 3] rX[1] + eps  dA[2, 3] dX[1],

    rA[2, 1] rX[2] + rA[2, 1] eps dX[2] + eps dA[2, 1] rX[2]
```

```
                                                                    2
     + eps  dA[2, 1] dX[2] - rA[2, 2] rX[1] - rA[2, 2] eps dX[1]

                                                  2
     - eps dA[2, 2] rX[1] - eps  dA[2, 2] dX[1]]

                                                                                  2
[rA[3, 2] rX[3] + rA[3, 2] eps dX[3] + eps dA[3, 2] rX[3] + eps  dA[3, 2] dX[3]

     - rA[3, 3] rX[2] - rA[3, 3] eps dX[2] - eps dA[3, 3] rX[2]

          2
     - eps  dA[3, 3] dX[2], - rA[3, 1] rX[3] - rA[3, 1] eps dX[3]

                                  2
     - eps dA[3, 1] rX[3] - eps  dA[3, 1] dX[3] + rA[3, 3] rX[1]

                                                            2
     + rA[3, 3] eps dX[1] + eps dA[3, 3] rX[1] + eps  dA[3, 3] dX[1],

    rA[3, 1] rX[2] + rA[3, 1] eps dX[2] + eps dA[3, 1] rX[2]

          2
     + eps  dA[3, 1] dX[2] - rA[3, 2] rX[1] - rA[3, 2] eps dX[1]

                                  2
     - eps dA[3, 2] rX[1] - eps  dA[3, 2] dX[1]]
```

Dans cet exemple, la matrice duale AEE est définie à partir du produit des matrices duales A et EE. L'opérateur qui a permis une telle opération est la fonction `multiply` (fonction de base de Maple). Visiblement, nous pouvons remarquer qu'un tel opérateur a généré des termes en ε^2. Pour les éliminer, on peut appliquer la fonction `objdualsimp`.

```
> objdualsimp(AEE);
 [(rA[1, 2] dX[3] + dA[1, 2] rX[3] - rA[1, 3] dX[2] - dA[1, 3] rX[2]) eps

      + rA[1, 2] rX[3] - rA[1, 3] rX[2],

     (- rA[1, 1] dX[3] - dA[1, 1] rX[3] + rA[1, 3] dX[1] + dA[1, 3] rX[1]) eps

      - rA[1, 1] rX[3] + rA[1, 3] rX[1],

     (rA[1, 1] dX[2] + dA[1, 1] rX[2] - rA[1, 2] dX[1] - dA[1, 2] rX[1]) eps

      + rA[1, 1] rX[2] - rA[1, 2] rX[1]]

 [(rA[2, 2] dX[3] + dA[2, 2] rX[3] - rA[2, 3] dX[2] - dA[2, 3] rX[2]) eps

      + rA[2, 2] rX[3] - rA[2, 3] rX[2],

     (- rA[2, 1] dX[3] - dA[2, 1] rX[3] + rA[2, 3] dX[1] + dA[2, 3] rX[1]) eps

      - rA[2, 1] rX[3] + rA[2, 3] rX[1],

     (rA[2, 1] dX[2] + dA[2, 1] rX[2] - rA[2, 2] dX[1] - dA[2, 2] rX[1]) eps
```

```
     + rA[2, 1] rX[2] - rA[2, 2] rX[1]]

 [(rA[3, 2] dX[3] + dA[3, 2] rX[3] - rA[3, 3] dX[2] - dA[3, 3] rX[2]) eps

     + rA[3, 2] rX[3] - rA[3, 3] rX[2],

    (- rA[3, 1] dX[3] - dA[3, 1] rX[3] + rA[3, 3] dX[1] + dA[3, 3] rX[1]) eps

     - rA[3, 1] rX[3] + rA[3, 3] rX[1],

    (rA[3, 1] dX[2] + dA[3, 1] rX[2] - rA[3, 2] dX[1] - dA[3, 2] rX[1]) eps

     + rA[3, 1] rX[2] - rA[3, 2] rX[1]]
```

Maintenant, considérons le nombre dual a précédemment défini et le polynôme dual dupoly1 défini par:

```
> dupoly1:=a*(y^2*x^2 + 5*y^2*x + 7*y*x^2) + eps*(4*y*x + x^2 + 2*x):
```

Le simplificateur duwithopt va permettre de simplifier la forme de ce polynôme à deux variables x et y et définir ses parties réelle et duale avec un coût minimum en nombre d'opérations élémentaires:

```
> duwithopt(dupoly1);
    x ra y (7 x + (x + 5) y) + eps x (x + 2 + (4 + 7 da x + (x + 5) da y) y)
```

Le simplificateur objwithopt permet de faire la même opération sur des objet de type scalaire, vecteur ou matrice.

```
> dupoly2:=eps*(x^2+3*x+4):

> W:=array([dupoly1,dupoly2]):

> objwithopt(W);
  [ x ra y (7 x + (x + 5) y) + eps x (x + 2 + (4 + 7 da x + (x + 5) da y) y),
      eps (4 + (3 + x) x) ]
```

de telle fonctions peuvent s'avérer très intéressantes pour la préparation de la génération de codes optimisés (en Fortran par exemple).

7.4 Fonctions

Le module dualstruc.map offre non seulement la possibilité de définir des fonctions d'une (ou plusieurs) variable duale, mais aussi de bénéficier de tous les développements théoriques étudiés dans la partie mathématique de ce mémoire de thèse.

7.4.1 Fonctions parties réelle et duale

Nous avons implémenter les fonctions Re et Du calculant respectivement les parties réelle et duale. Elle peuvent être appliquées aux différents types d'objets.

Exemples

On conserve ici en mémoire les différentes quantités définies dans les exemples précédents:

- La partie duale d'un scalaire dual:

```
> l:=Du(a);
                                    l:= da
```

- La partie réelle de la matrice duale d'une matrice:

```
> Rd:=Re(Dep2);
                                  [ 0  -1  0 ]
                                  [          ]
                             Rd := [ 1   0  0 ]
                                  [          ]
                                  [ 0   0  1 ]
```

Elle représente la matrice de la partie linéaire du déplacement.

- La partie duale du produit de deux matrices duales:

```
>Du(multiply(A,Dep2)) ;
        [ - c rA[1, 1] + dA[1, 2]  - dA[1, 1] - c rA[1, 2]  dA[1, 3] ]
        [                                                            ]
        [ - c rA[2, 1] + dA[2, 2]  - dA[2, 1] - c rA[2, 2]  dA[2, 3] ]
        [                                                            ]
        [ - c rA[3, 1] + dA[3, 2]  - dA[3, 1] - c rA[3, 2]  dA[3, 3] ]
```

7.4.2 Prolongement canonique

Les fonctions essentielles, du module `dualstruc.map`, pour la génération des prolongements canoniques des fonctions de variable réelle sont `generalizedform` qui donne la forme générale d'un prolongement canonique à partir de la fonction de variable réelle qu'on souhaite prolonger et `appliqued` qui applique le prolongement canonique duale d'une fonction de variable réelle à un nombre dual.

Exemples

Les deux fonctions constructrices `generalizedform` et `appliqued` peuvent être appliquées à toute sortes de fonctions de variable réelle. Ainsi dans cette exemple on considère une fonction:

$$g := exp \circ arcsin \circ f \circ sin + log$$

où f est une fonction de variable réelle dont on ignore à priori la forme explicite. Posons:

```
> g:=exp@arcsin@f@sin+log:

> gtild:=generalizedform(g);
     gtild := (x1,x2) -> exp(arcsin(f(sin(x1)))) + ln(x1)

                 /cos(x1) D(f)(sin(x1)) exp(arcsin(f(sin(x1))))     1 \
          + eps x2 |------------------------------------------- + ----|
                 |                              2 1/2               x1 |
                 \             (1 - f(sin(x1)) )                     /
```

Le module `dualstruc.map` offre alors deux possibilités pour appliquer `gtild` à un nombre dual:

- grâce à la fonction `appliqued`:

```
> e:=Pi/4+eps*c :
> appliqued(g,e);
                      1/2
     exp(arcsin(f(1/2 2   ))) + ln(1/4 Pi)

                /     1/2          1/2                    1/2          \
                |    2    D(f)(1/2 2   ) exp(arcsin(f(1/2 2   )))     4 |
          + eps c |1/2 ------------------------------------------ + ----|
                |                              1/2 2 1/2              Pi |
                \                (1 - f(1/2 2   ) )                     /
```

- à partir de la fonction `gtild` elle même:

```
> gtild(Re(e),Du(e));
                      1/2
     exp(arcsin(f(1/2 2   ))) + ln(1/4 Pi)

                /     1/2          1/2                    1/2          \
                |    2    D(f)(1/2 2   ) exp(arcsin(f(1/2 2   )))     4 |
          + eps c |1/2 ------------------------------------------ + ----|
                |                              1/2 2 1/2              Pi |
                \                (1 - f(1/2 2   ) )                     /
```

La solution dans les deux cas est absolument la même.

7.5 Fonctions spéciales

Dans cette rubrique sont classées les constructeurs de ce que nous avons convenu d'appeler dans la partie théorique angles d'Euler duaux et angles de Briant duaux.

7.5.1 Angles d'Euler duaux associés à un déplacement

Les précession, nutation et rotation propres duales pour un déplacement -quelconque, mais, sans singularité- sont données respectivement par les fonctions constructrices `make_phi_eul`, `make_theta_eul` et `make_psi_eul`.

Exemple

En supposant que la matrice duale `A` est la matrice associée à un déplacement donné relativement à une famille fondamentale quelconque:

```
> make_theta_eul(A);
                  2           2 1/2
         (rA[3, 1]  + rA[3, 2] )                    2
  arctan(------------------------) - eps (rA[3, 1]  dA[3, 3]
                 rA[3, 3]

                 2
       + rA[3, 2]  dA[3, 3] - dA[3, 1] rA[3, 1] rA[3, 3]

       - dA[3, 2] rA[3, 2] rA[3, 3])
                                                 /            2           2\
         /           2           2 1/2         2 |    rA[3, 1]  + rA[3, 2] |
        /  ((rA[3, 1]  + rA[3, 2] )    rA[3, 3]  |1 + ---------------------|)
       /                                         |                   2     |
                                                 \          rA[3, 3]       /

> make_psi_eul(A);
              rA[2, 3]    eps (- rA[2, 3] dA[1, 3] + dA[2, 3] rA[1, 3])
        arctan(--------) + ---------------------------------------------
              rA[1, 3]                          2           2
                                       rA[1, 3]  + rA[2, 3]

> make_phi_eul(A);
               rA[3, 2]    eps (- rA[3, 2] dA[3, 1] + dA[3, 2] rA[3, 1])
       - arctan(--------) - ---------------------------------------------
               rA[3, 1]                          2           2
                                        rA[3, 1]  + rA[3, 2]
```

7.5.2 Angles de Briant duaux associés à un déplacement

Les roulis, tangage et lacet duaux pour un déplacement -quelconque, mais, sans singularité- sont données respectivement par les fonctions constructrices `make_phi_bri`, `make_theta_bri` et `make_psi_bri`.

Exemple

```
> make_theta_bri(A);
                  rA[3, 1]
  - arctan(------------------------) - eps (- rA[3, 1] dA[1, 1] rA[1, 1]
                  2           2 1/2
          (rA[1, 1]  + rA[2, 1] )

                                                          2                   2
       - rA[3, 1] dA[2, 1] rA[2, 1] + dA[3, 1] rA[1, 1]  + dA[3, 1] rA[2, 1] )

                                      /                 2      \
         /           2           2 3/2 |          rA[3, 1]       |
        /  ((rA[1, 1]  + rA[2, 1] )    |1 + --------------------|)
       /                               |           2           2|
                                       \    rA[1, 1]  + rA[2, 1] /

> make_psi_bri(A);
              rA[2, 1]    eps (- rA[2, 1] dA[1, 1] + dA[2, 1] rA[1, 1])
        arctan(--------) + ---------------------------------------------
              rA[1, 1]                          2           2
```

```
                                        rA[1, 1]  + rA[2, 1]

> make_phi_bri(A);
                rA[3, 2]    eps (rA[3, 2] dA[3, 3] - dA[3, 2] rA[3, 3])
         arctan(--------) - ------------------------------------------
                rA[3, 3]                          2          2
                                        rA[3, 3]  + rA[3, 2]
```

Il est, bien entendu, clair que l'expression formelle de la matrice `A` peut être aussi complexe qu'on puisse l'imaginer (provenant de la multiplication des matrices de plusieurs déplacements).

Exemple

A titre d'exemple, le tangage dual d'un déplacement obtenu du produit des matrices duales des trois déplacements `RD3(Pi/4+eps)`, `A` et `RD3(Pi/3+eps*c)` est donné par:

```
> DD1:=matdualsimp(multiply(A,RD3(Pi/3+eps*c))):

> DD:=multiply(RD3(Pi/4+eps),DD1):

> make_theta_bri(DD);
                                                 1/2              1/2          2
- arctan(1/8 (2 rA[3, 1] rA[2, 1] rA[2, 2] 3    + rA[3, 2] 3    rA[1, 1]

                                      1/2
    + 2 rA[3, 1] rA[1, 1] rA[1, 2] 3    + 6 rA[3, 2] rA[2, 1] rA[2, 2]

                                                          2
    + 6 rA[3, 2] rA[1, 2] rA[1, 1] + rA[3, 1] rA[2, 1]

                 1/2          2                        2                      2
    + 3 rA[3, 2] 3    rA[1, 2]  + 3 rA[3, 1] rA[1, 2]  + rA[3, 1] rA[1, 1]

                         2              1/2          2
    + 3 rA[3, 1] rA[2, 2]  + rA[3, 2] 3    rA[2, 1]

                 1/2          2    /              2
    + 3 rA[3, 2] 3    rA[2, 2] )  /  (3/4 rA[2, 2]
                                 /

                              1/2                            1/2                2
    + 1/2 rA[2, 1] rA[2, 2] 3    + 1/2 rA[1, 1] rA[1, 2] 3    + 3/4 rA[1, 2]

                 2               2                                        2
    + 1/4 rA[1, 1]  + 1/4 rA[2, 1] )^3/2) - 1/8 eps (4 rA[3, 2] c rA[2, 1]

                 1/2                                    1/2
    - 3 rA[3, 2] 3    dA[1, 2] rA[1, 2] - rA[3, 2] 3    dA[2, 1] rA[2, 1]

               1/2                                      1/2          2
    - rA[3, 2] 3    dA[1, 1] rA[1, 1] - 4 rA[3, 1] c 3    rA[2, 2]

                   1/2          2                                      1/2
    - 4 rA[3, 1] c 3    rA[1, 2]  + 2 dA[3, 1] rA[1, 1] rA[1, 2] 3
```

```
                                      1/2                                    1/2
  + 2 dA[3, 1] rA[2, 1] rA[2, 2] 3    + 4 rA[3, 2] c rA[2, 1] rA[2, 2] 3

                                    1/2             1/2         2
  + 4 rA[3, 2] c rA[1, 1] rA[1, 2] 3    + dA[3, 2] 3    rA[2, 1]

                1/2         2               1/2         2
  + 3 dA[3, 2] 3    rA[2, 2]  + 3 dA[3, 2] 3    rA[1, 2]

  + 6 dA[3, 2] rA[2, 1] rA[2, 2] + 6 dA[3, 2] rA[1, 1] rA[1, 2]

  - 3 rA[3, 2] dA[2, 1] rA[2, 2] - 3 rA[3, 2] dA[2, 2] rA[2, 1]

  - 3 rA[3, 2] dA[1, 1] rA[1, 2] - 3 rA[3, 2] dA[1, 2] rA[1, 1]

                                  1/2                       1/2
  - rA[3, 1] dA[1, 1] rA[1, 2] 3    - rA[3, 1] dA[2, 2] 3    rA[2, 1]

                                  1/2                       1/2
  - rA[3, 1] dA[2, 1] rA[2, 2] 3    - rA[3, 1] dA[1, 2] 3    rA[1, 1]

                1/2                                   1/2         2
  - 3 rA[3, 2] 3    dA[2, 2] rA[2, 2] + dA[3, 2] 3    rA[1, 1]

                     2                       2
  + dA[3, 1] rA[2, 1]  + 3 dA[3, 1] rA[1, 2]  - 3 rA[3, 1] dA[1, 2] rA[1, 2]

  - rA[3, 1] dA[2, 1] rA[2, 1] - 3 rA[3, 1] dA[2, 2] rA[2, 2]

  - 4 rA[3, 1] rA[1, 2] c rA[1, 1] - 4 rA[3, 1] rA[2, 1] c rA[2, 2]

                       2                       2                    2
  + 3 dA[3, 1] rA[2, 2]  + 4 rA[3, 2] c rA[1, 1]  + dA[3, 1] rA[1, 1]

                                  /               2
  - rA[3, 1] dA[1, 1] rA[1, 1]) /  ((3/4 rA[2, 2]
                               /

                            1/2                           1/2                2
  + 1/2 rA[2, 1] rA[2, 2] 3    + 1/2 rA[1, 1] rA[1, 2] 3    + 3/4 rA[1, 2]

                2                2
  + 1/4 rA[1, 1]  + 1/4 rA[2, 1] )^3/2 (1 + 1/64 (

                                  1/2             1/2         2
  2 rA[3, 1] rA[2, 1] rA[2, 2] 3    + rA[3, 2] 3    rA[1, 1]

                                    1/2
  + 2 rA[3, 1] rA[1, 1] rA[1, 2] 3    + 6 rA[3, 2] rA[2, 1] rA[2, 2]

                                                        2
  + 6 rA[3, 2] rA[1, 2] rA[1, 1] + rA[3, 1] rA[2, 1]

                1/2         2                       2                    2
  + 3 rA[3, 2] 3    rA[1, 2]  + 3 rA[3, 1] rA[1, 2]  + rA[3, 1] rA[1, 1]
```

```
                        2              1/2          2
+ 3 rA[3, 1] rA[2, 2]  + rA[3, 2] 3    rA[2, 1]

              1/2          2      /              2
+ 3 rA[3, 2] 3    rA[2, 2] )^2  /  (3/4 rA[2, 2]
                               /

                           1/2                              1/2                2
+ 1/2 rA[2, 1] rA[2, 2] 3    + 1/2 rA[1, 1] rA[1, 2] 3    + 3/4 rA[1, 2]

              2               2
+ 1/4 rA[1, 1]  + 1/4 rA[2, 1] )^3))
```

7.6 Conclusion

Le module `dualstruc.map` définit de puissants outils informatiques de calcul et de manipulation formelle des objets duaux pour le mathématicien. D'autant plus, certains de ces outils sont spécialisés en géométrie et s'appliquent à des problèmes de mécanique. Le but de leur conception est de permettre d'écrire et de manipuler des expressions mathématiques formelles à partir d'objets duaux. L'ensemble de ces outils permet d'exprimer dans une syntaxe simple des modèles de réalités matérielles et physiques très complexes. Ils sont le fruit d'une étude théorique très approfondie des outils mathématiques qu'ils simulent. Ils apportent ainsi au mathématicien, pour ces besoins de modélisation, toutes les performances de rapidité et de précision du calcul sur la machine. Pour l'étudiant et l'utilisateur occasionnel, ils représentent une boîte à outils qui facilite grandement l'utilisation de l'ensemble des moyens de cette représentation mathématique. Le mécanisme d'héritage utilisé permet d'aller encore plus loin dans la programmation sous *MEDUSA MF77* et avec un moindre efforts de programmation.