Une structure de données EF “à la Castem” sous Matlab (pour le linéaire. . .)

David Dureisseix

version 1.6

1 Principales limitations

L’absence de pointeur dans Matlab conduit l’utilisateur à gérer lui-même les dépendances entre les objets.

L’absence de variables globales protégées en écriture contraint l’utilisateur à veiller à ne pas écraser les variables globales positionnées dans la routine GlobalVar qui est appelée en début de plusieurs fonctions Matlab :

```
GlobalVar;
```

2 Principales notions

Les *nœuds géométriques* (ou plus simplement, les nœuds) : ils sont données par un numéro, et leurs coordonnées (2 ou 3 si on est en 2D ou en 3D). L’ensemble des informations les concernant est donc donnée dans une “*pile de nœuds*”. Le principal problème concerne une renumérotation éventuelle : si on souhaite enlever des nœuds en trop dans la pile, on change la numérotation ; il faut donc updater tous les objets qui utilisent cette numérotation (en gros, tous les maillages). L’utilisateur doit donc gérer aussi une “pile de maillages” à updater lors d’un changement de numérotation de nœuds.

Les *éléments géométriques* se distinguent uniquement par les nœuds géométriques qu’ils utilisent ; par exemple : un triangle à 3 nœuds, un segment à 2 nœuds. . . Les *éléments finis* sont plus complexes. Ils s’appuient sur un élément géométrique, mais contiennent aussi une information sur les degrés de libertés, les fonctions de forme associées, les techniques d’intégration utilisées. Par exemple : un triangle à 6 nœuds P2 pour le déplacement, P1 continu d’un élément à l’autre pour la pression, dont la rigidité est intégrée par 4 points de Gauss, et la masse par une sous-intégration à 1 seul point de Gauss.

Un *maillage* est une collection d’éléments géométriques (identiques ou non). Il ne contient pas les coordonnées des nœuds mais uniquement la connectivité des éléments (il utilise donc la numérotation d’une pile de nœuds particulière). Si des éléments sont identiques, ils peuvent (mais pas obligatoirement) être regroupés par sous-zone. À part pour cette notion de groupement dans une sous-zone, il y a équivalence entre un

maillage et un *graphe de connectivité*. Tout maillage peut être transformé en graphe de connectivité. Un graphe de connectivité ne peut être transformé en maillage que sous réserve d'une hypothèse pour pouvoir reconstruire des sous-zones.

Un *modèle* est une collection d'éléments finis. Il contient donc un maillage (pour avoir les éléments géométriques), mais aussi les informations sur les degrés de libertés, les fonctions de forme associées, les techniques d'intégration utilisées.

3 Entrées - sorties

Le problème de l'échange de données n'est pas simple. Le format indépendant proposé est AVS. Trois problèmes de principe avec ce format : par défaut, on est toujours en 3D ; les champ par éléments sont toujours constants par élément (ceci permet de ne disposer que les éléments géométriques et non plus d'avoir besoin des éléments finis) ; enfin, lors de plusieurs enregistrements successifs dans le même fichier, on perd l'éventuelle cohérence entre eux (les mêmes nœuds par exemple, deviennent des nœuds différents). L'inconvénient dans chaque cas est la nécessité imposée à l'utilisateur de faire un choix avec ce qui arrive par un fichier AVS (faut-il repasser en 2D ? Faut-il fusionner des nœuds ?). Un dernier problème, de technologie celui-la : le fait d'utiliser un fichier formatté limite la précision des réels en virgule flottante.

Les routines proposée vont permettre de faire l'aller retour entre un fichier AVS et des objets complexes de type EF, mais aussi entre ces mêmes objets et leur représentation matricielle.

4 Utilisation matricielle

L'intérêt est de pouvoir utiliser la version matricielle des différents objets pour en faire ce qu'on veut avec Matlab. Il faut donc pouvoir passer des objets EF aux objets matriciels dans un sens comme dans l'autre.

Les *vecteurs* sont les représentations matricielles des champs. Un champ par point peut être transformé en vecteur avec une liste de numérotation de nœuds (typiquement un maillage d'éléments POI1 à 1 seul nœud par élément), et une hypothèse sur l'ordre de rangement des degrés de liberté. Avec ces mêmes informations, plus les noms des composantes du champ, la transformation inverse est possible. L'hypothèse typique est de ranger les composantes dans l'ordre pour chaque nœud, eux même dans l'ordre du maillage POI1.

Les vecteurs peuvent aussi être les représentations des champs par éléments, de façon plus complexe. Cette transformation est associée à l'opération qui permet de passer d'un champ par élément au champ par point discret de forces généralisées associées. La définition matricielle de cet opérateur BSIGMA contient la définition de ces vecteurs. En matriciel, on a S le vecteur associé au champ de contrainte σ, U le vecteur déplacement associé au champ de déplacement u, F le vecteur des efforts généralisés qui équilibre S : $F = B^T S$, $F \cdot U = \int \mathrm{Tr}\sigma\varepsilon(u) = S \cdot BU$, $E = BU$. Le vecteur directement associé au champ de déformation $\varepsilon(u)$ est par contre $\Sigma = bU$.

5 Objets complexes

5.1 La sous-zone

Une sous-zone peut être définie pour un maillage comme une suite d'éléments géométriques de même type.

Une sous-zone peut être définie pour un champ par point comme un groupe de nœuds sur chacun desquels sont définies les valeurs des mêmes composantes.

Une sous-zone peut être définie pour un champ par élément (ou un modèle) comme une suite d'éléments finis ayant tous le même élément de référence (même type géométrique d'élément et même intégration) *et* sur chacun desquels sont définies les valeurs des mêmes composantes.

5.2 Le maillage

Le maillage est constitué d'une suite de sous-zones. Chaque sous-zone est un maillage élémentaire.

```
mail1 : maillage (array of cells de taille
                  nbzo1 = nombre de sous-zones)
  mail1{zo1} = maillage élémentaire numéro zo1
```

Un maillage élémentaire est quant à lui défini de la façon suivante :

```
maile1 : maillage élémentaire (structure)
maile1.TYPE : type d'element géométrique (mot)
maile1.MAIL : connectivité (tableau d'entiers de taille (nbeli,nbnni)
                            nbeli = nombre d'éléments géométriques
                            nbnni = nombre de n\oe{}uds par élément)
  maile1.MAIL(eli,nni) = numéro du n\oe{}ud nni de l'élément eli
```

Pour définir entièrement un maillage, il faut aussi lui associer une pile de nœuds.

```
xcoor1 : pile de n\oe{}uds (tableau d'entiers de taille (nbno,idim)
                        nbno = nombre de n\oe{}uds
                        idim = nombre de coordonnées)
  xcoor1(no,i) = coordonnée i du n\oe{}ud no de cette pile (double)
```

Sauf si on a besoin que de la connectivité, un maillage est donc un couple (mail1, xcoor1).

5.3 Le modèle

Le modèle se base déjà sur un maillage géométrique. Sur chaque élément de référence de ce maillage (autant que de sous-zones, donc), doivent être en plus définis des informations sur les degrés de libertés, les fonctions de forme associées, les techniques d'intégration utilisées.

Pour l'intégration, il est nécessaire de pouvoir disposer d'un (voire de plusieurs) *segment d'intégration*. L'intégration fait appel à des points particuliers dits *points d'intégration* dont on doit connaître les coordonnées dans l'élément géométrique de référence, en lesquels doivent être définis les poids d'intégration et le Jacobien de la

transformation entre l'espace réel et l'espace de référence. Ce dernier dépendant de l'élément géométrique réel, il est plus facile de stocker pour les éléments isoparamétriques, la dérivée des fonctions de forme aux points d'intégration. On aura de plus besoin de la valeur des fonctions de forme en ces mêmes points.

```
intg1 : segment d'intégration (array of cells de taille
                               nbzo1 = nombre de sous-zones)
  intg1{zo1}.PHI : valeur des fonctions de forme
                   aux points d'integration
                   (tableau de doubles de taille (nbnni,nbptg)
                    nbnni = nombre de fonctions de forme
                    nbptg = nombre de points d'intégration)
    intg1{zo1}.PHI(nni,ptg) = valeur de la fonction de forme
                              numéro nni au point d'intégration ptg
  intg1{zo1}.DPHI : valeur des dérivées des fonctions de forme
                   aux points d'integration
                   (tableau de doubles de taille (idimr,nbnni,nbptg)
                    idimr = dimension dans l'espace de référence
                    nbnni = nombre de fonctions de forme
                    nbptg = nombre de points d'intégration)
    intg1{zo1}.DPHI(i,nni,ptg) = valeur de la dérivée par rapport
      à la coordonnée numéro i dans l'espace de référence,
      de la fonction de forme nni, au point d'intégration numéro ptg
  intg1{zo1}.COOR : coordonnees des points d'integration
                    dans l'espace de référence
                    (tableau de doubles de taille (nbptg,idimr)
                     nbptg = nombre de points d'intégration
                     idimr = dimension dans l'espace de référence)
    intg1{zo1}.COOR(ptg,ir) = coordonnée ir du point d'intégration ptg
  intg1{zo1}.WEIGHT : poids des points d'integration
                      (liste de taille nbptg
                       nbptg = nombre de points d'intégration)
    intg1{zo1}.WEIGHT(ptg) = valeur du poids du point d'intégration
                             numéro ptg
```

Bien entendu, les sous-zones d'un segment d'intégration doivent être cohérentes avec les sous-zones du maillage associé.

Pour les degrés de liberté et les fonctions de forme associées, chaque sous-zone peut avoir ses caractéristiques particulières. Il faut ainsi avoir la liste des noms de degrés de liberté attachés à chaque valeur, dans l'ordre suivant : 1er ddl du 1er nœud, 2e ddl du 1er nœud…1er ddl du 2e nœud… et en tout cas, pouvoir repérer à quel nœud appartient chaque degré de liberté (d'où les liste de nœuds, dans l'ordre des nœuds de l'élément de référence), et à quelle fonction de forme il est associé (une même fonction de forme peut être utilisée pour différents degrés de liberté). De plus, afin de pouvoir faire la transformation de l'élément de référence vers l'élément réel, pour les éléments non-isoparamétriques, on a besoin aussi de savoir quels sont les fonctions de forme utilisées et à quels nœuds elles se rapportent (une fonction par nœud, cette fois-ci).

```
modl1 : modèle (array of cells de taille
                nbzo1 = nombre de sous-zones)
   modl1{zo1}.DDLP : differents noms des ddl primaux
                     (array of cells de taille nbn)
   modl1{zo1}.DDLD : différents noms des ddl duaux
                     (array of cells de taille nbn)

   modl1{zo1}.NDDP : numéro des noms de ddl primaux
                     (liste de taille nbep)
   modl1{zo1}.NDDD : numéro des noms de ddl duaux
                     (liste de taille nbed)
   modl1{zo1}.NNOP : numéro des n\oe{}uds primaux associés
                     (liste de taille nbep)
   modl1{zo1}.NNOD : numéro des n\oe{}uds duaux associés
                     (liste de taille nbed)
   modl1{zo1}.NNIP : numéro des fonctions de forme associées
                     (liste de taille nbep1)
   modl1{zo1}.NNID : numéro des fonctions de forme associées
                     (liste de taille nbed1)

   modl1{zo1}.COMP : differents noms des composantes primales
                     (array of cells de taille nbc)
   modl1{zo1}.COMD : différents noms des composantes duales
                     (array of cells de taille nbc)
   modl1{zo1}.NCOP : numéro des composantes primales
                     (liste de taille nbcp)
   modl1{zo1}.NCOD : numéro des composantes duales
                     (liste de taille nbcd)


   modl1{zo1}.NNOT : numéros des n\oe{}uds utilisés pour la transformation
                     (liste de taille nbe)
   modl1{zo1}.NNIT : numéros des fonctions de forme associées (entier)
                     (liste de taille nbe)
```

Pour l'instant, la description de la fonction de forme par la fonction qui, aux coordonnées de référence d'un point, associe la valeur des fonctions de forme, n'est pas encore prévue…

Concernant les données relatives aux champs par éléments, il faut aussi préciser les noms des composantes de déformation (ou équivalent suivant la formulation), et de leur dual : les contraintes. De façon similaire aux degrés de liberté, on stocke donc les noms de ces composantes dans les champs COMP et COMD, les valeurs utiles sont repérées dans les numéros d'apparition des noms NCOP et NCOD. Une remarque importante est la suivante : a priori, tous les points d'intégration sont supports des mêmes composantes, les points d'intégration considérés ne sont donc pas repérés, puisqu'ils sont tous similaires. C'est un choix ; on aurait par exemple pu ajouter des champs NPTP

et NPTD qui listeraient les points d'intégration, comme on a pu lister les nœuds dans NDDP et NDDD. Il n'y a pas non plus de fonction de forme associée, puisque ces champs ne sont pas interpolés. Les interpoler reviendrait à les considérer comme des nœuds.

Dans le cas le plus général, un modèle est donc un triplet (modl1, mail1, intg1).

Il existe un cas plus particulier : celui des éléments de type structure (poutre, coque...) qui possèdent une orientation particulière, donc une base locale. Les quantités interpolées sont les composantes dans la base locale. Sur un tel élément, la procédure est la suivante : à partir des degrés de libertés aux nœuds, on peut passer dans la base locale pour avoir les degrés de libertés locaux aux nœuds (moyennant les valeurs des dérivées des fonctions de forme de la transformation aux nœuds). Ce sont ces degrés de liberté qui sont interpolés, et qui permettent de construire les opérateurs tels que la rigidité. Enfin, la matrice élémentaire est "tournée" pour être remise dans la base globale. On aura donc les champs supplémentaires suivants :

```
modl1{zo1}.DDLPLOC : differents noms des ddl primaux locaux
                     (array of cells de taille nbnloc)
modl1{zo1}.DDLDLOC : différents noms des ddl duaux locaux
                     (array of cells de taille nbnloc)
modl1{zo1}.NDDPLOC : numéro des noms de ddl primaux
                     (liste de taille nbeploc)
modl1{zo1}.NDDDLOC : numéro des noms de ddl duaux
                     (liste de taille nbedloc)
modl1{zo1}.NNOPLOC : numéro des n\oe{}uds primaux associés
                     (liste de taille nbeploc)
modl1{zo1}.NNODLOC : numéro des n\oe{}uds duaux associés
                     (liste de taille nbedloc)
```

À ce moment-là, nbep1 = nbeploc, et nbed1 = nbedloc, alors qu'ils étaient précédemment égaux respectiuvment à nbep et nbed. Les noms de composantes sont par contre toujours les noms locaux.

5.4 La rigidité

Il s'agit ici des matrices de rigidité sous forme élémentaire. La matrice assemblée sera, elle, un objet matriciel.

Toutes les matrices élémentaires d'une même sous-zone ont la même taille. Les ddl sont rangés dans l'ordre défini par le modèle modl1, les nœuds support sont ceux du maillage mail1.

```
rigi1 : rigidités élémentaires (array of cells de taille
                                nbzo1 = nombre de sous-zones)
 rigi1{zo1}.XVAL : valeurs des termes des matrices élémentaires
                   (tableau de doubles de taille (nbed,nbep,nbel)
                     nbed = nombre de ddl duaux
                     nbep = nombre de ddl primaux
                     nbel = nombre d'éléments)
   rigi1{zo1}.XVAL(i,j,el) : terme (i,j) de la matrice élémentaire
```

```
                                de l'élément el de la zone zo1
```

La description complète des rigidités élémentaires est donc le triplet (rigi1, modl1, mail1).

5.5 La masse

Il ne s'agit que d'une rigidité particulière. L'intégration est cependant différente (points d'intégration pour la masse plus nombreux que pour la rigidité ; attention, la masse est souvent sous-intégrée par défaut).

5.6 Les champs

5.6.1 Le champ par point

Défini sur un certain nombre de nœuds, il en est de deux types : le champ par point diffus (interpolable par des éléments finis *via* leurs fonctions de forme) et le champ par point discret (valeurs aux nœuds). Les deux sont mis en dualité par l'énergie (produit scalaire entre un champ par point diffus, par exemple : un champ de déplacement, et un champ par point discret, par exemple : des efforts généralisés).

Un champ par point contient donc une liste de nœuds (typiquement un maillage d'éléments POI1 à 1 seul nœud par élément), ainsi que des valeurs de composantes. Problème : les composantes peuvent être différentes, et porter sur tout ou partie des nœuds ; c'est pourquoi la notion de sous-zone existe aussi pour les champs par point. Pour être plus flexible, une composante peut être à une seule valeur (exemple : la pression, la composante UX du champ de déplacement) ou à plusieurs valeurs (exemple : le champ de déplacement).

```
chpo1 : champ par point (array of cells de taille
                         nbzo1 = nombre de sous-zones)
  chpo1{zo1}{i}.COMP = nom de la composante numero i (mot)
  chpo1{zo1}{i}.UNIT = nom de l'unite de la composante numero i (mot)
  chpo1{zo1}{i}.XVAL : valeurs du champ
    (tableau de doubles de taille (nbno,nbval)
     nbno = nombre de n\oe{}uds dans la sous-zone
     nbval = nombre de valeurs de cette composante)
    chpo1{zo1}{i}.XVAL(no,val) = valeur numéro val
                                 au n\oe{}ud no (double)
```

La numérotation des nœuds est celle d'un maillage POI1 nmail1 qui doit être associé au champ par point pour qu'il soit complet. Ce maillage POI1 doit avoir des sous-zones (toutes POI1) cohérentes avec celles du champ par point. On peut donc dire que le champ par point est le couple (chpo1, nmail1). Si on pouvait utiliser des pointeurs, il suffirait que la structure chpo1 ait un pointeur vers la structure nmail1, mais ce n'est pas le cas…

L'ordre global des composantes est déterminé à partir du numéro de la sous-zone et de celui de la composante (zo1, i). L'hypothèse la plus simple est de numéroter les composantes de façon croissante en incrémentant le numéro de la sous-zone.

5.6.2 Le champ par élément

Il est intimement lié à la notion d'intégration.

Un champ par élément peut être éventuellement interpolable quand il s'agit d'un champ par élément définit aux nœuds d'un élément fini.

Cas particulier : le champ par élément constant par élément. Dans ce cas, les valeurs sont données élément par élément (on pourrait dire qu'elles sont données au centre de gravité des éléments géométriques) ; il suffit alors d'avoir une numérotation des éléments pour savoir à qui attribuer la (les) valeur(s). Un champ par élément constant peut donc être associé à un simple graphe de connectivité (c'est d'ailleurs ce que fait AVS) ; cependant, pour rester cohérent, il est préférable de l'associer à un maillage, le numéro d'élément global au maillage utilisé est construit à partir des sous-zones avec une hypothèse déjà mentionnée. L'hypothèse la plus simple est de numéroter les éléments de façon croissante en incrémentant le numéro de la sous-zone.

```
chml1 : champ par élément constant par élément
       (array of cells de taille nbcomp1
        nbcomp1 = nombre de composantes)
  chml1{i}.COMP = nom de la composante numero i (mot)
  chml1{i}.UNIT = nom de l'unite de la composante numero i (mot)
  chml1{i}.XVAL : valeurs du champ
    (tableau de doubles de taille (nbelt,nbval)
     nbelt = nombre total d'éléments
     nbval = nombre de valeurs de cette composante)
    chml1{i}.XVAL(elt,val) = valeur numéro val
                             de l'élément elt (double)
```

La numérotation des éléments est celle d'un maillage mail1 qui doit être associé au champ par élément pour qu'il soit complet. On peut donc dire que le champ constant par élément est le couple (chml1, mail1). Si on pouvait utiliser des pointeurs, il suffirait que la structure chml1 ait un pointeur vers la structure nmail1, mais ce n'est pas le cas...

Dans tous les cas, un champ par élément contient un modèle, ainsi que des valeurs de composantes. Problème : les composantes peuvent être différentes, et porter sur tout ou partie du modèle. La solution envisagée est la même que pour le champ par point ; c'est pourquoi la notion de sous-zone existe aussi pour les champs par éléments. Pour être plus flexible, une composante peut être à une seule valeur (exemple : la pression, la composante EPXX du champ de déformation) ou à plusieurs valeurs (exemple : le champ de contraintes).

Dans le cas plus général d'un champ par éléments (variable par élément), on va trouver un numéro de sous-zone (zo) et un numéro d'élément (el) dans la sous-zone. On peut remarquer que l'ensemble des deux, (zo, el), correspond à une numérotation globale des éléments (elt), comme on l'a vu pour les champs constants par élément. Le problème est cette fois-ci qu'il y a aussi plusieurs composantes possible, donc un numéro de composante (i). le champ peut être rangé d'abord par composante, puis par élément ou le contraire. La solution choisie est de classer par sous-zone, puis par composante, puis élément de la sous-zone, par similarité avec Cast3M.

```
cham1 : champ par élément
       (array of cells de taille nbzo1
        nbzo1 = nombre de sous-zones)
  cham1{zo1}{i}.COMP = dans la sous-zone zo1,
                       nom de la composante numero i (mot)
  cham1{zo1}{i}.UNIT = dans la sous-zone zo1,
                       nom de l'unite de la composante numero i (mot)
  cham1{zo1}{i}.XVAL : dans la sous-zone zo1,
                       valeurs du champ
                       (tableau de doubles de taille (nbel,nbvalt)
                        nbel = nombre d'éléments dans la sous-zone
                        nbvalt = nombre de valeurs total
                                 de cette composante)
    cham1{zo1}{i}.XVAL(el,valt) = valeur numéro valt de l'élément el
                                  de la sous-zone zo1 (double)
```

Il faut remarquer que le nombre de valeurs total pour un élément, une composante, une zone est le nombre de valeurs effectives dans la composantes multiplié par le nombre de points d'intégration. L'hypothèse de rangement est de la aire dans l'ordre des points d'intégration croissant (ptg, val). Le cas le plus simple est celui où il n'y a qu'une valeur par composante, et donc où le nombre total de valeurs est le nombre de points d'intégration. Pour être complet, le champ cham1 doit être associé à un modèle modl1, lui-même associé à un maillage géométrique mail1. On peut donc dire que le champ par élément est le couple (cham1, modl1, mail1). Si on pouvait utiliser des pointeurs, il suffirait que la structure cham1 ait un pointeur vers la structure modl1, mais ce n'est pas le cas...

Exemple de champ par élément : le matériau Il est soit constant par élément, soit défini aux points d'intégration de la rigidité (pour construire la matrice de rigidité) et aux points d'intégration de la contrainte (pour construire le champ par éléments de la contrainte, qui doit lui-même pouvoir être utilisé pour construire les efforts généralisés qui l'équilibrent). Une solution : confondre ces deux techniques d'intégration ; les points d'intégration de la rigidité et de la contraintes sont identiques !

Voici par exemple une façon de construire "manuellement" un champ de masse volumique RHO, de module d'Young YOUN et de coefficient de Poisson NU constant par élément à partir des valeurs constantes partout (nbelt1 est le nombre total d'éléments) :

```
clear mater0;
mater0{1} = struct('COMP','YOUN','UNIT','','XVAL',1.*ones(nbelt1,1));
mater0{2} = struct('COMP','RHO' ,'UNIT','','XVAL',100.*ones(nbelt1,1));
mater0{3} = struct('COMP','NU'  ,'UNIT','','XVAL',0.3*ones(nbelt1,1));
```

ou, ce qui est équivalent, d'appeler la routine ManuChml qui prend un nombre d'arguments variable, et construit un champ constant :

```
mater0 = ManuChml(mail1,'YOUN','',1.,'RHO','',100.,'NU','',0.3);
```

6 Routines et exemples d'utilisation

Un exemple typique est le suivant : récupération de données EF à partir d'un ou de plusieurs fichiers AVS, passage en matriciel, opération typiques de Matlab sur les matrices, retour au format EF et sauvegarde du résultat dans un ou des fichiers AVS. Le plus délicat est la nécessité pour l'utilisateur de maintenir la cohérence entre objets nécessaire par le manque de pointeurs dans Matlab.

6.1 Lecture de données à partir d'un fichier AVS

La lecture va se faire de la façon suivante : après ouverture du fichier (laissé à la charge de l'utilisateur, pour pouvoir faire des lectures successives dans un même fichier), le contenu est placé dans une structure de donnée de type EF.

```
fid = fopen('fichier.avs','rt');
  [xcoor1,mail1,chpo1,chml1,cara1,error1] = Read1AVS5(fid);
fclose(fid);

xcoor1 = xcoor1(:,1:2);

nbno1  = size(xcoor1,1);
clear nmail1;
  nmail1{1} = struct('MAIL',[1:nbno1]','TYPE','POI1');
```

xcoor1 est la pile des nœuds (à 3 coordonnées par défaut dans AVS : l'utilisateur est prié de se débarrasser lui-même de la troisième coordonnée s'il est en 2D).

mail1 est un maillage (il est construit à partir de l'information de type "graphe de connectivité" et des noms d'éléments propres à AVS, en rangeant au fur et à mesure les éléments, nommés *cells* dans AVS, dans des sous-zones). Les noms des éléments utilisés sont ceux de Cast3M.

chpo1 est un champ par point. Par définition dans AVS, il est défini sur tous les nœuds de la pile de nœuds xcoor1, à charge à l'utilisateur de construire le maillage POI1 associé nmail1 pour compléter la définition de chpo1.

chml1 est un champ constant par element. Par définition dans AVS, il est défini sur tous les éléments de mail1. La routine respecte la cohérence des données pour que le champ soit complètement défini par le couple (chml1, mail1).

cara1 est un champ constant par element de caractéristiques (dénomination de *model* par AVS). Là encore, le champ est complètement défini par (cara1, mail1).

error1 est un éventuel message d'erreur en cas de problème de lecture.

Dans le cas où un de ces objects ne serait pas contenu dans le fichier AVS, la routine retourne une liste vide : [].

Il est tout à fait possible de lire une série de données dans un fichier AVS :

```
fid = fopen('fichier.avs','rt');
  [xcoor1,mail1,chpo1,chml1,cara1,error1] = Read1AVS5(fid);
  [xcoor2,mail2,chpo2,chml2,cara2,error2] = Read1AVS5(fid);
fclose(fid);
```

Il faut cependant bien avoir en tête que chaque lecture a sa propre structure de données, indépendante de la première ; par exemple, la numérotation des nœuds dans mail2 est locale à la pile de nœuds xcoor2. Si on veut reconstruire une unique base de donnée, il va falloir fusionner les deux. Ceci pose un problème uniquement au niveau des nœuds, les éléments étant toujours numérotés localement dans les maillages. Deux situations peuvent exister : les nœuds sont indépendants et il suffit de les mettre bout a bout dans une unique grande pile de nœuds, ou les nœuds situés au même endroit (mêmes coordonnées) sont en fait les mêmes nœuds.

Dans la première situation, il suffit de fusionner les deux piles, et d'updater les numéros de nœuds dans le deuxième maillage, et éventuellement dans les maillages POI1 construits pour sous-tendre les champs par point :

```
xcoor3 = [xcoor1 ; xcoor2];
  decal1 = size(xcoor1,1);
  mail2  = ShiftNodeNumber(mail2,decal1);
  nmail2 = ShiftNodeNumber(nmail2,decal1);
clear xcoor1 xcoor2;
```

Il est recommandé de détruire les anciennes piles car elles n'ont plus de cohérence avec rien. De façon générale, il doit être plus simple pour l'utilisateur de ne conserver qu'une seule base de donnée, donc qu'une seule pile de nœuds.

La deuxième situation est traitée déjà comme la première par fusion des bases de données, puis par substitution des nœuds situés au même endroit. Il est ensuite possible ou non d'éliminer les nœuds inutiles de la pile de nœuds xcoor3. On retombe sur le problème de la renumérotation des nœuds avec la "pile de maillages" déjà évoquée.

```
clear pile_mail;
  pile_mail{1} = mail1;
  pile_mail{2} = mail2;
  pile_mail{3} = mail3;
  xcrit1 = 1.e-6;
  [pile_mail,ListUsedNodes] = ElimNode(pile_mail,xcoor3,xcrit1);
  mail1 = pile_mail{1};
  mail2 = pile_mail{2};
  mail3 = pile_mail{3};
clear pile_mail;
```

Il est nécessaire d'avoir un critère de proximité xcrit1 pour éliminer des nœuds proches. Attention, il faut que *tous* les maillages de la base de donnée soient dans la pile, sinon des incohérences vont apparaître. La routine retourne aussi la liste des nœuds effectivement utilisés si on souhaite ensuite éliminer les autres.

Ayant fusionné des bases de données, on peut vouloir ne sortir qu'une partie de la base de données finale dans un fichier AVS. Pour cela, l'extraction d'une partie de la base est faire par la fonction PrepareAVS :

```
[xcoor1a,mail1a,nmail1a,chpo1a] = PrepareAVS(xcoor3,mail1,nmail1,chpo1);
```

Cette fonction commence par choisir les seuls nœuds utiles de xcoor3 (ceux qui sont mail1 ou nmail1), et complète si besoin en 3D pour avoir xcoor1a. mail1 et nmail1 sont alors renumérotés de façon cohérente avec xcoor1a. Ensuite, (chpo1, nmail1) est

change en (chpo1a, nmail1a) qui ne contient plus qu'une sous-zone car il est sous-tendu par tous les nœuds de xcoor1a, les composantes ayant été éventuellement complétées par des valeurs nulles sur les nœuds où elles n'étaient pas définies.

Le résultat est alors directement utilisable par la routine d'écriture au format AVS (voir plus loin).

6.2 Objets matriciels

Pour passer des objets de type EF aux objets matriciels, il faut déterminer l'ordre dans lequel les valeurs vont être rangées. On peut soit ranger les valeurs par degré de liberté, soit par nœud (pour chaque nœud, les degrés de liberté étant attachés dans un certain ordre). Bien que moins générale, c'est cette deuxième solution qui est envisagée ici. On a donc besoin d'une liste de nœuds qui vont donner un ordre de rangement. Celle-ci pourrait être un maillage POI1, mais comme il s'agit ici d'avoir des valeurs matricielles, on va utiliser une liste d'entiers. Ces numéros de nœuds sont bien sûr liés à une pile de nœuds particulière pour pouvoir s'y retrouver. Cette liste permet aussi de trouver une numérotation adéquate pour stocker les matrices creuses, par exemple. Ensuite, pour repérer les degrés de liberté, il faut garder en tête que les noms des ddl ne sont pas forcément les même pour tous les nœuds, que les nœuds n'ont pas forcément tous les même ddl. Pour s'en sortir, on va avoir besoin de la liste de tous les noms de ddl possibles (les *composantes*) dans le champ : listComp1.

6.2.1 Cas des vecteurs et des champs par point

Prenons le cas d'un champ par point (chpo1, nmail1). On cherche à assembler ce champ par point dans un vecteur de degrés de liberté U1. L'ordonnancement des nœuds est donné dans une liste des nœuds numer1 (de taille nbnot1), l'ordonnancement des ddl par nœud est faite à partir d'une hypothèse qui peut être celle de l'apparition dans les composantes de chpo1.

Pour réaliser cet assemblage, il faut déjà connaître tous les noms de degrés de liberté (ou *composantes*) possibles ; ces noms sont rangés dans la liste ListComp1, de taille nbddl1 (pour compléter, la liste des unités est dans listUnit1). Cette liste donne l'ordre d'apparition des ddl.

Ensuite, il faut construire une matrice de "mapping" mapddl1 pour réaliser l'assemblage, de taille (nbnot1, nbddl1). Typiquement mapddl1(not1,ddl1) est le numéro d'apparition du ddl correspondant.

```
[listComp1,listUnit1] = ListCompChpo2(chpo1);
mapddl1 = MapddlChpo(chpo1,nmail1,numer1,ListComp1);
U1 = ChpoToVect3(chpo1,nmail1,numer1,mapddl1,listComp1);
```

Si des nœuds de nmail1 ne sont pas dans numer1, il s'agit en plus d'une restriction ; si des nœuds de numer1 ne sont pas dans nmail1, la valeur mise en place est nulle, il s'agit d'une extrapolation. L'utilisateur peut bien sur fournir sa propre liste listComp1 (pour ne pas assembler toutes les composantes, ou pour fournir sa propre numérotation des composantes), ou même son propre "mapping" mapddl1.

La transformation inverse (vecteur U1 en champ par point (chpo1, nmail1)) se fait de la manière suivante. Pour l'instant, on suppose qu'on va extrapoler le vecteur de

façon à construire un champ par point “full”, c’est à dire à une seule sous-zone qui contient toutes les composantes de listComp1 sur tous les nœuds de numer1 (ce format est celui qu’attendra le fichier AVS de sortie). Les valeurs qui n’existent pas dans U1 seront extrapolées à 0. listComp1 et numer1 devront bien sur être compatible avec un “mapping” mapddl1.

```
[chpo1,nmail1] = VectToChpo2(U1,numer1,mapddl1,ListComp1,ListUnit1);
```

Une routine plus évoluée permettrait de ne prendre qu’une partie des nœuds de numer1, avec la donnée de nmail1, ou en prendre plus. De même, elle pourrait ne pas construire un champ “full” mais avec le minimum de valeurs en distinguant plusieurs sous-zones avec le nombre minimal de nœuds...

Ici, la restriction ou l’extrapolation doit se faire en modifiant “à la main” les numer1, mapddl1, ListComp1, ListUnit1 en gardant leur cohérence.

6.2.2 Construction des rigidités ou masses

Pour construire les rigidités élémentaires, on va avoir besoin du modèle (modl1, mail1, intg1) et des coordonnées des nœuds xcoor1, ainsi que du champ par élément de matériau (matr1, mail1). Le modèle lui-même est construit à partir du maillage géométrique mail1. Le matériau doit contenir les composantes voulues pour calculer la rigidité. Dans l’exemple suivant, il est construit à partir d’un champ constant par élément mater0 :

```
[modl1,intg1] = ModlIntg4(mail1,'ELASTIQUE','RIGIDITE',mode1);
mater1 = ChmlToCham(mater0,mail1,intg1);
[rigi1,bsig1,b1] = Rigi5(modl1,matr1,mail1,intg1,xcoor1,mode1);
```

(on verra plus loin ce que sont bsig1 et b1).

Dans le cas des matrices de masse élémentaires, le maillage géométrique peut être le même, mais le modèle est différent au moins à cause d’une intégration différente. De même, le matériau doit contenir les composantes voulues pour calculer la masse. Dans l’exemple suivant, il est construit à partir d’un champ constant par élément mater0 :

```
[modl2,intg2] = ModlIntg4(mail1,'ELASTIQUE','MASSE',mode1);
mater2 = ChmlToCham(mater0,mail1,intg2);
[mass1,bm1] = Mass2(modl2,matr2,mail1,intg2,xcoor1,mode1);
```

(il y aura a terme bm1, l’équivalent de bsig1 pour la masse. L’équivalent de b1 pour la masse sera un simple passage des nœuds pour un champ aux points d’intégration).

En 2D, différentes analyses (ou *modes*) peuvent être réalisées : par exemple, en contraintes planes ou en déformation plane. C’est pourquoi, il faut passer le mode avec le mot mode1 aux routines précédentes.

D’autres matrices associées à d’autres opérateurs différentiels pourraient être construites. Elles peuvent nécessiter d’autres types d’intégration, de matériaux, et d’autres routines.

6.2.3 Cas des matrices et des rigidités ou des masses

La situation est similaire à celle des vecteurs et des champs par point. En effet, il s’agit maintenant de trouver la description de l’objet qui permet de faire passer d’un

vecteur primal (par exemple déplacements) à un vecteur dual (par exemple forces généralisées). On va donc avoir besoin du double d'information : l'une pour les ddl primaux, l'autre pour les ddl duaux.

Une matrice de rigidité assemblée utilise donc deux numérotation de nœuds (si elles sont identiques, la matrice est carrée) numerd1 et numerp1, ainsi que deux mappings de degrés de liberté mapddld1 et mapddlp1. Elle est alors constructible à partir des matrices élémentaires rigi1 par assemblage :

```
K1 = RigiToMatrix5(rigi1,modl1,mail1,numerd1,numerp1,mapddld1,mapddlp1);
```

avec les mêmes notions de restriction et d'extrapolation que pour les champ par points.

ATTENTION : PLUTOT NUMER ET MAPPING INVERSES !

Le cas de la matrice de masse est exactement le même. La même routine d'assemblage est donc utilisée :

```
M1 = RigiToMatrix(mass1,modl2,mail1,numerd1,numerp1,mapddld1,mapddlp1);
```

Bien entendu, pour qu'un produit matrice-vecteur (K1 * U1) ait un sens, il faut que les couples numérotation-liste de composantes qui ont servi à les construire aient été les mêmes.

6.2.4 Cas des vecteurs et des champs par élément

Pour transformer un champ (non constant) par element en vecteur, il faut procéder d'une façon tout à fait similaire au cas des champs par point. La liste des composantes listComp1 et celle des unités listUnit1 sont construites similairement. La principale différence vient du fait qu'il n'est plus nécessaire de faire un assemblage : les "points" d'un champ par élément sont en fait des points d'intégration qui sont indépendants d'un élément à un autre. La deuxième différence tient au fait qu'il n'y a pas de numérotation globale des points d'intégration comme il y en avait une sur les nœuds. Cependant cette numérotation peut être déduite de la donnée d'un modèle et d'une hypothèse sur l'ordre de rangement des points : par exemple, on peut ranger par ordre de changement le plus rapide dans le vecteur : la composante, le point d'intégration, l'élément, la sous-zone. On pourra alors faire intervenir une numérotation éventuelle numer2, comme pour les champs par point, et construire un mapping :

```
[listComp2,listUnit2] = ListCompCham2(cham1);
mapcomp2 = MapCompCham(cham1,mail1,intg1,numer2,listComp2);
S1 = ChamToVect3(cham1,mail1,numer2,mapcomp2,listComp2);
```

Si la liste de composantes fournie listComp1 est plus grande ou plus petite que celle qui est dans le champ, on procède à une extrapolation (avec des valeurs nulles) ou une restriction du champ.

6.2.5 Cas des matrices et des opérateurs de type Bsigma

IL FAUT ASSEMBLER bsig1 ! ! ! BToMatrix

Typiquement, un opérateur Bsigma permet de faire passer d'un champ par élément de constraintes à un champ par point discret de forces généralisées qui l'équilibrent. De façon plus générale, pour chaque opérateur de type rigidité correspond une tel opérateur Bsigma.

Comme cet opérateur est intimement lié à la rigidité (ou équivalent), il est construit en même temps que cette dernière : c'est l'objet complexe bsig1 construit précédemment en même temps que rigi1.

Le version matricielle B1 de cet opérateur est construite en même temps que la matrice de rigidite K1 (voir précédemment). B1 fait passer d'un vecteur (issu d'un champ par élément) S1 à un vecteur (qui pourra être transformé en champ par point) F1 :

```
BSIG1 = BToMatrix(bsig1
F1 = BSIG1' * S1;
```

À noter : la transposition dans l'utilisation de B1 (pour retrouver des notations conventionnelles en EF).

À terme, un opérateur similaire existera associé à la masse.

IL FAUT ASSEMBLER b1 ! ! !

Un autre opérateur est assez lié à ce dernier. Pour la rigidité, il s'agit de l'opérateur qui calcule la déformation à partir du champ de déplacement. On a choisi ici de le construire uniquement au niveau matriciel. Il utilise donc un vecteur U1, et bien entendu l'information relative à un vecteur U1 de champ par point de déplacement (numer1, mapddl1, listComp1) et fournit une matrice de passage qui permettra d'obtenir un vecteur E1 associé à un champ par élément, donc associé à (numer2, mapComp1, listComp2) :

```
B1 = BToMatrix(b1
E1 = B1 * U1;
```

6.3 Écriture de résultats dans un fichier AVS

C'est l'opération inverse de la lecture. On retrouve toutes les limitations précédentes dues à AVS : les coordonnées doivent être en 3D (quitte à rajouter une colonne de zéros), les champ par élément doivent être constants.

```
fid = fopen('fichier.avs','w');
  error1 = Write1AVS5(xcoor1,mail1,chpo1,chml1,cara1,fid);
fclose(fid);
```

On peut bien évidemment écrire plusieurs structures de données à la suite, comme on pouvait en lire plusieurs ; il faut remarquer, que même si l'utilisateur avait une unique structure de donnée, l'écriture de deux passes dans un fichier AVS sépare cette structure de données en deux structures de données indépendantes.

Si l'un ou l'autre des objets mail1, chpo1, chml1 ou cara1 n'existe pas, il suffit de le remplacer par une liste vide : [].

Des limitations existent ici aussi à cause du format AVS :

– le champ par point chpo1 doit porter sur tous les nœuds de xcoor1 (d'où l'absence de nmail1 en entrée de la routine), et ne doit avoir qu'une sous-zone. Un champ par point plus général peut être transformé en ce format "full" en extrapolant les valeurs manquantes par 0. Une façon brutale de faire ça consiste à transformer ce champ par point en vecteur puis le vecteur en champ par point, come cela est décrit plus haut ;

– le champ par élément (chml1, mail1) doit être un champ constant par élément et porter sur l'ensemble des éléments de mail1. De la même façon, toutes les composantes doivent être définies sur tous les éléments.

6.4 Cas tests

Afin de vérifier le bon fonctionnement, notamment lorsque les routines sont en cours de développement, une batterie de cas tests doit être mise en place. Pour être validée, toute nouvelle modification ou routine doit passer tous les cas tests.

Pour l'instant, les cas tests sont :

– `test_AVS_1.m`
pour les routines d'E/S au format AVS. Après avoir lancé ce cas test, les fichiers
`test_AVS_1.avs`
et
`test_AVS_1a.avs`
doivent être identiques ;

– `test_AVS_2.m`
pour les routines d'E/S au format AVS, et les routines de fusion et défusion de base de donnée. Après avoir lancé ce cas test, les fichiers
`test_AVS_2.avs`
et
`test_AVS_2a.avs`
doivent être identiques ;

– `test_Vect_1.m`
pour le passage vecteur-chpo. L'erreur calculée err1 doit être nulle.