Introduire un nouveau calcul élémentaire

Résumé :

Ce document décrit ce qu'il faut faire pour introduire un nouveau calcul élémentaire dans *Code_Aster* .

En quelques mots, il faut :

- introduire un petit bloc de texte dans le catalogue d'un `type_element`
- écrire une nouvelle routine fortran de nom `te00ij.f` où `00ij` est un nombre à 4 chiffres

Table des Matières

1 Introduction

Pour Code_Aster, un calcul élémentaire correspond à un couple (type d'élément fini, option de calcul). Exemples de types d'élément finis (type_element) :

- MEDKTR3 : élément DKT triangulaire à 3 nœuds
- THER_PENTA15 : élément de thermique pentaèdre à 15 nœuds

Exemples d'options de calcul (option) :

- RIGI_MECA : calcul de la rigidité (comportement élastique)
- FLUX_ELGA : calcul du flux thermique connaissant la température aux nœuds

Dans le reste de ce document, l'exemple qui nous servira de fil conducteur sera celui du calcul du flux thermique aux points d'intégration (FLUX_ELGA) des éléments PENTA15 de la modélisation '3D ' du phénomène ' THERMIQUE ' (type_element = THER_PENTA15).

On va supposer que ce calcul élémentaire n'existe pas encore mais que l'option FLUX_ELGA existe déjà (pour d'autres éléments finis) et que le type_element THER_PENTA15 existe également (il sait déjà calculer d'autres options). Dans ce document, nous essaierons de répondre aux questions :

- Que faut-il faire pour réaliser ce nouveau calcul élémentaire ?
- Quels fichiers sources faut-il modifier ou ajouter ?

D'autres questions relatives aux éléments finis sont traitées dans d'autres documents :

- [D5.02.03] Comment introduire une nouvelle option de calcul élémentaire ? (par exemple un nouveau post-traitement)
- [D5.02.04] Comment introduire une nouvelle famille d'éléments finis (modelisation) ?
- [D5.02.01] Comment introduire une nouvelle grandeur ou de nouvelles composantes dans une grandeur existante ?
- [D5.02.02] Comment introduire un nouveau type de maille (type_maille) ou un nouvel élément de référence (ELREFE) ?

Nous avons déjà dit dans le résumé que l'introduction d'un nouveau calcul élémentaire dans Code_Aster nécessitait 2 actions :

- l'ajout d'un bloc de texte dans le catalogue du type_element (ici THER_PENTA15)
- l'ajout (ou la modification) d'une routine fortran de nom te00ij.F90

Nous allons détailler successivement ces deux actions.

2 Modification du catalogue Elements/ther_hexa20.py

2.1 Trouver le nom du fichier à modifier

La première difficulté à résoudre est de trouver le nom du fichier catalogue à modifier. Remarquons déjà que le nom du type_element qui nous concerne (THER_PENTA15) ne nous est pas forcément familier. Nous pouvons le découvrir en utilisant la commande AFFE_MODELE sur un maillage contenant des PENTA15 :

```
MOTH = AFFE_MODELE(  MAILLAGE = MAIL,
                     AFFE=_F( TOUT = 'OUI', MODELISATION = '3D',
                                PHENOMENE = 'THERMIQUE'))
```

Dans le fichier .mess nous pouvons alors voir :

```
 SUR LES          132 MAILLES DU MAILLAGE MAIL
    ON A DEMANDE L'AFFECTATION DE          132
    ON A PU EN AFFECTER                    124

    MODELISATION      ELEMENT FINI      TYPE MAILLE          NOMBRE
    3D                THER_PENTA15      PENTA15                  40
    3D                THER_FACE6        TRIA6                     4
    3D                THER_FACE8        QUAD8                    80
```

Nous voyons que la modélisation appelée '3D' appliquée sur des PENTA15 conduit à l'affectation d'éléments finis de type THER_PENTA15 . C'est le nom que nous cherchions.

Nous aurions pu aussi trouver ce nom en consultant le catalogue :

```
.../catalo/cataelem/Commons/phenomenons_modelisations.py
…

phen.add('3D', Modelisation(dim=(3,3), code='3D_',
    elements=(
        (MT.HEXA8     , EL.THER_HEXA8),
        (MT.PENTA6    , EL.THER_PENTA6),
        (MT.TETRA4    , EL.THER_TETRA4),
        (MT.PYRAM5    , EL.THER_PYRAM5),
        (MT.QUAD4     , EL.THER_FACE4),
        (MT.TRIA3     , EL.THER_FACE3),
        (MT.HEXA27    , EL.THER_HEXA27),
        (MT.HEXA20    , EL.THER_HEXA20),
        (MT.PENTA15   , EL.THER_PENTA15),
```

Ce catalogue donne le nom de tous les type_element associés aux différents éléments finis de la modélisation '3D'. Pour le type de maille PENTA15 , le nom du type_element est bien THER_PENTA15 .

Une fois connu le nom THER_PENTA15 , pour trouver le nom du catalogue concernant cet élément fini, il faut faire un grep dans les fichiers de catalo/cataelem/Elements/* :
grep -wl THER_PENTA15 Elements/*.py

Ce que devrait donner : Elements/ther_hexa20.py

C'est le nom du fichier que nous devons modifier.

Si nous éditons ce fichier, nous voyons qu'il contient la description de plusieurs éléments : THER_HEXA20, THER_PENTA15, ...

Les calculs élémentaires réalisés par les éléments finis de ce fichier sont en général décrits lors de la description du premier élément (ici THER_HEXA20). Les autres éléments étant obtenus par héritage du premier. Le premier élément est, en quelque sorte, un élément « modèle » (pour les autres).

Pour que l'élément THER_PENTA15 puisse calculer l'option FLUX_ELGA, le plus simple est d'ajouter ce calcul élémentaire dans la description de l'élément « modèle »

Ce qui veut dire que notre nouveau calcul élémentaire sera (par défaut) accessible à tous les éléments finis décrits dans le fichier catalogue.

En général, c'est ce que l'on souhaite car si on sait faire un calcul sur un PENTA15 , on sait également le faire pour le TETRA4 , le PENTA6 , ...

Dans la suite du document, on continuera à se référer à l'élément THER_PENTA15 , mais en réalité, notre développement concernera tous les éléments volumiques de la modélisation '3D' (THER_TETRA4 , THER_TETRA10 , ..., THER_HEXA27).

Si l'on ne souhaitait pas rendre ce calcul élémentaire disponible pour les PYRAM13 par exemple, on modifierait la description du PYRAM13 :

```
class THER_PYRAM13(THER_HEXA20):
    meshType = MT.PYRAM13
    elrefe =(... )

    calculs = (
        OP.FLUX_ELGA(te=-1),
```

le nombre "-1" en regard de l'option FLUX_ELGA indique que ce calcul élémentaire n'est pas disponible pour les PYRAM13 . Une erreur fatale sera émise si l'utilisateur tente de l'utiliser.

2.2 Modifier le fichier ther_hexa20.py

Le bloc de texte à ajouter dans l'argument « calculs » est :

```
    OP.FLUX_ELGA(te=62,
        para_in=((SP.PCAMASS, CCAMASS), (SP.PGEOMER, NGEOMER),
                 (SP.PMATERC, LC.CMATERC), (SP.PTEMPER, DDL_THER),
                 (SP.PTEMPSR, CTEMPSR), (OP.FLUX_ELGA.PVARCPR, LC.ZVARCPG),
                     ),
        para_out=((OP.FLUX_ELGA.PFLUXPG, EFLUXPG), ),
        ),
```

Dans ce bloc de texte, on peut reconnaître différents items :

- le nom de l'option que l'on veut "réaliser" (FLUX_ELGA)
- Le nombre "62" qui est le numéro de la routine te00ij.F90 associée au calcul élémentaire (ici : te0062.F90)
- le bloc des champs d'entrée du calcul élémentaire (argument para_in)
- le bloc des champs de sortie du calcul élémentaire (argument para_out)

Remarque :

un champ est soit " IN " soit " OUT ", il ne peut pas être " INOUT ". Les blocs des champs " IN " et " OUT " sont décrits par des listes de couples (paramètre, *mode_local).*
Ici, il y a 6 champs " IN " et un seul champ " OUT ".

2.2.1 Champ de sortie

Commençons par le champ "OUT" (but du calcul élémentaire) :

PFLUXPG (le paramètre) est le nom donné au champ de flux résultat pour l'option FLUX_ELGA .

Ce nom a été choisi lorsque le catalogue de l'option (Options/FLUX_ELGA.py) a été introduit dans le code. Ce nom est utilisé dans tous les catalogues des éléments finis qui calculent l'option FLUX_ELGA .

EFLUXPG est le mode local associé au paramètre. Il permet de décrire la structure du champ local de flux pour les éléments THER_PENTA15 . Nous y reviendrons plus tard.

Regardons ce que contient le catalogue de l'option FLUX_ELGA concernant son champ de sortie :

```
PFLUXPG  = OutputParameter(phys=PHY.FLUX_R, type='ELGA',
           comment="""  PFLUXPG : FLUX AUX POINTS DE GAUSS """)

FLUX_ELGA = Option(
    para_in=(
        SP.PCACOQU,
        ...
           PVARCPR,
    ),
    para_out=(
           PFLUXPG,
    ),
```

On voit que le champ paramètre PFLUXPG est un champ de la grandeur FLUX_R et que c'est un champ par éléments aux points de Gauss (type='ELGA')

C'est le "cahier des charges" du développement que nous devons réaliser : nous devons calculer un champ de FLUX_R sur les points de Gauss du THER_PENTA15 . Le choix du type de champ (ici ELGA) est imposé par l'option. En revanche les composantes de la grandeur FLUX_R que l'on va utiliser sont au choix de l'élément THER_PENTA15 . C'est l'objet du mode_local EFLUXPG .

Ce mode_local est décrit dans le catalogue de l'élément :

```
EFLUXPG  = LocatedComponents(phys=PHY.FLUX_R, type='ELGA', location='RIGI',
           components=('FLUX','FLUY','FLUZ',))
```

On y voit que ce mode_local concerne bien la grandeur FLUX_R et qu'il décrit bien la structure d'un champ "ELGA" (c'est dans le cahier des charges imposé par l'option). Le choix des composantes : FLUX , FLUY et FLUZ n'a rien d'étonnant puisque l'élément est 3D .

Le nom des composantes que l'on peut utiliser dans la description d'un mode_local est donné dans le catalogue des grandeurs (Commons/physical_quantities.py). On peut y lire concernant la grandeur FLUX_R :

```
FLUX_R   = PhysicalQuantity(type='R',
    components=(
       'FLUX',
       'FLUY',
       'FLUZ',
       'FLUX_INF',
...
```

2.2.2 Champs d'entrée

La liste des champs d'entrée a la même structure que celle les champs de sortie : c'est une liste de couples (parametre , mode_local). Les paramètres sont imposés par le catalogue de l'option.

Pour FLUX_ELGA on trouve dans le catalogue de l'option 9 champs paramètres :

```
FLUX_ELGA = Option(
    para_in=(
        SP.PCACOQU,
        SP.PCAMASS,
        SP.PGEOMER,
        SP.PHARMON,
        SP.PMATERC,
           PNBSP_I,
        SP.PTEMPER,
        SP.PTEMPSR,
           PVARCPR,
    ),
```

Il faut choisir dans cette liste les paramètres qui seront utiles au type_element THER_PENTA15 . Après réflexion, on retient les 5 paramètres suivants :

- PTEMPER : c'est le champ de température dont il va falloir calculer le gradient
- PGEOMER : le champ de géométrie de l'élément (ses coordonnées) est nécessaire pour calculer le gradient.
- PMATERC : le champ de matériau est nécessaire pour pouvoir connaître la conductivité thermique (lambda)
- PCAMASS : c'est le champ contenant l'éventuel repère local des éléments 3D . Il est nécessaire si le matériau n'est pas isotrope : les caractéristiques anisotropes sont données dans un repère local.
- PTEMPSR : ce champ sert à transmettre l'instant du calcul. Il est nécessaire ici car, en thermique, lorsque les coefficients matériau sont des fonctions (DEFI_MATERIAU / THER_FO par exemple), ces fonctions peuvent dépendre du temps.

Une fois que l'on a retenu les paramètres utiles au type_element , il faut leur attribuer à chacun un mode_local . On va choisir :

```
PTEMPER → DDL_THER
PGEOMER → NGEOMER
PCAMASS → CCAMASS
PMATERC → CMATERC
PTEMPSR → CTEMPSR
```

Ces modes locaux doivent être décrits dans le catalogue des modes locaux partagés (Commons/located_components.py) ou dans le catalogue de l'élément (ther_hexa20.py).

Commentons ces modes locaux :

2.2.2.1 DDL_THER

```
DDL_THER = LocatedComponents(phys=PHY.TEMP_R, type='ELNO',
           components=('TEMP',))
```

Ce mode_local indique que le champ local attendu par l'élément fini est un champ sur les nœuds de l'élément (ELNO).

Tous les nœuds de l'élément (15 pour un PENTA15) portent les mêmes composantes.
La liste des composantes portées par ces nœuds est réduite à une seule composante de la grandeur TEMP_R : TEMP .

2.2.2.2 NGEOMER

```
NGEOMER  = LocatedComponents(phys=PHY.GEOM_R, type='ELNO',
           components=('X','Y','Z',))
```

Ce mode_local indique que le champ local attendu par l'élément fini est un champ sur les nœuds de l'élément (ELNO).

La liste des composantes portées par ces nœuds est X , Y , Z car l'élément est 3D .

2.2.2.3 CMATERC

```
CMATERC = LocatedComponents(phys=PHY.ADRSJEVE, type='ELEM',
                            components=('I1',))
```

Ce mode_local indique que le champ local attendu par l'élément fini est un champ constant sur l'élément (ELEM).

La liste des composantes portées par cet élément est I1 .

Pour des raisons de performance, la structure informatique représentant le matériau au niveau d'un élément est dite "codée" (on parle alors de matériau codé), elle est représentée par un nombre entier (composante I1 de la grandeur ADRSJEVE) qui est une adresse mémoire. On verra plus tard comment ce matériau est utilisé dans les routines utilitaires.

2.2.2.4 CCAMASS

```
CCAMASS  = LocatedComponents(phys=PHY.CAMASS, type='ELEM',
           components=('C','ALPHA','BETA','KAPPA','X','Y','Z',))
```

Ce mode_local correspond à un champ constant sur l'élément (ELEM).

Les 7 composantes (C , ALPHA , ...) sont des nombres réels qui permettent de représenter le changement de repère Global → Local

Nous n'en dirons pas plus ici.

2.2.2.5 CTEMPSR

```
CTEMPSR  = LocatedComponents(phys=PHY.INST_R, type='ELEM',
           components=('INST','DELTAT','THETA','KHI','R','RHO',))
```

On remarque que 6 composantes sont indiquées dans le mode_local : INST , DELTAT , ... Pourtant seule la valeur de l'instant INST nous est utile pour pouvoir évaluer d'éventuelles fonctions du temps. Les autres composantes sont probablement inutiles et elles pourraient être retirées du mode_local .

Pourtant, il ne faut pas le faire trop brutalement. En effet, le mode_local CTEMPSR est utilisé par d'autres options qui elles ont besoin de plus d'informations (DELTAT : valeur du pas de temps, ...)

Si on voulait améliorer la lisibilité du catalogue de notre élément, on pourrait dédoubler ce mode_local :

```
CTEMPSR  = LocatedComponents(phys=PHY.INST_R, type='ELEM',
           components=('INST','DELTAT','THETA','KHI','R','RHO',))
CTEMPS1  = LocatedComponents(phys=PHY.INST_R, type='ELEM',
           components=('INST',))
```

Pour notre option FLUX_ELGA , on pourrait alors décrire le champ local d'instant par le couple (PTEMPSR, CTEMPS1)

3 Écrire (ou modifier) la routine fortran te0062.F90

Nous avons déjà vu que le catalogue de l'élément THER_PENTA15 indiquait que le calcul de l'option FLUX_ELGA se passera dans la routine fortran te0062.F90 . L'objet de cette routine est de calculer (pour un élément fini) le champ local de flux sur les points de Gauss de l'élément.
De façon générale, une routine te00ij.F90 a toujours comme but de calculer des champs "OUT" à partir de ses champs " IN ".

3.1 Arguments de la routine

Toutes les routines te00ij.F90 ont les mêmes arguments. La raison en est que la routine qui les appelle (te0000.F90) est écrite une fois pour toutes et qu'on ne veut pas la changer à chaque fois que l'on ajoute un calcul élémentaire.

Les deux seuls arguments des routines te00ij.F90 sont des arguments d'entrée :

- OPTION est une chaîne de caractères contenant le nom de l'option (pour nous : FLUX_ELGA)
- NOMTE est une chaîne de caractères contenant le nom du type_element (pour nous : THER_PENTA15)

Ces 2 arguments peuvent être utilisés (ou pas).

On utilise l'argument OPTION lorsque l'on traite dans une même routine te00ij.F90 2 options (ou plus) qui se ressemblent.

On peut alors écrire des tests comme :

```
...
IF (OPTION.EQ.'FLUX_ELGA') THEN
   ...
ELSE IF (OPTION.EQ.'FLUX_ELNO') THEN
   ...
ENDIF
```

De la même façon, on traite en général tous les éléments d'une même modélisation (TETRA4 , TETRA10 , ...) dans la même routine te00ij.F90 . L'argument NOMTE permet de distinguer certains traitements. On pourrait par exemple imaginer un petit bloc de code propre aux éléments pyramidaux.

Mais les véritables arguments des routines te00ij.F90 sont en réalité leurs champs "IN" et leurs champs "OUT". Ces arguments sont "souterrains" et on y accède via les 2 routines utilitaires JEVECH et TECACH que l'on va présenter dans le paragraphe suivant.

3.2 Présentation de quelques utilitaires utilisés dans le te0062.F90

3.2.1 Routine JEVECH

Cette routine permet de récupérer l'adresse mémoire d'un champ local. Par exemple, dans la routine te0062.F90 , on trouve :

```
CALL JEVECH('PGEOMER','L',IGEOM)
```

Le 1er argument de la routine JEVECH est le nom du paramètre qui nous intéresse.
Le 2ème argument est "documentaire" (il n'est pas utilisé dans le code). Il indique si le paramètre est un champ " IN " (accès en lecture 'L') ou un champ "OUT" (accès en écriture 'E').
Le 3ème argument (de sortie) est l'adresse de la zone mémoire contenant le champ local.

C'est au programmeur de la routine te0062.F90 d'assurer la cohérence entre l'usage qu'il fait de cette adresse et ce qui est écrit dans le catalogue de l'élément concernant ce paramètre :

- Puisque le paramètre PGEOMER est associé à la grandeur GEOM_R et que cette grandeur est de type réel, l'adresse IGEOM est une adresse dans le common JEVEUX ZR
- Puisque le mode_local NGEOMER (associé à PGEOMER) est décrit dans le catalogue comme un champ aux nœuds ayant 4 composantes (X Y Z) par nœud, c'est au programmeur de savoir que le champ local représenté en mémoire à l'adresse ZR(IGEOM) est de longueur 3*15 (3 composantes X , Y , Z pour chacun des 15 nœuds).

Plus précisément, le programmeur doit savoir qu'il va trouver dans la mémoire JEVEUX :

```
ZR(IGEOM-1+1) -> 'X' du noeud 1
ZR(IGEOM-1+2) -> 'Y' du noeud 1
ZR(IGEOM-1+3) -> 'Z' du noeud 1
ZR(IGEOM-1+4) -> 'X' du noeud 2
ZR(IGEOM-1+5) -> 'X' du noeud 2
...
ZR(IGEOM-1+45) -> 'Z' du noeud 15
```

3.2.2 Routine ELREFE_INFO
À compléter ...

3.2.3 Routine DFDM3D
À compléter …

3.2.4 Routine RCVALA
À compléter ...

3.3 Routine TE0062

```
      SUBROUTINE TE0062(OPTION,NOMTE)
      IMPLICIT NONE
      CHARACTER*16 OPTION,NOMTE

C --- DEBUT DECLARATIONS NORMALISEES JEVEUX ----------------------------
      INTEGER ZI
      COMMON /IVARJE/ZI(1)
      REAL*8 ZR
      COMMON /RVARJE/ZR(1)
      COMPLEX*16 ZC
      COMMON /CVARJE/ZC(1)
      LOGICAL ZL
      COMMON /LVARJE/ZL(1)
      CHARACTER*8 ZK8
      CHARACTER*16 ZK16
      CHARACTER*24 ZK24
      CHARACTER*32 ZK32
      CHARACTER*80 ZK80
      COMMON /KVARJE/ZK8(1),ZK16(1),ZK24(1),ZK32(1),ZK80(1)
C --- FIN DECLARATIONS NORMALISEES JEVEUX ------------------------------

      INTEGER ICODRE
      CHARACTER*8 NOMRES(1)
      REAL*8 LAMBDA,FLUXX,FLUXY,FLUXZ,DFDX(27),DFDY(27),DFDZ(27),POIDS
      INTEGER JGANO,IPOIDS,IVF,IDFDE,IGEOM,IMATE,NNO,KP,NPG1,I,IFLUX,
     &        ITEMPS,ITEMPE,NDIM,NNOS
C.......................................................................


C     -- récupération d'informations concernant l'élément de référence :
      CALL ELREFE_INFO(FAMI='RIGI',NPG=NPG1,
     &                 JPOIDS=IPOIDS,JVF=IVF,JDFDE=IDFDE)

C     -- récupération des adresses des champs locaux :
      CALL JEVECH('PGEOMER','L',IGEOM)
      CALL JEVECH('PMATERC','L',IMATE)
      CALL JEVECH('PTEMPSR','L',ITEMPS)
      CALL JEVECH('PTEMPER','L',ITEMPE)
      CALL JEVECH('PFLUX_R','E',IFLUX)

C     -- récupération de la conductivité LAMBDA :
      CALL RCVALA(ZI(IMATE),' ','THER',1,'INST',ZR(ITEMPS),1,'LAMBDA',
     &            LAMBDA,ICODRE,1)


C     -- boucle sur les points de Gauss :
      DO 20 KP=1,NPG1
C       -- récupération des dérivées des fonctions de forme
C          (sur l'élément réel) :
        CALL DFDM3D(NNO,KP,IPOIDS,IDFDE,ZR(IGEOM),DFDX,DFDY,DFDZ,POIDS)

C       -- calcul du gradient du champ de température :
        FLUXX=0.0D0
        FLUXY=0.0D0
        FLUXZ=0.0D0

        DO 10 I=1,NNO
```

```
            FLUXX=FLUXX+ZR(ITEMPE-1+I)*DFDX(I)
            FLUXY=FLUXY+ZR(ITEMPE-1+I)*DFDY(I)
            FLUXZ=FLUXZ+ZR(ITEMPE-1+I)*DFDZ(I)
   10     CONTINUE

C         -- calcul du flux et stockage dans le champ résultat:
          ZR(IFLUX+(KP-1)*3)   =-LAMBDA*FLUXX
          ZR(IFLUX+(KP-1)*3+1)=-LAMBDA*FLUXY
          ZR(IFLUX+(KP-1)*3+2)=-LAMBDA*FLUXZ
   20   CONTINUE

        END
```

Commentaires sur cette routine :

Il ne faut pas oublier que cette routine doit pouvoir traiter tous les éléments 3D : TETRA4 , ..., HEXA27 . C'est pourquoi, Les tableaux DFDX , DFDY et DFDZ sont dimensionnés à 27 qui est le "max" du nombre de nœuds d'un élément volumique.

La famille de points de Gauss qui est utilisée ici est 'RIGI'. C'est elle qui est donnée en argument de la routine ELREFE_INFO . Cette famille est cohérente avec le choix fait dans le catalogue :

```
EFLUXPG  = LocatedComponents(phys=PHY.FLUX_R, type='ELGA', location='RIGI',
                             components=('FLUX','FLUY','FLUZ',))
```

La routine RCVALA est utilisée pour récupérer la valeur de la conductivité thermique (LAMBDA). Comme ce paramètre peut être une fonction du temps, on fournit la valeur de l'instant en argument d'entrée de la routine RCVALA .

le calcul du gradient de la température est fait en deux temps. La routine DFDM3D permet de calculer le gradient des fonctions de forme sur l'élément en 1 point de Gauss donné. On peut ensuite utiliser le gradient des fonctions de forme et la valeur de la température sur les nœuds pour calculer le gradient de la température.

Le stockage des flux calculés à l'adresse IFLUX respecte bien la convention de stockage des champs locaux : 3 composantes FLUX , FLUY et FLUZ pour chaque point de Gauss.

4 Détails non utilisés dans l'exemple choisi

4.1 Description de l'entête d'un type_element

À compléter …

4.2 Modes Locaux ELNO / DIFF

À compléter …

4.3 Conventions de noms pour les modes locaux

À compléter …

4.4 À compléter …Noms réservés pour certains modes locaux (ddl_meca, ddl_ther, ddl_acou)

À compléter …

4.5 Champs locaux de type vecteur élémentaire ou matrice élémentaire

À compléter …

4.6 Champs facultatifs, routine tecach.F90

À compléter …

4.7 Famille de points de Gauss "MATER"

À compléter …