Introduire une nouvelle option de calcul élémentaire

Résumé :

Ce document décrit ce qu'il faut faire pour introduire une nouvelle option de calcul élémentaire dans *Code_Aster* .

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Table des Matières

1 Introduction....3
2 Trouver un nom pour ce catalogue....4
3 Contenu du catalogue....4
3.1 Paramètres....5
3.2 Option....5
3.3 Bloc condition / CondCalcul....5
3.4 Commentaire....6
3.5 Remarques....6

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

1 Introduction

Pour Code_Aster, une option de calcul élémentaire correspond à un calcul conduisant à produire un ou plusieurs champs « par éléments »

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
- `CHAR_MECA_PESA_R` : calcul du second membre lié à un chargement de pesanteur.

Les 3 exemples précédents montrent que les calculs élémentaires peuvent intervenir un peu partout dans Code_Aster : dans les commandes de calcul (matrices et seconds membre), comme dans les commandes de post-traitement (calcul des flux en thermique, des énergies mécaniques, ...)

Une `option` de calcul élémentaire a un nom (K16) et est décrite dans un catalogue rangé dans le répertoire `catalo/cataelem/Options`

Faire un calcul élémentaire correspond à demander le calcul d'une `option` sur une liste d'éléments finis (`ligrel`). Ce calcul est réalisé par l'appel à la routine « chapeau » de tous les calculs élémentaires : `calcul.F90`

Le calcul est dit « élémentaire » car ce sont les éléments finis du `ligrel` qui font le calcul. Pour qu'un calcul soit « élémentaire », il faut qu'il soit « local », c'est à dire qu'un élément fini puisse calculer sa participation en ayant uniquement connaissance de données locales (champs locaux sur lui-même).
Ne connaissant que des quantités locales, il calcule logiquement un résultat « local » (la participation de l'élément fini).
La collection des résultats calculés par les différents éléments constitue un champ (global) « par éléments ». C'est un champ par nature discontinu entre les éléments.

Remarque : le calcul d'une option de calcul élémentaire peut produire plusieurs champs. C'est par exemple le cas de l'option `RAPH_MECA` utilisée dans l'opérateur `STAT_NON_LINE` : on calcule en même temps un champ de contrainte, un champ de variables internes et un champ de « résidus ».

Le champ global produit par un calcul élémentaire peut-être un `cham_elem` ou un `resuelem`. La différence entre les deux types de structure de données est petite : le `resuelem` est un champ élémentaire contenant des matrices ou vecteurs élémentaires destinés à être assemblés pour former une matrice globale ou un second membre.

Lorsqu'un calcul élémentaire produit un `cham_elem`, celui-ci peut-être `ELNO`, `ELGA` ou `ELEM`. Un champ `ELNO` contient des valeurs sur les nœuds des éléments. Un champ `ELEM` est un champ constant par maille. Un champ `ELGA` est un champ contenant des valeurs sur les points d'intégration (ou de Gauss) des éléments.

Dans le paragraphe suivant, nous allons détailler la syntaxe que doit avoir le catalogue d'une option.
Écrire le catalogue d'une nouvelle option n'est pas une fin en soi. Ce n'est qu'une étape préliminaire pour que des éléments finis puissent calculer cette option. Ceci est expliqué dans le document [D5.02.05] « Introduire un nouveau calcul élémentaire ».

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2 Trouver un nom pour ce catalogue

L'ensemble des catalogues d'option de *Code_Aster* se trouve dans le répertoire `Options` du répertoire `catalo/cataelem`. L'objectif est de concevoir un catalogue relatif à cette nouvelle option et de le stocker dans le répertoire `Options`.

Par convention, le nom du catalogue d'option est celui de l'option. Le nom d'une option a au plus 16 caractères. Ainsi, le catalogue de l'option 'FLUX_ELGA' sera `flux_elga.py`.

3 Contenu du catalogue

Le catalogue choisi comme exemple est le suivant (`flux_elga.py`) :

```
from cataelem.Tools.base_objects import InputParameter, OutputParameter,
Option, CondCalcul
import cataelem.Commons.physical_quantities as PHY
import cataelem.Commons.parameters as SP
import cataelem.Commons.attributes as AT

PNBSP_I  = InputParameter(phys=PHY.NBSP_I, container='CARA!.CANBSP',
comment="""  PNBSP_I :  NOMBRE DE COUCHE """)

PVARCPR  = InputParameter(phys=PHY.VARI_R, container='VOLA!
&&CCPARA.VARI_INT_N',
comment="""  PVARCPR : VARIABLES DE COMMANDE  """)

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


FLUX_ELGA = Option(
    para_in=(
        SP.PCACOQU,
        SP.PCAMASS,
        SP.PGEOMER,
        SP.PHARMON,
        SP.PMATERC,
           PNBSP_I,
        SP.PTEMPER,
        SP.PTEMPSR,
           PVARCPR,
    ),
    para_out=(
           PFLUXPG,
    ),
    condition=(
      CondCalcul('+', ((AT.PHENO,'TH'),(AT.BORD,'0'),)),
    ),
    comment="""  FLUX_ELGA : CALCUL DU FLUX AUX POINTS DE GAUSS """,
)
```


Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Commentons ce fichier :

3.1 Paramètres

On commence par décrire les paramètres de l'option qui sont spécifiques à l'option (c'est à dire, ceux qui ne sont pas des paramètres « partagés » trouvés dans `cataelem.Commons.parameters`).

Il y a des paramètres d'entrée (`InputParameter`) et des paramètres de sortie (`OutputParameter`)

Pour chaque paramètre, il faut indiquer :
- son nom (ce qui est à gauche du signe '=')
- la grandeur associée à ce paramètre
- un commentaire

Si c'est un paramètre d'entrée, on peut également indiquer une « localisation » du champ qui est associé à ce paramètre. Cette localisation est une chaîne de caractère ayant un format particulier : plusieurs « champs » séparés par le signe « ! ». Cette localisation permet d'indiquer au programme où trouver (par défaut) le champ paramètre : dans quelle structure de données, avec quel numéro d'ordre si cette structure de données est une `sd_resultat` .

Exemple de localisation : `container='CARA!.CANBSP'`
Ce qui veut dire : le nom du champ est obtenu en concaténant la chaîne `'.CANBSP'` au nom de la `sd_cara_elem`.

Si c'est un paramètre de sortie, il faut ajouter une information sur le « type » du champ associé à ce paramètre (`type='xxx'`). Les types possibles sont :
- `'ELEM'` : pour un champ constant par élément,
- `'ELGA'` : pour un champ aux points de Gauss de l'élément,
- `'ELNO'` : pour un champ aux nœuds par élément.
- `'RESL'` : pour un champ de type `resuelem` (matrice ou vecteur).

3.2 Option

On décrit ensuite l'option proprement dite (`FLUX_ELGA = Option(...)`).
A gauche du signe '=', on trouve le nom de l'option (en majuscules).
La description de l'option est très simple : on indique la liste des paramètres d'entrée (`para_in=(...)`) et la liste des paramètres de sortie (`para_out=(...)`).

Les paramètres apparaissant dans ces deux listes sont soit des paramètres partagés (par exemple : `SP.PMATERC`) soit des paramètres définis plus haut dans le catalogue de l'option (par exemple: `PFLUXPG`)

3.3 Bloc `condition` / `CondCalcul`

Ce bloc est très important. Il permet de définir l'ensemble des éléments qui sont concernés par l'option (ceux qui devraient la calculer).

Prenons l'exemple du bloc correspondant à l'option `RIGI_MECA_GE` :

```
  condition=(
      CondCalcul('+', ((AT.PHENO,'ME'),(AT.BORD,'0'),)),
      CondCalcul('-', ((AT.PHENO,'ME'),(AT.DISCRET,'OUI'),)),
    ),
```


Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Il s'agit de désigner l'ensemble de **tous** les éléments finis pour lesquels cette option a du sens (et qui donc devraient savoir la calculer). En les désignant tous, on exprime, a crontrario, que tous les autres éléments ne sont pas concernés par cette option (et donc qu'ils ne doivent pas calculer cette option).

On désigne des « paquets » d'éléments finis utilisant les attributs définis dans les catalogues d'éléments ou dans le catalogue phenomenons_modelisations.py.

Dans l'exemple précédent, on définit deux « paquets » d'éléments :
1) ceux qui ont l'attribut PHENO='ME' **et** l'attribut BORD='0'
2) ceux qui ont l'attribut PHENO='ME' **et** l'attribut DISCRET='OUI'

L'ensemble de tous les éléments devant calculer l'option est obtenue en « ajoutant » ou en « retirant » les paquets définis selon le « signe » de CondCalcul ('+' ou '-').

Dans l'exemple précédent, l'ensemble des éléments devant calculer l'option RIGI_MECA_GE est obtenu en prenant tous les éléments du phénomène MECANIQUE , qui sont « principaux » (BORD='0') et en retirant les éléments discrets.

Le nombre de « lignes » CondCalcul n'est pas limité. L'ordre de ces lignes important : on constitue la liste des éléments concernés par l'option par ajouts et suppressions de paquets dans l'ordre de la liste.

3.4 Commentaire

On termine la description d'une option par un champ commentaire (argument comment= '…').
Ce commentaire peut être utilisé, par exemple, pour clarifier un message d'erreur.
Il doit décrire explicitement le but de l'option de calcul.

Par exemple, pour l'option FLUX_ELGA, on peut écrire:

```
Comment='FLUX_ELGA : CALCUL DU FLUX THERMIQUE AUX POINTS DE GAUSS'
```

3.5 Remarques

A chaque champ paramètre de l'option correspond un couple (nom du paramètre, grandeur). La grandeur est souvent suffisante pour caractériser le champ. Par exemple, quand on parle du champ de GEOM_R (géométrie des nœuds), tout le monde comprend de quoi il s'agit.

Mais il arrive qu'une option nécessite plusieurs champs d'une même grandeur. Penser par exemple à une option qui aurait comme champs d'entrée 2 champs de déplacements (grandeur DEPL_R) : l'un correspondant au déplacement au début du pas de temps et l'autre correspondant à un incrément de déplacement.
Les paramètres servent à distinguer ces différents champs. On associe un « petit nom » à chaque champ : c'est le nom du paramètre.

Pour notre exemple, on écrirait :

```
PDEPLMR = InputParameter(phys=PHY.DEPL_R,
                         comment=""" DEPLACEMENT INSTANT PRECEDENT """)
PDEPLPR = InputParameter(phys=PHY.DEPL_R,
                         comment=""" INCREMENT DE DEPLACEMENT. """)
```

Quand il n'y a pas d'ambiguïté (un seul champ pour une grandeur donnée), il est d'usage de nommer le paramètre associé à cette grandeur en ajoutant un 'P' devant le nom de la grandeur. Par exemple :

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
PCACOQU = InputParameter(phys=PHY.CACOQU, ...)
```

Le catalogue d'une option est utilisé par tous les éléments finis qui calculent cette option. La liste des champs paramètres doit donc être une liste « enveloppe » des champs utilisés par les éléments.

Par exemple, les éléments de coque ont en général besoin d'informations géométriques (épaisseur, ...) qui ne se trouvent pas dans le champ de géométrie (coordonnées des nœuds du maillage). Ces données géométriques manquantes se trouvent dans un champ de la structure de données CARA_ELEM (de la grandeur cacoqu). On trouvera donc souvent dans le catalogue d'une option le bloc :

```
PCACOQU = InputParameter(phys=PHY.CACOQU, ...)
```

Comme le catalogue d'une option est une « enveloppe » des besoins des éléments, la liste des champs paramètres d'une option peut être longue. C'est pourquoi il est important de commenter tous les champs paramètres.

On impose que le responsable d'une option choisisse pour chaque champ de sortie un «type» (ELGA , ELNO , …). Le but de cette contrainte est de forcer une certaine homogénéité pour les différents éléments finis qui calculent cette option. Cela permet aussi de pouvoir « typer » le cham_elem global produit. Un cham_elem sera ainsi toujours soit ELGA , soit ELNO soit ELEM .

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)