Introduire une nouvelle grandeur (ou une nouvelle composante)

Résumé :

Ce document décrit ce qu'il faut faire pour introduire une nouvelle grandeur dans *Code_Aster* ou une nouvelle composante dans une grandeur existante.

En quelques mots, pour ajouter une grandeur, il faut :

- modifier le catalogue décrivant les grandeurs,
- ajouter le nom de la grandeur dans un catalogue de commandes.

Pour introduire une composante dans une grandeur existante, il suffit :

- d'enrichir le catalogue des grandeurs.

Table des Matières

1 Introduction

Pour Code_Aster, une grandeur est composée d'un identifiant (son nom), d'un type et d'une liste de composantes.

Exemple de grandeur :

- DEPL_R : nom de la grandeur des déplacements réels aux nœuds. On associe à cette grandeur le type réel R et les composantes DX, DY, DZ, DRX, DRY, DRZ, ...

Dans *Code_Aster*, on distingue les grandeurs simples et les grandeurs élémentaires. Les grandeurs élémentaires sont attachées aux vecteurs ou matrices élémentaire. Elles sont construites à partir des grandeurs simples. Toutes ces grandeurs (simples et élémentaires) sont décrites dans le fichier Commons/physical_quantities.py. La description des grandeurs élémentaires est faite au [§4]

Nous essaierons de répondre aux questions :

- que faut-il faire pour ajouter une nouvelle grandeur ?
- que faut-il faire pour ajouter une nouvelle composante dans une grandeur existante ?
- quels catalogues faut-il modifier ?

L'introduction d'une nouvelle grandeur nécessite deux actions:

- la modification du catalogue des grandeurs : physical_quantities.py,
- la modification du catalogue des commandes : c_nom_grandeur.capy

Nous allons détailler successivement ces deux actions.

2 Modification du catalogue des grandeurs «physical_quantities.py»

On présente dans ce paragraphe la structure du catalogue physical_quantities.py, et on décrit comment on procède pour ajouter une grandeur ou une composante.

2.1 Description du catalogue

Le catalogue physical_quantities.py est présent dans le répertoire Commons du répertoire catalo/cataelem. Nous présentons ci-dessous un extrait de ce catalogue :

```
ABSC_R   = PhysicalQuantity(type='R', components=('ABSC[4]', ),
        comment="""  ABSC_R  Type:R  Abscisse curviligne le long d'un
maillage filaire
       ABSC1 : abscisse curviligne du 1er noeud d'un SEG ou d'un POI1
       ABSC2 : abscisse curviligne du 2eme noeud d'un SEG
       ABSC3 : abscisse curviligne du 3eme noeud d'un SEG (s'il existe)
       ABSC4 : abscisse curviligne du 4eme noeud d'un SEG (s'il existe)
""")
```

Le bloc définissant une grandeur contient :

- Un argument « type='R' ». Il représente le type fortran des composantes de la grandeur.
- Un argument « components=(...) » pour définir la liste des noms des composantes de la grandeur. L'ordre des composantes est important.
  Les noms des composantes sont des chaînes ayant au plus 8 caractères. Lorsqu'il existe une « série » de composantes dont les noms sont « numérotés », on peut (et on doit!) utiliser une syntaxe condensée. Par exemple : 'ABSC[4]' remplace ('ABSC1', 'ABSC2', 'ABSC3', 'ABSC4',).
- Un argument « comment= » permettant de commenter la grandeur et ses composantes. Il est important de documenter TOUTES les composantes. Ce commentaire peut-être imprimé dans certains messages d'erreur pour aider l'utilisateur. Quand on écrit ces commentaires, c'est d'abord à l'utilisateur qu'il faut penser.

2.2 Ajout d'une grandeur

Le développeur désireux d'ajouter une grandeur (parfois appelée « grandeur simple » pour la distinguer d'une grandeur « élémentaire ») va devoir choisir un nom de grandeur, lui attribuer un type, et choisir les noms des composantes de cette grandeur.

2.2.1 Choix du type

Le type à associer à la grandeur est le type fortran des valeurs de ses composantes. Le développeur devra choisir un type parmi les choix ci-dessous:

- R : réel,
- I : entier,
- C : complexe,
- K8 : chaine de 8 caractères,
- K16 : chaine de 16 caractères,
- K24 : chaine de 24 caractères,
- ...

2.2.2 Nom de la grandeur

La première chose à faire est de définir un nom de grandeur qui soit suffisamment explicite et non utilisé dans ce catalogue.

Comment définir un nom de grandeur?

Les noms sont représentés par un chaîne d'au plus 8 caractères. Dans l'exemple précédent, les deux grandeurs définies ont pour nom : ABSC_R et STAOUDYN.

Remarque :

Il est conseillé d'introduire le type de la grandeur dans son nom (par exemple ABSC_R), car il est plus simple pour le développeur de manipuler une grandeur dont le type lui est implicitement connu. Usuellement, on le fait précéder du caractère «_» (underscore) afin de le dissocier du nom symbolique de la grandeur.

2.2.3 Noms des composantes

La seconde chose à faire est définir la liste des composantes. On doit se poser les questions suivantes :

- ai-je besoin de nommer explicitement chaque composante ?
- connaît-on par avance le nombre de composantes ?

Premier cas (le plus simple) : on connaît exactement le nombre de composantes et on souhaite nommer chacune d'elles avec un nom significatif. Pour cela, on définit pour chaque composante un nom d'au plus 8 caractères. C'est le cas de l'exemple suivant :

```
XCONTAC  = PhysicalQuantity(type='R',
    components=( 'RHON', 'MU', 'RHOTK', 'INTEG', 'COECH', ...
```

Second cas : on souhaite définir une grandeur dont le nombre de composantes est conséquent et dont le nom de chacune d'elles importe peu. Pour ce faire, il existe dans Code_Aster des grandeurs « neutres ». Les grandeurs NEUT_X (où X représente le type de la grandeur).

Exemple:

```
NEUT_R   = PhysicalQuantity(type='R', components=( 'X[30]', ),
    comment="""  NEUT_R Type:R Grandeur 'neutre' de type reel
       La signification des composantes varie d'une option a l'autre. Cette
       grandeur 'passe-partout' ne sert qu'a eviter l'introduction de
       nombreuses grandeurs sans grand interet.
       X(1)  : composante 1
       ...
       X(30) : composante 30
""")
```

Cet exemple décrit une grandeur «neutre» de type R pouvant recueillir au plus 30 composantes.

Quand le nombre de composantes dépasse 30, il est également possible d'utiliser les grandeurs « neutres » N120_R , N120_I et N480_I. Celles-ci peuvent contenir respectivement 120 composantes de type R , 120 composantes de type I et 480 composantes de type I .

Remarques :

On renseigne dans `physical_quantities.py` le nombre maximal de composantes qu'une grandeur puisse disposer. Le nombre de composantes d'une grandeur nécessaire au calcul élémentaire est défini dans chaque catalogue d'élément.
L'utilisation des grandeurs «`NEUT_X`» permet d'éviter la multiplication des grandeurs.

Un **troisième cas** peut se présenter : il existe dans Code_Aster **une** grandeur «spéciale» pouvant avoir un nombre indéterminé de composantes. Il s'agit de la grandeur **`VARI_R`** :

```
VARI_R   = PhysicalQuantity(type='R',
components=( 'VARI', ),
comment="""  VARI_R Type:R … """)
```

Cette grandeur n'a apparemment qu'une seule composante (`'VARI'`), mais en réalité, ce nombre est variable (d'un élément à l'autre par exemple). Quand on imprime cette grandeur, les composantes sont alors nommées : `'V1', 'V2', 'V3'` , ...

Remarques :

La grandeur `VARI_R` est souvent utilisée pour représenter les variables internes des lois de comportement non linéaire. Mais elle peut avoir d'autres usages.
Il existe aussi la grandeur `VAR2_R` similaire à `VARI_R`. Le développeur est invité à consulter le catalogue `physical_quantities.py` pour plus d'informations sur cette grandeur .

2.3 Ajout d'une composante dans une grandeur existante.

Il suffit d'enrichir la liste des composantes de la grandeur avec un nouveau nom. Vous n'oublierez pas de commenter cette nouvelle composante.

Remarque : il est plus prudent d'ajouter la nouvelle composante à la fin de la liste.

3 Modification du catalogue «`c_nom_grandeur.capy`»

Une deuxième étape à ne pas oublier lors de l'introduction d'une nouvelle grandeur (simple) dans Code_Aster, est la modification du catalogue `c_nom_grandeur.capy`.

Ce catalogue est présent dans le répertoire `commun` du répertoire `catapy`.

Il recense toutes les grandeurs simples présentes dans Code_Aster.

Extrait:

```
def C_NOM_GRANDEUR() : return  (
"ABSC_R",
"ADRSJEVE",
"ADRSJEVN",
"CAARPO",
"CACABL",
"CACOQU",
"CADISA",
"CADISK",
…
)
```

A quoi sert-il ?

Il est utilisé par le superviseur python pour vérifier la saisie des utilisateurs dans les fichiers de commandes. Par exemple, lors de la création d'un champ (opérateur `CREA_CHAMP`), il faut fournir au

mot-clé TYPE_CHAM une chaîne de caractères décrivant le type de champ à construire (localisation et grandeur). Si TYPE_CHAM = 'ELNO_SIEF_R' (champ de contraintes réel aux nœuds par élément), Code_Aster exploite le catalogue c_nom_grandeur.capy afin de vérifier si la chaîne de caractères 'SIEF_R' saisie par l'utilisateur correspond à une grandeur.

4 Grandeurs « élémentaires »

Jusqu'à présent, nous n'avons parlé que des grandeurs « simples ». Une grandeur « simple » est une liste de composantes nommées.

Les champs aux nœuds (cham_no_xxx), les cartes (carte_xxx) et les champs par éléments (cham_elem_xxx) sont tous associés à une grandeur « simple ». Par exemple, un champ aux nœuds de déplacement est associé à la grandeur DEPL_R. Sur chacun des nœuds du maillage, ce champ peut « porter » une (ou plusieurs) composantes de la grandeur DEPL_R.

Les grandeurs « élémentaires » sont celles qui sont associées aux structures de données resuelem (vecteurs élémentaires ou matrices élémentaires).

Un resuelem est un « champ » contenant 1 vecteur élémentaire (ou 1 matrice élémentaire) par maille. La grandeur associée à ce champ est une grandeur élémentaire.

4.1 Syntaxe

La description des grandeurs élémentaires dans le fichier physical_quantities.py est faite à la fin du catalogue (car on y utilise les grandeurs simples définies au début du fichier).

Exemples :

```
MDEP_C   = ArrayOfQuantities(elem='MS', phys= DEPL_C)
MDNS_R   = ArrayOfQuantities(elem='MR', phys= DEPL_R)
MPRE_C   = ArrayOfQuantities(elem='MS', phys= PRES_C)
VDEP_C   = ArrayOfQuantities(elem='V', phys= DEPL_C)
VDEP_R   = ArrayOfQuantities(elem='V', phys= DEPL_R)
```

Une grandeur élémentaire de type « vecteur » utilise elem='V'.
Une grandeur élémentaire de type «matrice» utilise elem='MS' (si elle est symétrique) ou 'MR' (si elle est non symétrique).

4.1.1 Grandeur « vecteur élémentaire »

Une grandeur de type « vecteur élémentaire » (par exemple VDEP_R ci-dessus) est simplement décrite par elem='V' et la grandeur simple associée à la grandeur élémentaire (ici DEPL_R).

4.1.2 Grandeur « matrice élémentaire »

Une grandeur de type «matrice élémentaire » (par exemple MDNS_R ci-dessus) est décrite par elem='MR' (matrice non symétrique) et la grandeur simple associée à la grandeur élémentaire (ici DEPL_R).

4.2 Conventions de stockage

Les scalaires (en général des réels) qui permettent de représenter une grandeur élémentaire sont nombreux : par exemple, la matrice de rigidité (symétrique) d'un élément mécanique MECA_HEXA20 contient (60*61/2) réels.

Il n'est pas question de nommer tous ces réels (comme les composantes d'une grandeur simple).

Des conventions sont nécessaires concernant le stockage de ces scalaires.

Pour un vecteur élémentaire (destiné à être assemblé pour donner un second membre), on cherche à stocker quelque chose qui ressemble à un champ « aux nœuds » pour chaque élément. La convention de stockage est naturelle. Par exemple :

N1			N2			N3			...
DX	DY	DZ	DX	DY	DZ	DX	DY	DZ	...
1	2	3	4	5	6	7	8	9	...

Pour une matrice élémentaire, deux conventions sont nécessaires :

Pour une matrice symétrique, on stocke dans l'ordre suivant :

1				
2	3			
4	5	6		
7	8	9	10	
11	12	13	14	15

Pour une matrice non-symétrique, on stocke dans l'ordre suivant :

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Sachant que l'ordre des lignes et des colonnes est le même que celui d'un vecteur élémentaire.