Opérateur FORMULE

1 But

Définir une formule à valeur réelle ou complexe à partir de son expression mathématique.

La formule sera utilisable dans une commande ultérieure comme argument de type fonction/formule ou évaluée avec des valeurs particulières des variables.

Dans de nombreuses applications, on peut tabuler cette formule pour des valeurs particulières par la commande CALC_FONC_INTERP [U4.32.01] qui produit un concept de type fonction ou fonction_c comme DEFI_FONCTION [U4.31.02] ou DEFI_NAPPE [U4.31.03].

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

2 Syntaxe

```
F = FORMULE    (

       ♦  NOM_PARA      =  nom des paramètres                          [l_K8]
       ♦  /  VALE       =  """définition de la formule réelle"""       [K]
          /  VALE_C     =  """définition de la formule complexe"""     [K]

        ◊  arg_1 = val_1,
           ...
        ◊  arg_ N = val_ N

              )
```

F est de type formule ou formule_c .

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

3 Opérandes

3.1 Définition de la fonction

Le corps de la fonction est une expression algébrique Python représentée par une chaîne de caractères. Elle doit être évaluable : c'est-à-dire respecter la syntaxe Python et les fonctions, constantes ou autres objets nécessaires à son évaluation doivent définis en argument.

Exemple :
```
alpha = 1.23
form = FORMULE(NOM_PARA='X', VALE='sin(X) * alpha', alpha=alpha)
```

L'expression de la formule est `'sin(X) * alpha'`. Pour l'évaluer, `X` est la variable de la formule. La fonction `sin` est fournie par le module `math` (voir ci-dessous). La constante `alpha` est définie dans le fichier de commande. Il est donc nécessaire de la définir en argument et d'en fournir la valeur. Ainsi, la formule pourra être évaluée où que ce soit.

Attention

Les fonctions, classes (et autres objets Python) définies dans le fichier de commandes ne sont pas disponibles en POURSUITE (Python ne peut pas les « pickler »). Les formules utilisant ce type d'argument complémentaires ne peuvent donc pas être évaluées de nouveau en POURSUITE.

Si on utilise `VALE`, la formule produite est à valeur réelle (concept de type `formule`). Si on utilise `VALE_C`, la formule est à valeur complexe (concept de type `formule_c`).
Dans les deux cas, les paramètres sont réels. Les noms des paramètres nécessaires à l'évaluation de la formule sont fournis derrière le mot-clé `NOM_PARA`.

En cas d'erreur de syntaxe, c'est le langage Python qui émet le message d'erreur et non *Code_Aster* lui-même.

Remarque

L'ordre des paramètres (mot-clé `NOM_PARA`) est important. Si on crée une formule à deux paramètres en vue de produire une nappe, le premier paramètre est le paramètre de la nappe, le second est le paramètre des fonctions composant la nappe.

3.2 Fonctions standards

Outre les signes algébriques ordinaires + - / * **, sont aussi disponibles des fonctions standards (builtins) : `min, max, abs, float …`

Attention, le signe de division désigne ici la division réelle :
`1 / 2 = 0.5`
Si on souhaite faire une opération en division entière, il faut utiliser l'opérateur // :
`1 // 2 = 0`

3.3 Fonctions mathématiques

Toutes les fonctions du module `math` de Python sont importées par défaut. Elles sont donc directement utilisables dans le corps des formules.

http://docs.python.org/lib/module-math.html

```
sin         sinh
cos         cosh
tan         tanh
atan        sqrt
```


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

```
atan2       log
asin        log10
acos        exp
```

De plus, la constante `pi`, du même module, est également disponible.

Attention :

> *Les fonctions trigonométriques sont donc celles de Python et attendent des angles exprimés en radians. Il faut être vigilant sur la cohérence avec les mots clés simples `ANGL_*` du langage de commande qui requièrent en général des angles en degrés.*

4 Exemples d'utilisation

Pour différents exemples on se reportera au cas test ZZZZ100A.

4.1 Une formule s'utilise comme une fonction tabulée

Définition de la formule `Sia` :

```
SIa = FORMULE(NOM_PARA='X',VALE='sin(X)')
```

Fonction tabulée équivalente `SI` :

```
LR  = DEFI_LIST_REEL(  DEBUT = 0.,
                  INTERVALLE = _F(  JUSQU_A = pi , PAS = 0.01 )

SI  = CALC_FONC_INTERP( FONCTION  = SIa,
                        LIST_PARA = LR,
                        NOM_PARA  = 'X',
                        NOM_RESU  = 'DEPL',   )
```

Pour définir ainsi une fonction tabulée à partir d'une formule interprétable, voir `CALC_FONC_INTERP` [U4.32.01].

Usage de `SI` ou de `SIa` dans un mot clé simple attendant une fonction ou une formule :

```
champ=CREA_CHAMP( ... AFFE = _F( ... VALE_F  = SI ou SIa,   ) )
```

4.2 Une formule peut être évaluée comme un réel

Dans le corps du fichier de commande :

```
SIa = FORMULE(NOM_PARA='X',VALE='sin(X)')

X = SIa(1.57)
print SIa(1.57)
```

Derrière un mot clé simple attendant un réel :

```
LR = DEFI_LIST_REEL(DEBUT=SIa(0.),
                    INTERVALLE=_F(JUSQU_A=SIa(pi/2.), PAS=0.01))
```

Dans une autre formule :

```
SIb = FORMULE(NOM_PARA='X', VALE='X*SIa(5.)', SIa=SIa )
```

4.3 Invoquer une formule ou une fonction dans une autre formule

```
SIa = FORMULE(NOM_PARA='X',VALE='sin(X)')
```


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

Attention à penser à mettre l'argument (X ici) dans l'appel à la fonction SIa :

```
SIb = FORMULE(NOM_PARA='X', VALE='X*SIa(X)', SIa=SIa)
```

4.4 Formule à plusieurs paramètres

```
NAP = FORMULE(NOM_PARA=('AMOR','FREQ'),
              VALE='''(1./((2.*pi*FREQ)**2 - OMEGA**2 )**2
                      +(2.*AMOR*2.*pi*FREQ*OMEGA)**2)''',
              OMEGA=OMEGA)
```

Dans cet exemple, on définit une formule à 2 paramètres. Compte tenu de la longueur de l'expression, elle est écrite pour plus de commodité sur plusieurs lignes avec des triples quotes pour la délimiter. La constante pi est une constante standard (cf paragraphe [§3.2]), la constante OMEGA doit être fournie explicitement.

Dans l'état actuel, seules les formules de $\mathbb{R}$ dans $\mathbb{R}$ ou $\mathbb{C}$ sont possibles : un seul scalaire produit.

4.5 Formule issue de programmation d'une fonction Python

On peut faire référence dans une formule à des fonctions programmées en Python, ce qui autorise des formules beaucoup plus complexes que de simples expressions algébriques.
Par exemple une fonction de Heavyside :

$$HEAVYSIDE(x)=\begin{cases}0. si\ x<0.\\1. si\ x\geqslant 0.\end{cases}$$

La fonction Python se programme ainsi :

```
def heaviside(x):
    if x < 0.:
        return 0.
    else:
        return 1.

F_HVS = FORMULE(NOM_PARA='INST',
                VALE=' heaviside (INST)',
                heaviside=heaviside)
```

Attention :

L'usage de programmation Python dans le fichier de commandes (ici la méthode heaviside) est incompatible avec l'édition de ce fichier en mode graphique avec AsterStudy.

4.6 Exemple de définition de formules dans une boucle Python

Quand on définit, dans une boucle, des formules dont l'expression dépend de l'indice de la boucle, on voit tout l'intérêt de passer explicitement les constantes.

Exemple :

```
for i in range(3):
    FO[i] = FORMULE(VALE='cos(i*INST)', NOM_PARA='INST', i=i)
    CH[i] = CREA_CHAMP(OPERATION='AFFE', ..., VALE_F=FO[i])
```

Avec ces instructions, on a défini 3 formules qui ont toutes la même expression.

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

À chaque itération, une valeur différente est stockée pour l'argument `i`. Quand la formule est évaluée pour ensuite affecter les valeurs du champ, il n'y a pas d'ambiguïté sur la valeur de `i`. C'est la valeur attachée à la formule lors de sa création, et non pas la valeur actuelle de `i`.
En effet, en mode `PAR_LOT='OUI'`, toutes les commandes sont crées, et ensuite seulement, exécutées. Dans ce mode, `i` vaut 2 lors de l'exécution proprement dite des commandes.

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