

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].

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]
)
```

F est de type `formule` ou `formule_c` .

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 dans le contexte : c'est-à-dire respecter la syntaxe Python et n'utiliser que des fonctions, méthodes ou constantes définies préalablement au moment de son évaluation.

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

Pour une formule représentée par une fonction algébrique ordinaire, se référer à :

« Using Python as a calculator », paragraphe [§3.1.1]

<http://docs.python.org/tut/tut.html>

Outre les signes algébriques ordinaires `+` `-` `/` `*` `**`, sont aussi disponibles des fonctions standards (buildins) : `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

Les fonctions principales 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
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.*

On peut en utiliser d'autres en prenant soin de les importer préalablement à la déclaration de la formule. Exemple de redéfinition de l'exponentielle :

```
from math import e,pow
f_exp = FORMULE(NOM_PARA='X',VALE='pow(e,X)')
```

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 :

```
S1b = FORMULE(NOM_PARA='X',VALE='X*SIa(5.)')
```

4.3 Invoquer une formule ou une fonction dans une autre formule

```
S1a = FORMULE(NOM_PARA='X',VALE='sin(X)')
```

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

```
S1b = FORMULE(NOM_PARA='X',VALE='X*SIa(X)')
```

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) ''' )
```

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 aura été définie plus haut par l'utilisateur.

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 :

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

La fonction Python se programme ainsi :

```
def HEAVYSIDE(x) :  
    if x<0. : return 0.  
    if x>=0. : return 1.  
  
F_HVS = FORMULE( NOM_PARA = 'INST',  
                 VALE      = 'HEAVYSIDE(INST)')
```

Attention :

L'usage de programmation Python dans le fichier de commandes (ici la méthode HEAVYSIDE) est incompatible avec l'édition de ce fichier avec EFICAS.

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, il faut être vigilant et bien avoir compris le sens de `PAR_LOT` dans `DEBUT/POURSUITE` (cf. [U1.03.01]).

Exemple :

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

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

En `PAR_LOT='OUI'` (le jeu de données est entièrement construit puis exécuté), lors de l'appel à `CREA_CHAMP` (que ce soit le premier ou le dernier), `i` vaut 2 (dernière valeur en sortant de la boucle), donc la formule évaluée en fonction de `INST` est `'cos(2*INST)'`.

Alors qu'en `PAR_LOT='NON'`, le premier `CREA_CHAMP` est exécuté dès qu'il est construit, donc la formule est évaluée avec `i=0`. Pour le second, `i=1`, pour le troisième, `i=2`.

En fait, le jeu de données utilisé est celui-ci :

```
i = 0  
FO[0] = FORMULE (VALE='cos(i*INST)', NOM_PARA='INST')  
CH[0] = CREA_CHAMP (OPERATION='AFFE', ..., VALE_F=FO[0])  
i = 1  
FO[1] = FORMULE (VALE='cos(i*INST)', NOM_PARA='INST')  
CH[1] = CREA_CHAMP (OPERATION='AFFE', ..., VALE_F=FO[1])  
i = 2  
FO[2] = FORMULE (VALE='cos(i*INST)', NOM_PARA='INST')  
CH[2] = CREA_CHAMP (OPERATION='AFFE', ..., VALE_F=FO[2])
```

On voit bien que l'indice de la boucle n'intervient pas dans l'expression de la formule.

Dans ce cas, ce que souhaite l'utilisateur serait obtenu en faisant :

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

On définit ainsi 3 formules dont les expressions sont `'cos(0*INST)'`, `'cos(1*INST)'` et `'cos(2*INST)'`. De cette manière, qu'il soit exécuté en `PAR_LOT='OUI'` ou `'NON'`, le résultat sera le même.