Operator FORMULATES

1 Goal

Defining a formula in actual value or complex from his mathematical statement.

The formula will be usable in a further order like argument of type function/formula or evaluated with particular values of the variables.

In many applications, one by the command can tabuler this formula for particular values CALC_FONC_INTERP [U4.32.01] which produces a concept of type function or fonction_c like DEFI_FONCTION [U4.31.02] or DEFI_NAPPE [U4.31.03].

2 Syntax

```
F = FORMULA    (

♦NOM_PARA=nom           of the parameters                          [l_K8]
♦/VALE=                 """ definition of the real formula """     [K]
   /VALE_C          =   """ definition of the formula complexes """   [K]
                  )
```

F is of type formulates or formule_c .

3 Operands

3.1 Definition of the function

the body of the function is an algebraical expression Python represented by a character string. It must be appraisable in the context: i.e. to respect syntax Python and to use only functions, methods or constants defined prior to moment of its evaluating.

If VALE is used, the produced formula is with actual value (concept of the type formulates). If VALE_C is used, the formula is with complex value (concept of the formule_c type).
In both cases, the parameters are real. The names of the parameters necessary to the evaluating of the formula are provided behind the key word NOM_PARA.

In the event of error of syntax, it is the language Python which transmits the error message and not *Code_Aster* itself.

Notice

the order of the parameters (key word NOM_PARA) is important. If one creates a formula with two parameters in order to produce a three-dimensions function, the first parameter is the parameter of the three-dimensions function, the second is the parameter of the functions composing the three-dimensions function.

3.2 Functions standards

For a formula represented by an ordinary algebraic function, to refer to:

"Using Python ace has calculator", paragraph [§3.1.1]
http://docs.python.org/tut/tut.html

In addition to the ordinary algebraical signs + -/***, functions standards (buildins) are also available:
min, max, ab, float…

Attention, the sign of division indicates real division here:
1/2 = 0.5
If one wishes to make a whole division operation, should be used the //operator:
1 //2 = 0

3.3 mathematical Functions

the principal functions of the modulus maths of Python are imported by default. They are thus directly usable in the body of the formulas.

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

```
sin          sinh
cos          cosh
tan          tanh
atan         sqrt
atan2        log
asin         log10
acos         exp
```

Moreover, constant pi, of the same modulus, is also available.

Caution:

The goniometrical functions are thus those of Python and expect angles expressed in radians. It is necessary to be vigilant on coherence with the keywords simple ANGL_* of the process control language which require angles in degrees in general.

One can use others of them by taking care to import them prior to the declaration of the formula. Example of redefinition of the exponential one:

```
from math importation E, pow
f_exp = FORMULA (NOM_PARA=' X', VALE=' pow (E, X) ")
```

4 Examples D" use

For various examples one will refer to the case test ZZZZ100A.

4.1 A formula is used like a tabulated function

Definition of the Sia formula :

```
SIa = FORMULA (NOM_PARA=' X', VALE=' sin (X) ")
```

equivalent Tabulated function IF :

```
LR  = DEFI_LIST_REEL (  DEBUT = 0. ,
                  INTERVALLE = _F (  JUSQU_A = pi, NOT = 0.01)

IF  = CALC_FONC_INTERP (FONCTION  = SIa,
                        LIST_PARA = LR,
                        NOM_PARA  = "X",
                        NOM_RESU  = "DEPL",   )
```

to thus define a tabulated function from an interpretable formula, to see CALC_FONC_INTERP [U4.32.01].

Use of IF or of SIa in a simple key word expecting a function or a formula:

```
champ=CRÉA_CHAMP (… AFFE = _F (… VALE_F  = IF or SIa,   ))
```

4.2 a formula can be evaluated like a reality

In the body of the command file:

```
SIa = FORMULA (NOM_PARA=' X", VALE=' sin (X) ")

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

Behind a simple key word expecting a reality:

```
LR = DEFI_LIST_REEL (debut = SIa (0.),
                     INTERVALLE = _F (JUSQU_A = SIa (pi/2.), NOT = 0.01))
```

In another formula:

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

4.3 To call upon a formula or a function in another SIa

```
formula = FORMULA (NOM_PARA=' X", VALE=' sin (X) ")
```

Attention to think of putting L" argument (X here) in the call to the SIa function :

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

4.4 Formulates with several parameters

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

In this example, one defines a formula in 2 parameters. Taking into account the length of L" statement, it is written for more convenience on several lines with triple quotes to delimit it. Constant pi is constant a standard (cf paragraph [§3.2]), the OMEGA constant will have been higher defined by the user.

In the actual position, only the formulas of $\mathbb{R}$ in $\mathbb{R}$ or $\mathbb{C}$ are possible: only one produced scalar.

4.5 Formulate resulting from programming of function Python

One can refer in a formula to functions programmed in Python, which authorizes formulas much more complex than of algebraic simple expressions.

For example a function of Heavyside:

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

The function Python is programmed as follows:

```
def HEAVYSIDE (X):
    yew x<0.  : return 0.
    Yew x>=0. : return 1.

F_HVS = FORMULA (NOM_PARA = "INST",
                 VALE     = "HEAVYSIDE (INST)")
```

Attention:

The use of programming Python in the command file (here method HEAVYSIDE) is incompatible with the edition of this file with EFICAS.

4.6 Example of definition of formulas in a loop Python

When one defines, in a loop, formulas whose statement depends on the index of the loop, it is necessary to be vigilant and to have understood well the meaning of PAR_LOT in DEBUT/POURSUITE (cf [U1.03.01]).

Example:

```
for I in arranges (3):
    FO [I] = FORMULA (VALE=' cos (i*INST) ", NOM_PARA=' INST")
    CH [I] = CREA_CHAMP (OPERATION=' AFFE',…, VALE_F=FO [I])
```

With these instructions, one defined 3 formulas which have all the same statement.
In PAR_LOT=' OUI' (the data file entirely is built then carried out), at the time of the call to CREA_CHAMP (that it is the first or the last), I am worth 2 (last value while leaving the loop), therefore the formula evaluated according to INST is "cos (2*INST)".

Whereas in PAR_LOT=' NON', the first CREA_CHAMP is carried out as soon as it is built, therefore the formula is evaluated with i=0. For the second, i=1, for the third, i=2.

In fact, the data file used is this one:

```
I = 0
FO [0] = FORMULA (VALE=' cos (i*INST) ", NOM_PARA=' INST")
CH [0] = CREA_CHAMP (OPERATION=' AFFE',…, VALE_F=FO [0])
I = 1
FO [1] = FORMULA (VALE=' cos (i*INST) ", NOM_PARA=' INST")
CH [1] = CREA_CHAMP (OPERATION=' AFFE',…, VALE_F=FO [1])
I = 2
FO [2] = FORMULA (VALE=' cos (i*INST) ", NOM_PARA=' INST")
CH [2] = CREA_CHAMP (OPERATION=' AFFE',…, VALE_F=FO [2])
```

One sees well that the index of buckle does not intervene in the statement of the formula.

In this case, which wishes the user would be obtained while making:

```
for I in arranges (3):
    FO [I] = FORMULA (VALE=' cos (%d*INST) "% I, NOM_PARA=' INST")
    CH [I] = CREA_CHAMP (OPERATION=' AFFE',…, VALE_F=FO [I])
```

One thus defines 3 formulas whose statements are "cos (0*INST)", "cos (1*INST)" and "cos (2*INST)". Of this way, that it is carried out in PAR_LOT=' OUI' or "NON", result will be the same one.