Layout of curves with *Code_Aster*

Summary:

This document explains how, starting from a computation result, one can produce tables or functions, to extract the values from these tables or functions, to handle them, and finally lastly to use the orders of impression to plot curves.

If you have of a function or a table and that you wish simply to represent it in the form of a curve, go directly in the paragraph [§4], and consult documentations of the orders IMPR_FONCTION [U4.33.01] and IMPR_TABLE [U4.91.03].

Notice preliminary

Postprocessing must be carried out in CONTINUATION and not following calculation. Several reasons with that:

1) *in case of error, one does not lose the hours of calculation which made it possible to reach it result,*
2) *one can carry out many postprocessings directly while launching Stanley on the result basis of calculation (see astk [U1.04.00] or STANLEY [U4.81.31]),*
3) *to simplify postprocessing, one can use the opportunities given by Python which require to be in PAR_LOT=' NON' in CONTINUATION what prevents the use of eficas for this kind of postprocessing, whereas it is simpler to use eficas to build the principal data file.*

1 To extract the data using the orders Aster

It is supposed that the user has a computation result got to leave, for example, order CALC_MODES for a calculation of clean modes, STAT_NON_LINE for a nonlinear static calculation, or DYNA_NON_LINE for a nonlinear dynamic calculation…

One has in all the cases a concept result which will be for example of type mode_meca, mode_flamb, dyna_trans, tran_gene, evol_elas, evol_noli, evol_ther, etc according to the order used and which contains fields of values that one wishes to represent in the form of curves.

1.1 To produce a function or a table

Recall

A function is made up of two lists of values, X-coordinates and ordered; the X-coordinates are necessarily monotonous.
A table is not necessarily a standard agglomerate of values in the same way which one reaches via a parameter, "name of column". In the use of the tables which interests us here, one will generally produce columns of real numbers; their variation is unspecified.
For more details on than is a table, one will be able to consult the documentation of IMPR_TABLE [U4.91.03].

The values can be extracted by using the following orders:

- RECU_FONCTION [U4.32.03]: product a function starting from a result, of a field, a table… Example: temporal evolution of a component of a field in a particular point.
- POST_RELEVE_T [U4.81.21]: product a table starting from a result or of a field. One can extract a quantity associated with the components from a field (a component, an invariant…) in certain particular points or along a way not necessarily rectilinear.
- MACR_LIGN_COUPE [U4.81.13]: product a table starting from a result or of a field along a line of cut (straight line made up of regular segments).
- RECU_TABLE [U4.71.02]: product a table starting from the values of one or more parameters of a result. For example: evolution of the parameter of piloting during a calculation. One can also extract a table from some structures of particular data.
- CREA_CHAMP [U4.72.04] : allows to extract a field from a structure of data result. This can be useful when an order cannot treat certain results. One can for example then recover a function via RECU_FONCTION/CHAM_GD.

1.2 To treat storage blocks

A storage block is simply a table of values to NR lines, P columns.

In certain cases, one lays out of one or more files made up each one of one or more storage blocks, separated by lines from text. To build functions starting from such files, one can use LIRE_FONCTION [U4.32.02].

For example, one carries out a parametric study, each calculation produces a table which comes to enrich a file by result; the file can also be built by a single calculation or manually, it does not matter. Such a file could resemble that:

```
WITH PARA=1.23
INST         COOR_X       DY
 1.00000E+00  1.00000E+01 -3.02717E-2
 1.20000E+00  1.00000E+01 -8.14498E-2
 1.40000E+00  1.00000E+01 -7.97278E-2
 1.60000E+00  1.00000E+01 -3.86827E-2
 1.80000E+00  1.00000E+01 -8.48309E-2
 2.00000E+00  1.00000E+01 -9.37561E-2
 2.20000E+00  1.00000E+01  7.18293E-2
 2.40000E+00  1.00000E+01  6.05322E-2

WITH PARA=1.98
INST         COOR_X       COOR_Y       DX           DY
 1.00000E+00  1.00000E+01  0.00000E+00 -3.02717E-2  2.07127E-01
 1.10000E+00  1.00000E+01  0.00000E+00 -8.14498E-2  4.14928E-01
 1.20000E+00  1.00000E+01  0.00000E+00 -7.97278E-2  7.92728E-01
 1.80000E+00  1.00000E+01  0.00000E+00 -7.86827E-2  6.88227E-01
 2.45000E+00  1.00000E+01  0.00000E+00  8.48309E-2  3.43029E-01
```

There are several blocks which inevitably do not have the same number of columns.

Let us suppose that one wants to compare displacement DY obtained with the two values of PARA, one will use for example:

```
fDY1=LIRE_FONCTION (
   TYPE=' FONCTION',
   INDIC_PARA= (1.1,),        # the X-coordinates are taken in block 1, column 1
   INDIC_RESU= (1.3,),        # the ordinates are taken in block 1, column 3
   UNITE=38,
   NOM_PARA=' INST',
   NOM_RESU=' DY',)

fDY2=LIRE_FONCTION (
   TYPE=' FONCTION',
   INDIC_PARA= (2.1,),        # the X-coordinates are taken in block 2, column 1
   INDIC_RESU= (2.5,),        # the X-coordinates are taken in block 2, column 5
   UNITE=38,
   NOM_PARA=' INST',
   NOM_RESU=' DY',)
```

traced classical of two functions with IMPR_FONCTION :

```
IMPR_FONCTION (FORMAT=' XMGRACE',
               UNITE=29,
               COURBE= (_F (FONCTION=fDY1,
                            LEGENDE=' PARA=1.23',),
                        _F (FONCTION=fDY2,
                            LEGENDE=' PARA=1.98',),),
               TITRE=' DY=f (T) ‘,)
```

2 To transform the values of a table or a function into object Python

Notice

The treatments in Python which are used from now force to be in `PAR_LOT=' NON'` *in* `CONTINUATION` *(or* `BEGINNING` *).*

The object is here to recover the values of a table or a function in a Python object to handle them then. Let us note that it is sometimes practical to produce a function starting from the data of a table, by filtering possibly certain lines of the table; it is a use of `RECU_FONCTION` [U4.32.03], that we will not approach here.

On the objects of type function, one has the methods:

1) `Values` to recover the X-coordinates and the ordinates in 2 lists Python of realities.

With the data of the preceding paragraph:

```
>>> lx, ly = fDY2.Valeurs ()
```

One obtains:

```
>>> print lx
[1.0, 1.1,1.2,1.8,2.45]
>>> print ly
[0.207127, 0.414928,0.792728,0.688227,0.343029]
```

1) `Absc` and `Ordo` allow to recover the X-coordinates and the ordinates separately.

```
>>> lx = fDY2.Absc ()
>>> lx = fDY2.Ordo ()
```

One can reach the contents of a cell of a table with [*nom_parameter*, *numéro_ligne*]:

```
>>> print ['DY', 2]
-8.14498E-2
```

One can also transform the object (`JEVEUX`) table in an authority of the class Python Counts.

```
>>> tabpy = tab.EXTR_TABLE ()
```

The document [U1.03.02] details the methods Python available on the objects of the type Counts. For the extraction of the values, one has in particular a method `been worth ()` who turns over a dictionary whose keys are the names of parameters (ex `'DY'`) and values lists of the values of the table.

Caution

The lists Python are indexed of 0 with $n-1$ (for n elements), the equivalent of `['DY', 2]` is thus `tabpy ['DY'] [1]` !

For the layout of curves starting from lists of real Python, to see [§4].

3 To handle the values in Python

One gives here some examples of handling of the values obtained previously in the form of lists or of Numeric tables.

Numeric is a module optional Python (i.e. not included with the distribution of Python provided on www.python.org) but essential to use *Code_Aster*, one can thus make `Numeric importation` on all installations of *Code_Aster*.

3.1 With lists Python

The lists Python are easily easy to handle by using the loops. Let us take the example of the note [U2.51.01] for $-10.<x<10.$ in 100 not:

$$\begin{cases} y=-1.5 & \text{if } \sin(x)<0 \\ y=-5. & \text{if not} \end{cases}$$

One thus seeks to build two lists of realities, lx and ly.

As always in Python, it is possible several to make, more or less simply, more or less elegantly!

```
x0=-10.
pas=20. /100
lx= []
for I in arranges (101):
   lx.append (x0+i*pas)

def F (X):
   yew sin (X) <0.:
      return -1.5
   else:
      return -5.

ly=map (F, lx)
```

or

```
lx= [x0+i*pas for I in arranges (101)]
```

who applies the function f with all the elements of lx

One can plot this curve by using the keywords X-COORDINATE and ORDINATE of IMPR_FONCTION (cf [§4]).

3.2 With Numeric tables

The handling of the data in the form of Numeric tables is simplified by the use of very powerful operations (called *ufunc*) on the whole table (in the following example one uses sin ()). Numeric also manages the tables with several dimensions.

By taking again the preceding example:

```
from Numeric importation *
lx = arrayrange (- 10. , 10.+0.2, 0.2, Float)
ly = array (map (F, lx))
```

or without using F :

```
ly = -1.5*less (sin (lx), 0.) + (- 5.) * (1-them (sin (lx), 0.))
```

Let us note that map () turn over a list and not a Numeric table. The second expression is between 10 and 20 times faster on *very* large tables (10^5 – 10^6 terms), which is however rather seldom the case of the functions or tables resulting from*Aster*.

One can plot this curve by using the keywords X-COORDINATE and ORDINATE of IMPR_FONCTION by taking care to convert Numeric tables into list, for example (cf [§4]):

```
X-COORDINATE = lx.tolist ()
```

4 Examples of use of IMPR_FONCTION/IMPR_TABLE

4.1 Function rebuilt starting from two lists Python

Example where one readjusts the results got to be able to compare them with a reference solution (X-coordinates are reversed, it is necessary to compare the absolute value, [SSLS501a]), one uses the methods allowing to extract the X-coordinates and the ordinates from a function:

```
IMPR_FONCTION (FORMAT=' XMGRACE',
               UNITE=53,
               COURBE=_F (ABSCISSE= [57766.1-x for X in MTAST.Absc ()],
                          ORDONNEE= [ABS (there)     for there in
MTAST.Ordo ()],),
               TITRE=' Curves recalée',
               LEGENDE_X=' P2',
               LEGENDE_Y=' MT',)
```

4.2 Layout of a result according to another

This example is extracted partly from the CAS-test [FORMA03a], it acts of a plate perforated in traction.
After a calculation carried out with STAT_NON_LINE, one wishes to trace the effort resulting from traction according to the average vertical displacement of the upper part of the test-tube.

Details of postprocessing:

```
CONTINUATION ()

SOLNL2=CALC_CHAMP (                          Calculation of the nodal forces
   reuse    = SOLNL2, RESULT = SOLNL2,
   OPTION   = 'FORC_NODA',)

M=DEFI_GROUP (                          Definition of the group of nodes of
   reuse=M,                            examination, the higher line of the plate
   MAILLAGE=M,
   CREA_GROUP_NO=_F (GROUP_MA = 'LFG',
                     NAME     = 'LINE',),
)

tab=POST_RELEVE_T (
   ACTION= (
      _F (ENTITLES   = 'Umoyen',      Statement of average displacement to all them
         RESULT   = SOLNL2,          moments of calculation
         NOM_CHAM   = 'DEPL',
         NOM_CMP    = 'DY',
         TOUT_ORDRE = 'YES',
         GROUP_NO   = 'LINE',
         OPERATION  = 'AVERAGE',
      ),
      _F (ENTITLES   = 'Fresultante',     Statement of the resulting effort
         RESULT   = SOLNL2,
         NOM_CHAM   = 'FORC_NODA',
         TOUT_ORDRE = 'YES',
         GROUP_NO   = 'LINE',
         RESULTANT = 'DY',
         OPERATION  = 'EXTRACTION',),),)
```

```
IMPR_TABLE (TABLE=tab,)
```

Just to locate the name of the parameters

Contents (partial, limited to the first 3 moments) of the table:

ENTITLE	NODE	RESU	NOM_CHAM	NUME_ORDRE	INST	DY	SIYY	QUANTITY
Umoyen	-	SOLNL2	DEPL	1	1.00000E+00	2.38745E-02	-	MOMENT_0
Umoyen	-	SOLNL2	DEPL	1	1.00000E+00	-3.70291E-04	-	MOMENT_1
Umoyen	-	SOLNL2	DEPL	1	1.00000E+00	2.36709E-02	-	MINIMUM
Umoyen	-	SOLNL2	DEPL	1	1.00000E+00	2.40291E-02	-	MAXIMUM
Umoyen	-	SOLNL2	DEPL	1	1.00000E+00	2.40597E-02	-	MOYE_INT
Umoyen	-	SOLNL2	DEPL	1	1.00000E+00	2.36894E-02	-	MOYE_EXT
Umoyen	-	SOLNL2	DEPL	2	1.20000E+00	2.86494E-02	-	MOMENT_0
Umoyen	-	SOLNL2	DEPL	2	1.20000E+00	-4.44349E-04	-	MOMENT_1
Umoyen	-	SOLNL2	DEPL	2	1.20000E+00	2.84050E-02	-	MINIMUM
Umoyen	-	SOLNL2	DEPL	2	1.20000E+00	2.88350E-02	-	MAXIMUM
Umoyen	-	SOLNL2	DEPL	2	1.20000E+00	2.88716E-02	-	MOYE_INT
Umoyen	-	SOLNL2	DEPL	2	1.20000E+00	2.84273E-02	-	MOYE_EXT
Umoyen	-	SOLNL2	DEPL	3	1.40000E+00	3.34244E-02	-	MOMENT_0
Umoyen	-	SOLNL2	DEPL	3	1.40000E+00	-5.18504E-04	-	MOMENT_1
Umoyen	-	SOLNL2	DEPL	3	1.40000E+00	3.31393E-02	-	MINIMUM
Umoyen	-	SOLNL2	DEPL	3	1.40000E+00	3.36410E-02	-	MAXIMUM
Umoyen	-	SOLNL2	DEPL	3	1.40000E+00	3.36837E-02	-	MOYE_INT
Umoyen	-	SOLNL2	DEPL	3	1.40000E+00	3.31652E-02	-	MOYE_EXT
[...]								
Fresultante	-	SOLNL2	FORC_NODA	1	1.00000E+00	2.50000E+03	-	-
Fresultante	-	SOLNL2	FORC_NODA	2	1.20000E+00	3.00000E+03	-	-
Fresultante	-	SOLNL2	FORC_NODA	3	1.40000E+00	3.50000E+03	-	-
[...]								

Filtering of the table to extract some $Dy = f(t)$

```
Dy=RECU_FONCTION (
   TABLE =,
   PARA_X = 'INST',
   PARA_Y = 'DY',
   FILTER = (
      _F (NOM_PARA = 'ENTITLES',
          VALE_K   = 'Umoyen',),
      _F (NOM_PARA = 'QUANTITY',
          VALE_K   = 'MOMENT_0',),),)
```

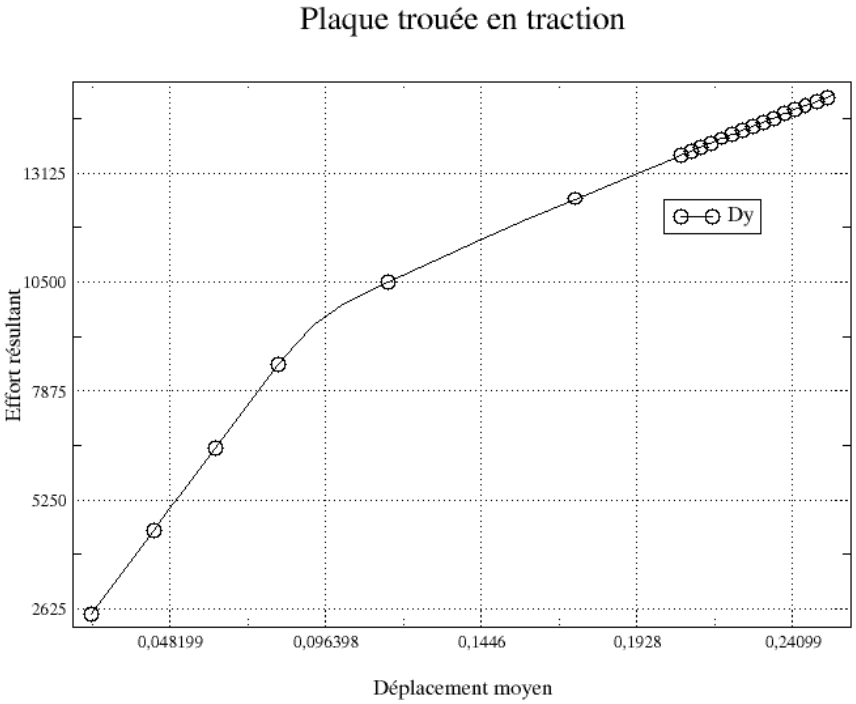
Filtering of the table to extract some $Fy = f(t)$

```
Fy=RECU_FONCTION (
   TABLE =,
   PARA_X = 'INST',
   PARA_Y = 'DY',
   FILTER = (
      _F (NOM_PARA = 'ENTITLES',
          VALE_K   = 'Fresultante',),),)
```

Layout of the ordinates of Fy according to Dy

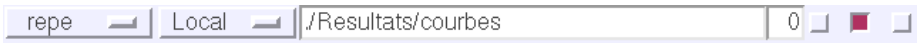
```
IMPR_FONCTION (
   UNIT  = 29,
   FORMAT = 'XMGRACE',
   CURVE = (
      _F (FONC_X = Dy,
          FONC_Y = Fy,),),
   TITLE     = 'Plate perforated in traction',
   LEGENDE_X = 'average Displacement',
   LEGENDE_Y = 'resulting Effort',)
```

What gives us the following curve:



4.3 Layout of a large number of curves

In certain applications, one is brought to plot many curves. Let us suppose that one wishes to compare our results with 50 points of measurement obtained in addition. It would be then tiresome to define 50 files in astk and to use 50 different logical units in the command file!
It is then enough to use the type " repe " as a result in astk (cf [U1.04.00]):



One proceeds then thus in the command file in PAR_LOT=' NON' in CONTINUATION :

```
# definition of the 50 groups of nodes of examination
lgrno = ['point01', 'point02',…, 'point50']

# for each node of examination…
for not in lgrno:
   links = 29
   # files DEPL_point0i.dat will be recopied in Curved results/
     DEFI_FICHIER (UNITE=unit, FICHIER= '. /REPE_OUT/DEPL_ '+point+'
.dat')

   tab=POST_RELEVE_T (
      ACTION=_F (ENTITLES   = 'VonMises',
                 RESULT   = resM,
                 NOM_CHAM   = 'SIEQ_ELNO',
                 NOM_CMP    = 'VMIS',
                 TOUT_ORDRE = 'YES',
                 GROUP_NO   = not,
                 OPERATION  = 'EXTRACTION',),)
```

```
IMPR_TABLE (
   UNIT  = links,
   TABLE  =,
   FORMAT = 'XMGRACE',
   NOM_PARA = ('INST', 'VMIS'),)

# one "releases" the unit to re-use it for the following curve
DEFI_FICHIER (UNITE=unit, ACTION= 'LIBERER')
```

5 Some useful tricks

One proposes here some handling of the data of the recurring tables in Python when from one wants to go further in the generation of curves since *Code_Aster*.

To extract from the table resulting from POST_RELEVE_T the list of the nodes of postprocessing:

When one post-draft a size on a group of nodes (with more than one node) for several moments, the nodes appear for each moment, it is thus necessary to extract the list from these nodes without repetition.

```
tabpy = tab.EXTR_TABLE ()          Creation of the object Counts Python
tno   = tabpy.NOEUD.values ()      One extracts the values from the column NODE
lno   = list (set ([i.strip () for I in tno]))
                                   Powerful method to eliminate them
                                   doubled blooms using set
lno.sort ()                        Sorting by ascending order
```

To build one or more keywords dynamically:

This can in particular be useful to inform the keyword factor FILTER ofIMPR_TABLE according to the context; one builds in this case a dictionary which is then provided in argument of the order.

This:

```
IMPR_TABLE (
   UNIT  = links,
   TABLE  =,
   FORMAT = 'XMGRACE',
   FILTER = (_F (NOM_PARA=' NOEUD',
                 VALE_K = 'N4',),
             _F (NOM_PARA=' NOEUD',
                 VALE_K = 'N4',),),
   NOM_PARA = ('INST', 'VMIS'),
)
```

is equivalent to that:

```
mfac = {'FILTER'    : [{'NOM_PARA': 'NODE',     'VALE_K': 'N4'},
                       {'NOM_PARA': 'ENTITLES', 'VALE_K': 'example'},],
        'NOM_PARA': ['INST', 'VMIS'],
        'UNIT'    : links,
        'FORMAT'    : 'XMGRACE',}

IMPR_TABLE (
   TABLE  =,
   ** mfac
)
```

The interest being of course to be able to build the dictionary as one wishes it.

Notice

The keywords factors (here `FILTER`) can be built while using `_F (mot_clé = value)` , but it is more flexible to see them like a list of dictionaries.