Great principles of operation of *Code_Aster*

Résumé:

One presents here in a summary way the principles of operation of *Code_Aster* and the main rules of use.

This document remains a general description and the player will refer usefully to the other documents, for all the details of use.

1 General principles

Code Aster makes it possible to carry out structural analyses for the phenomena thermal, mechanics, thermomechanical, or thermo-hydro-mechanics coupled, with a or not linear linear behavior, and computations of internal acoustics.

Nonthe linearities relate to the behaviors of the materials (metallurgical plasticity, viscoplasticity, damage, effects, hydration and drying of the concrete,…), large deformation or large rotations and the contact with friction. One will refer to the plate of presentation of *Code_Aster* for the presentation of the various functionalities.

The current industrial studies require the placement of tools of mesh and graphic visualization, which do not form part of the code. However, several tools are usable for these operations via procedures of application interface integrated into the code.

To make a study, the user must, in general, prepare two data files:

- the mesh file **:**

  This file defines geometrical and topological description mesh without choosing, at this stage, the type of formulation of the finite elements used or the physical phenomenon with modelizing. Certain studies can result in using several mesh files.

  This mesh file, in general, is produced by commands integrated into *Code_Aster* starting from a file coming from a software of mesh used out of preprocessor (SALOME, GIBI, GMSH, IDEAS…). Information which this file must contain is specific to *Code_Aster*. They define conventional entities of the finite element method:

  - **nodes** : points defined by **a name** and their **Cartesian coordinates** in space **2D** or **3D**,
  - **meshes** : plane or **voluminal** named topological figures (not, segment, triangle, quadrangle, tetrahedron,…) to which will be able to apply various types of finite elements, boundary conditions or loadings.

  To improve safety of use and comfort of the operations of modelization and examination of the results, one can define, in the mesh file, of the higher levels of entities, having an unspecified property jointly and which could be used directly by their name:

  - **nodes groups** : named lists of names of nodes,
  - **mesh groups** : named lists of names of meshes.

  It will be noted, as of now, that all the handled geometrical entities (nodes, meshes, nodes groups, mesh groups) **are named by the user** and usable constantly by their name (**8 characters to the maximum**). The user will be able to use this possibility to identify explicitly certain parts of studied structure and to thus facilitate the examination of the results. The dialup of the entities is never clarified: it is used for only in-house to point on the values of the various associated variables.

- the command file : to define the text of command which allows:

  - of reading and if required enriching the data of the mesh file (or other sources of offsite results),
  - to affect the data of modelization on the entities of the mesh,
  - to connect various operations of processing: specific computations, postprocessings,
  - to edit the results on various files.

  The text of command refers to the names of geometrical entities defined in the mesh file. It also makes it possible to define new groups constantly.

From the data-processing point of view, these two files are files **ASCII** in free format. One gives the main features here of them:

Syntax of the mesh file :

- length of line limited to 80 characters,
- the allowed characters are:
    - 26 tiny **uppercases** A-Z and 26 **a-z** converted automatically in capital letters, except in the texts (provided between quotes),
    - ten figures **0-9** and the signs of representation of the numbers (**+ -.**),
    - the character **_underscore** usable in key words or names,
- a word must always start with a letter,
- white space is always a separator,
- the character **%** indicates the beginning, until the end of the line, of **a comment**.
- The other rules of reading are specified in the booklet [U3.01.00]

Syntaxe of the command file :

- syntax related to the Python language, making it possible to include instructions of this language
- character **#** indicates the beginning, until the end of the line, of **a comment**.
- The commands must start in column 1, unless they belong to a block indenté (loops, test)

Les other rules of reading are specified in the booklet [U1.03.01].

2 Maillage

2.1 Généralités

the Aster *mesh file* can be written (for really elementary meshes) or modified manually with any text editor. It is a file read in free format, structured in subfile or records by imposed key words.

Various utilities were developed to facilitate the importation of mesh in *Code_Aster*. One distinguishes:

- the utilities of conversion which allow the conversion of a mesh file produced by another software package (IDEAS, GIBI, GMSH…) in a mesh file with the Aster *format,*
- the reading command of a mesh file to format MED, produced by Salome.

2.2 The Aster *mesh file*

the structure and the syntax of the Aster *mesh file* are detailed in **the Booklet [U3.01.00]**.

The Aster *mesh file* is read first line until the first occurrence of a line begin with word **FIN**. **This key word is compulsory.** The mesh file is structured under - independent files starting with **a key word** and is finished by the key word imposed **FINSF**.

This file must comprise at least two subfiles:

- coordinates of all the nodes of the mesh in a cartesian coordinate system 2D (COOR_2D) or 3D (COOR_3D).
- the description of all meshes (TRIA3, HEXA20, etc…), on which one will affect then physical properties, finite elements, boundary conditions or loadings.

It can possibly contain nodes groups (GROUP_NO) or meshes (GROUP_MA) to facilitate the operations of assignment, but also the examination of the results.

> ***It is essential to explicitly create** at this stage the meshes located on the borders of application of the loadings and the boundary conditions. One will find then, in the mesh file:*
> - *edge meshes of elements 2D necessary,*
> - *meshes of face of elements 3D solid masses necessary;*
> - *mesh groups of associated edge and/or face.*

This stress becomes bearable when one uses an application interface, which does the work starting from the indications provided at the time it mesh (see documents `PRE_IDEAS` [U7.01.01] or `PRE_GIBI` [U7.01.11]).

2.3 The utilities of conversion

Ces application interfaces make it possible to convert the files, with or without format, used by various software packages or codes computer, with the conventional format of the Aster *mesh file*.

The currently available interfaces are those which make it possible to use mesh generator IDEAS, mesh generator GIBI of CASTEM 2000 and mesh generator GMSH.

2.3.1 Universal file IDEAS

the convertible file is the universal file defined by documentation I-DEAS (see Fascicule [U3.03.01]). The reconnaissance of version IDEAS used is automatic.

A universal file IDEAS consists of several independent blocks called "**dated sets".** Each "**set dated"** is framed by the character string **-1** and is numbered. "Dated sets" recognized by the application interface are described in the booklet [U3.03.01].

2.3.2 Mesh file GIBI

the interface is produced using command `PRE_GIBI` [U7.01.11]).
The convertible file is file ASCII restored by command `SAUVER FORMAT` of CASTEM 2000. The precise description of the application interface is given in [U3.04.01].

2.3.3 Mesh file GMSH

the interface is produced using command `PRE_GMSH` [U7.01.31]).
The convertible file is file ASCII restored by command `SAVE of GMSH`.

2.4 The mesh file with format MED

the interface is produced using command `LIRE_MAILLAGE (FORMAT='MED')` [U4.21.01].

MED (Modelization and Echanges de Données) is a neutral format of data developed by EDF R & D and French atomic energy agency for the data exchanges between computer codes. Files MED are binary and portable files. The reading of a file MED by `LIRE_MAILLAGE`, makes it possible to recover a mesh produced by any other code able to create a file MED on any other machine. This format of data is in particular used for the file swapping of meshes and results between *Code_Aster* and Salomé or the tools of refinement of mesh HOMARD. The precise description of the application interface is given in [U7.01.21].

2.5 The use of incompatible meshes

Bien que the finite element method recommends the use of regular meshes, without discontinuity, to obtain a correct convergence towards the solution of the continuous problem, it can be necessary for certain modelizations to use incompatible meshes: on both sides of a border, the meshes do not correspond. The connection of this two meshes is then managed on the level of the command file by key word `LIAISON_MAIL` of command `AFFE_CHAR_MECA`. This makes it possible in particular to finely connect an area with a grid with another area where one can be satisfied with a coarse mesh.

2.6 The adaptive mesh

À partir de an initial mesh, it is possible to adapt the mesh, to minimize the made error, using the macro command `MACR_ADAP_MAIL`, which calls upon the software HOMARD. The software of adaptive mesh HOMARD functions on meshes formed by segments, triangles, quadrangles, tetrahedrons, hexahedrons and pentahedrons.

This mesh adaptation is placed after the first computation with *Code_Aster*. An indicator of the error will have been computed. According to its value nets by mesh, the software HOMARD will modify the mesh. It is also possible to interpolate fields of temperature or displacement to the nodes of the old mesh towards the new one [U7.03.01].

3 Commands

3.1 the command file

the command file contains a set of commands, expressed in a language specific to *Code_Aster* (which must respect Python syntax). These commands are analyzed and executed by a software layer of *Code_Aster* called “supervisor”.

3.2 The role of the supervisor

the supervisor carries out various tasks, in particular:

- **a phase of checking** and interpretation of the command file,
- **a stage of execution** of the interpreted commands.

These tasks are detailed in the document [U1.03.01].

The command file is treated starting from the line where is the first call with procedure **`DEBUT ()`** or procedure **`POURSUITE (),`** and until the first occurrence of command **`FIN ()`**. The commands located before **`DEBUT ()`** or **`POURSUITE ()`** and after **`FIN ()`** are not carried out, but must be syntactically correct).

Syntactic phase of checking:

reading and syntactic checking of each command; any error of detected syntax is the subject of a message, but the analysis continues,
checking that all the concepts used as arguments were declared in a preceding command like product concept of an operator; it is also checked that the type of this concept corresponds to the type required for this argument.

Stage of execution:

the supervisor activates successively the various operators and procedures, which carry out the tasks envisaged.

3.3 The principles and the syntax of the process control language

the modular concept of *Code_Aster* makes it possible to present the code like series of instructions independent:

the procedures, which do not produce results directly, but ensure, amongst other things, the management of the exchanges with the offsite files,
the operators, who carry out an operation of computation or data management and produce **a result concept** to which the user gives a name.

These concepts represent data structures, that the user can handle. These **concepts are typified** at the time of their creation and could be used only as argument of input of the corresponding type.

The procedures and the operators thus exchange the necessary information and of the values via **the named concepts.**

The complete syntax of the commands and its implications on the drafting of the command file are detailed in the booklet [U1.03.01]. Here an example of some commands is given (extracted the example with accompanying notes in [U1.05.00]):

```
mall = LIRE_MAILLAGE  ()

mod1 = AFFE_MODELE (MESH = mall,
                   AFFE=_F (TOUT='OUI',
                       PHENOMENE='MECANIQUE',
                       MODELISATION='AXIS'))

f_y = DEFI_FONCTION (NOM_PARA = “Y”
                     VALE =_F (0. , 20000. ,
                              4. , 0. )
                   )

charg = AFFE_CHAR_MECA_F (MODELE = mod1
        PRES_REP =_F (GROUP_MA = (` lfa', ` ldf'),
                   PRES = f_y))
.....
res1 = MECA_STATIQUE (MODELE=mod1,
                      ......
                      EXCIT=_F (LOAD = charg),
                      ….)

res1 = CALC_CHAMP (reuse=res1, RESULTAT=res1,
                    MODELE=mod1,
                    CONTRAINTE= (“SIGM_ELNO”),
                    DEFORMATION= ('EPSI_ELNO),);
```

Some general points will be noted, which one can observe on the preceding example:

- any command starts in first column,
- the list of the operands of a command is obligatorily between brackets, as well as the lists of elements,
- a nom_de_concept can appear only **once** in the text of command like product concept, on the left of the sign $=$,
- **the re-use of an existing concept like product concept**, is not possible that for the operators specified for this purpose. When one uses this possibility (concept reentrant), the command then uses the key word reserved “ reuse”.
    - a command made up of one or more is mot_clé or mot_cle_factor, the latter themselves being composed of a list of mot_clé between brackets and preceded by the prefix _F. In the example suggested, the command AFFE_CHAR_MECA_F uses mot_clé the MODELE and the mot_cle_factor PRES_REP, which is composed of both mot_clé GROUP_MA and PRES.

This operation is done:

- maybe with crushing of the initial values. By way of an example let us announce the factorization in core of a stiffness matrix:
  `masted = FACTORISER (reuse=matass, MATR_ASSE= masted)`
- that is to say with enrichment of the concept.

3.4 Regulate of Une

overload **regulates of overload** usable, in particular for all the operations of assignment, was added to the rules of use of a mot_cle_factor with several lists of operands:

- the assignments are done by superimposing the effects of different mot_clé,
- in the event of conflict, the mot_clé last overrides the precedents.

Example: one wishes to affect various materials MAT1, MAT2 and MAT3 with certain meshes:

```
subdue = AFFE_MATERIAU (MAILLAGE= mon_mail
        AFFE = _F (TOUT = ` OUI', MATER = MAT1),
               _F (GROUP_MA = ` mail2', MATER = MAT2),
               _F (GROUP_MA = ` mail1', MATER = MAT3),
               _F (MESH = (“m7','m8' ), MATER = MAT3))
```

- One starts by assigning material MAT1 to all the meshes.
- One assigns then material MAT2 to the mail2 mesh group which contains, the meshes m8, m9 and m10.
- One assigns finally material MAT3 to the mail1 mesh group (m5, m6 and m7) and to the meshes m7 and m8 , which is source of conflict since the mesh m7 forms already part of the mail1 group. The rule of overload will then be observed and one will obtain finally the following material field:

MAT1	:	meshes m1 m2 m3 m4
MAT2	:	meshes m9 m10
MAT3	:	meshes m5 m6 m7 m8

the progressive effect of the various assignments of material is illustrated in the table below.

Name of the mesh	Material field after the 1st assignment	Material field after the 2nd assignment	Material field after the 3rd assignment	final material field
m1	MAT1	MAT1	MAT1	MAT1
m2	MAT1	MAT1	MAT1	MAT1
m3	MAT1	MAT1	MAT1	MAT1
m4	MAT1	MAT1	MAT1	MAT1
m5	MAT1	MAT1	MAT3	MAT3
m6	MAT1	MAT1	MAT3	MAT3
m7	MAT1	MAT1	MAT3	MAT3
m8	MAT1	MAT2	MAT2	MAT3
m9	MAT1	MAT2	MAT2	MAT2
m10	MAT1	MAT2	MAT2	MAT2

3.5 Règle of remanence

the rule of preceding overload must be supplemented by another rule to specify what occurs when one can affect several quantities for each occurrence of a key word factor.

That is to say for example:

```
CHMEC1=AFFE_CHAR_MECA (  MODELE=MO,
               FORCE_INTERNE= (
            _F (TOUT     = "OUI", FX = 1.         ),
            _F (GROUP_MA = "GM1",          FY = 2.),
        ))
```

The rule of overload says to us that the second occurrence of FORCE_INTERNE overloads the first.

But what is worth FX on a mesh belonging to GM1 ? was it erased by the second occurrence?

If the only rule of overload is observed, FX is not defined on GM1.

The rule of remanence makes it possible to preserve the value of FX.
If the rule of remanence is observed, FX preserves the value affected as a preliminary. All the elements of the model have a value for FX and the elements of GM1 for a value for FX and FY.

3.6 Bases memory associated with a Code_Aster

study rests, for the management of all data structures associated with the various concepts handled, on library JEVEUX. The aforementioned deals with area management **memory** required by the user at the time of the request for execution (Mémoire **parameter** expressed in Mégaoctets). This space is frequently insufficient to store central all data structures. The library takes then loads some, the management of the exchanges between the main memory and the auxiliary storages on files.

Each entity is affected, during its creation by the code, with **a file of direct access**. The aforementioned can be regarded as **a data base**, since it contains, at the end of the execution **the directory** (names and attributes) which makes it possible to exploit all the segments of values that it contains.

Code_Aster uses several data bases:

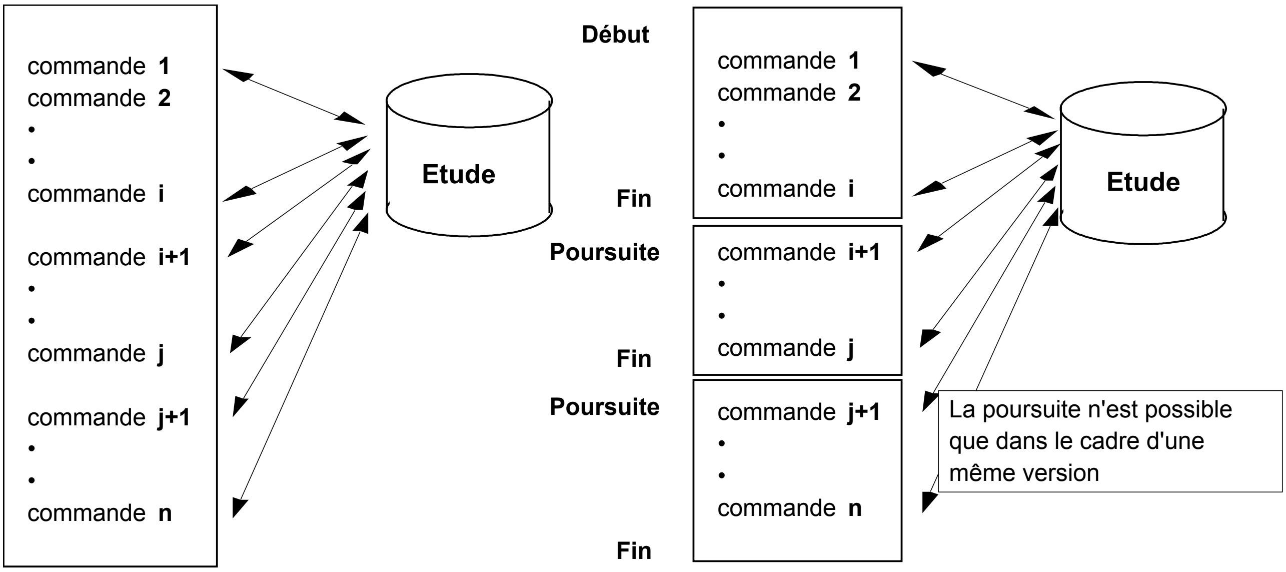
- the data base GLOBALE, which contains all the product concepts by the operators, as well as the contents of certain catalogues on which the concepts are pressed; the file associated with this base allows the later continuation of a study. It must thus be managed by the user.
- the other data bases, used only by Superviseur and the operators, during an execution, do not require a particular intervention of the user.

To make a study, it is to ask for the sequence of several commands:

- procedures to exchange files with the external world,
- of the operators to create product concepts as course of operation of modelization and computation.

The commands which correspond to this sequence of operations can be realized various ways, starting from the single executable modulus of *Code_Aster* :

- in only one sequential execution, without user intervention,
- by splitting the study in several successive executions, with re-use of the former results; starting from the second execution, the access to data base is done in **continuation**; at the time of a continuation, one can ask again the last command, if it stopped prematurely (lack of time, incomplete or incorrect data detected in stage of execution,…).



To manage these possibilities, it will be noted that three commands play a central role. It is those which correspond to **the procedures** which activate the supervisor:

- `DEBUT ()` **compulsory** for **the first** execution of a study,
- `POURSUITE ()` **compulsory** starting from **the second** execution of a study,
- `FIN ()` **compulsory** for all the executions.

For a given study, one can subject command files having following structure:

Note:

- *Command `INCLUDE` makes it possible to include in a flood of commands the contents of another command file. This in particular makes it possible to preserve a file of the main commands readable and to place in annexed files of the bulky digital data (ex: definition of functions).*
- *The command files can be cut out in several files which will be executed one after the other, with intermediate backup of data base. For that, it is necessary to define the successive command files, whose suffix will be: `.com1` , `.com2` ,..., `.com9`. The executions of these files are connected. The data base of the last execution which finished well is preserved.*

3.7 With the definition of the Substitution values

3.7.1 of values It

helps is possible to parameterize a command file.

For example:

```
Eptub = 26.187E-3

Rmoy  = 203.2E-3

Rext  = Rmoy+ (EPtub/2)

will cara  = AFFE_CARA_ELEM (MODELE =  model
         BEAM =_F (  GROUP_MA = all,  SECTION: "CERCLE",
                     CARA    = ("R", "EP"),   VALE  = (Rext, EPtub)))
```

3.7.2 Functions of one or more parameters

It is also often necessary to use quantities functions of other parameters.

Those can be:

- that is to say definite on an offsite file read by command `LIRE_FONCTION`.
- that is to say defined in the command file by:
  - `DEFI_CONSTANTE` which produces a concept `function` with only one constant value,
  - `DEFI_FONCTION` which produces a concept `function` for a quantity function of a real parameter,
  - `DEFI_NAPPE` which produces a concept `function` for a list of functions of the same quantity, each element of the list corresponding to a value of another real parameter.

  The product concept by these operators is of standard `function` and can be used only as argument of operands which accept this type. The operators who accept an argument of type `function` have as a suffix `F` (ex: `AFFE_CHAR_MECA_F`). The functions in this case are defined point by point, with a linear interpolation by default, therefore closely connected by pieces.

  The functions created are discrete arrays of the quantities specified with creation. At the time of an asset tracking, one proceeds according to the specified characteristics, by direct search or interpolation in the array (linear or logarithm). One can specify, during the creation of the function, the prolongation out of field of definition of the array, with various rules, or prohibit it.

- that is to say definite using their analytical statement by the operator `FORMULATES.` For example:

```
Omega   =  3,566  ;

linst = (0. , 0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09,0.10,0.20,0.40)

F = FORMULA (VALE = '''COS (OMEGA*INST)''')
F1=CALC_FONC_INTERP (FONCTION=F, VALE_PARA= linst,
                     NOM_RESU='ACCE”,)
```

the analytical function $F(t)=\cos(\Omega t)$ is then calculated by `CALC_FONC_INTERP` for times of the list `linst` list of times T.

3.8 How to write its command file with EFICAS?

To write a command file of *Code_Aster*, most immediate consists starting from an example already written by others. In particular, all the tests of *Code_Aster* often constitute a starting good base for a new modelization. They are documented as a documentation of validation.

But there is better: tools **EFICAS** make it possible to write in an interactive and user-friendly way its command file, by proposing for each command the list of the possible key words by checking syntax automatically, and by giving access to the documentation of Manuel of use (booklets [U4] and [U7]).

4 The great stages of a study

Les great stages of a study are in the general case:

- the preparation of the work, which finishes after the reading of the mesh,
- the modelization during which are defined and affected all the properties of the finite elements and the materials, the boundary conditions and the loadings,
- computation can then be carried out by the execution of total methods of resolution [U4.5-], which are possibly based on commands of computation and assembly of matrix and vectors [U4.6-]
- the operations of postprocessings complementary to computation [U4.8-],
- operations of printing of the results [U4.9-]
- operations of exchange of results with other software (graphic visualization for example) [U7.05-]

Une autre way of using *Code_Aster* consist in exploiting dedicated tools, available in the code in the form of MACRO_COMMANDES : let us quote for example the dedicated tools:

- ASCOUF (modelization of fissured elbows or elbows with under-thickness),
- ASPIC (modelization of or not fissured fissured bypasses),

4.1 To boot the study and to acquire the mesh

One will not reconsider here the possible fragmentation of the command file, which was presented in a preceding paragraph.

The first executable command is:

```
DEBUT ( )
```

Les argument of this command are useful only for the maintenance actions or case of very large studies.

For the reading of the mesh, coming from a software of external mesh, one can operate in two ways:

- convert the file of a software package into a file with the Code_Aster *format* by an execution separated, which allows, possibly, to modify it by word processing and to preserve it:

  ```
  DEBUT ( )
  PRE_IDEAS ( )
  FIN ( )
  ```

  the normal study will be able to then begin for example by:

  ```
  DEBUT  ( )
  my =   LIRE_MAILLAGE    ( )
  ```

- convert the file right before the lira:

  ```
  DEBUT   ( )
  PRE_IDEAS      ( )
  my =   LIRE_MAILLAGE    ( )
  ```

4.2 Affecter of the data of modelization to the Pour

mesh to build the modelization of a mechanical, thermal or acoustic problem, it is essential to assign to the topological entities mesh:

- a model of finite element,
- properties of the materials (constitutive law and parameters of the model),

- characteristics geometrical or mechanical complementary,
- boundary conditions or loadings.

These assignments are obtained by various operators whose name is prefixed by `AFFE_`. The syntax and the operation of these operators use the facilities brought by the rules already mentioned previously on the use of the key words factor.

4.2.1 Definition of a field of Pour

assignment to carry out an assignment, it is essential to define a field of assignment per reference to the names of the topological entities defined in the file mesh. Five key words are usable for that, according to the specification of the operator:

• refer to all the mesh by	`TOUT= ` OUI'`
• to assign to meshes by	`MAILLE=` (list of names of meshes)
• to assign to mesh groups by	`GROUP_MA=` (list of names of mesh groups)
• to assign to nodes by	`NOEUD=` (list of names of nodes)
• to assign to nodes groups by	`GROUP_NO=` (list of names of nodes groups)

4.2.2 Affecter the type of finite element

Sur the meshes of studied structure, which are at this stage only topological entities, it is essential to affect:

- one or more phenomena studied: `"MECANIQUE", "THERMAL", "ACOUSTICS" ;`
- a model of finite element compatible with the topological description of the mesh. This assignment induces an explicit list of degrees of freedom in each node and a model of interpolation in the element.

One uses for that the operator `AFFE_MODELE` [U4.41.01], who can be called several times on the same mesh. It uses the rule of overload.

Note :

> *For a study with several treated phenomena (` `MECANIQUE'` , ` `THERMAL'`), it is essential to build a model for each phenomenon, by as many calls to `AFFE_MODELE` . On the other hand, for a given computation (mechanical, thermal,…) one needs one and only one model.*

To know the characteristics of the various finite elements available, one will refer to the booklets [U2-], and [U3-].

4.2.3 To affect characteristics of materials

It is necessary to assign to this stage of the characteristics of material, and the associated parameters, with each finite element of the model (except for the discrete elements defined directly by a stiffness matrix, of mass and/or damping). In other words, `DEFI_MATERIAU` is used to define a material and `AFFE_MATERIAU` is used to define a material field by association of the mesh. For a given computation, one needs one and only one material field.

One can also use the validated characteristics of the catalogue material using procedure `INCLUDE_MATERIAU` [U4.43.02].

A certain number of models of behavior are usable: elastic, elastic orthotropic, thermal, acoustic, elastoplastic, elastoviscoplastic, endommagment. Let us note that it is possible to define several characteristics of materials for the same material: elastic and thermal, elastoplastic, thermoplastic,…

4.2.4 Affecter of the characteristics to the Lors

elements of the use of certain types of elements, for the phenomenon ` `MECANIQUE'`, the geometrical definition deducted of the mesh does not make it possible to describe them completely.

One must assign to the meshes the missing characteristics:

- for **the shells** : the constant thickness on each mesh and a reference of reference for the representation of the stress state,
- for **the beams, bars and pipes** : characteristics of the cross section, and possibly directional sense of this section around neutral fiber.

These operations are accessible by the operator `AFFE_CARA_ELEM` [U4.42.01]), who uses, to simplify the drafting of the command, the rule of overload.

Another possibility is offered by this operator: that to introduce, directly in the model, of **the stiffness matrixes** , mass or damping on meshes `POI1` (or nodes) or meshes `SEG2`. These matrixes correspond to the types of discrete finite elements with 3 or 6 degrees of freedom per node `DIS_T` or `DIS_TR` which must be affected at the time of the call to operator `AFFE_MODELE`.

4.2.5 To affect the boundary conditions and the loadings

Ces operations are, in general, essential. They are carried out by several operators whose name is prefixed by `AFFE_CHAR` or `CALC_CHAR`. On the same model, one will be able to carry out several calls to these operators to define, as the study, of the boundary conditions and/or the loadings.

The operators used differ with the phenomenon:

` MECANIQUE'	AFFE_CHAR_CINE	
	AFFE_CHAR_MECAdonnées	of the real type only
	AFFE_CHAR_MECA_Fdonnées	of type THERMAL
function `'	AFFE_CHAR_THERdonnées	of the real type only
	AFFE_CHAR_THER_Fdonnées	of type ACOUSTIC
function `'	AFFE_CHAR_ACOUdonnées	of the real type only

Moreover, one can establish the seismic loading to carry out a computation of response moving relative motion compared to the bearings, using command `CALC_CHAR_SEISME`.

The boundary conditions and loadings can be defined according to their nature:

- with the nodes,
- on meshes of edge (edge or face) or meshes support of finite elements, created in the file mesh. On this meshes operator `AFFE_MODELE` has affected the types of finite elements necessary.

For the detailed definition of the operands of these operators and the rules of directional sense of the meshes support (total reference, local coordinate system or unspecified reference) one will refer to the documents [U4.44.01], [U4.44.02], and [U4.44.04].

The boundary conditions can be dealt with two ways:

- by “elimination” of the degrees of freedom imposed (for linear mechanical models implementing only kinematical boundary conditions (locked degrees of freedom) **without linear relation.** One will define in this case the boundary conditions by command `AFFE_CHAR_CINE`.
- by dualisation [R3.03.01]. This method, thanks to its greater general information, makes it possible to treat all the types of boundary conditions (imposed degree of freedom, linear relations between degrees of freedom,…) ; the method used results in adding 2 LAGRANGE multipliers for each `d. o. f.` imposed or each linear relation.

Each product concept by the call to these operators, of type `AFFE_CHAR`, corresponds to a system of boundary conditions and loadings indissociable. In the commands of computation, one can incorporate these concepts while providing for the operands `CHARGES` a list with concepts of this type.

4.3 Carry out computations by total commands

4.3.1 Thermal analysis

Pour compute it (S) field (S) of temperature corresponding to a or not linear linear thermal analysis:

- **steady** (time 0),
- **evolutionary** times of computation are specified by an as a preliminary definite list of realities

Les commands to be used are:

- `THER_LINEAIRE` for a linear analysis [U4.54.01],
- `THER_NON_LINE` for a nonlinear analysis [U4.54.02],
- `THER_NON_LINE_MO` for a problem of mobile loads in permanent mode [U4.54.03].

Computations of the matrixes and elementary and assembled vectors necessary to the implementation of the methods of resolution are dealt with by these operators.

4.3.2 Static analysis

Pour compute mechanical evolution of a structure subjected to a list of loadings:

- `MECA_STATIQUE` [U4.51.01]: linear behavior, with superposition of the effects of each loading,
- `MACRO_ELAS_MULT` [U4.51.02]: linear behavior, by distinguishing the effects of each loading,
- `STAT_NON_LINE` [U4.51.03]: quasi-static evolution of a structure subjected to a load history in small or great transformations, made of a material whose behavior is linear or not linear, with taking into possible account of the contact and friction.

If this mechanical computation corresponds to a study of **thermoelasticity**, one will refer to one **time** of thermal computation already carried out. If the material were defined with **characteristics depending on the temperature**, those are interpolated for the temperature corresponding to the time of required computation.
For the problems of thermohydromecanic coupled, it is the operator `STAT_NON_LINE` who is used to solve simultaneously the 3 problems thermal, hydraulics and mechanics.
Computations of the matrixes and elementary and assembled vectors necessary to the implementation of the methods of resolution are dealt with by these operators.

4.3.3 Modal analysis

Pour compute eigen modes and eigenvalues of the structure (agent to a vibratory problem or a problem of buckling).

- `MODE_ITER_SIMULT` [U4.52.03]: computation of the eigen modes by simultaneous iterations; the clean eigenvalues and vector are real or complex,
- `MODE_ITER_INV` [U4.52.04]: computation of the eigen modes by inverse iterations; the clean eigenvalues and vector are real or complex,
- `MACRO_MODE_MECA` [U4.52.02]: the modal analysis reduces by cutting out the interval automatically of frequency under intervals,
- `MODE_ITER_CYCL` [U4.52.05]: computation of the eigen modes of a structure with cyclic repetitivity starting from a base of real eigen modes.

These four operators require as a preliminary the computation of the assembled matrixes [U4.61-].

4.3.4 Dynamic analysis

Pour compute the dynamic response, linear or not linear, of structure. Several operators are available. One can quote for example:

`DYNA_LINE_TRAN` [U4.53.02]: temporal dynamic response of a linear structure subjected to a transitory excitation,
`DYNA_LINE_HARM` [U4.53.02]: complex dynamic response of a linear structure subjected to a harmonic excitation,
`DYNA_TRAN_MODAL` [U4.53.21]: response transient dynamics in coordinated generalized by modal recombination.

These three operators require as a preliminary the computation of the assembled matrixes [U4.61-].

DYNA_NON_LINE [U4.53.01]: temporal dynamic response of a nonlinear structure subjected to a transitory excitation, which also computes the assembled matrixes.

4.4 The results

realizing Les results produced by the operators of computations by finite elements [U4.3-], [U4.4-] and [U4.5-] are of two main types:

- either of the cham_no type or cham_elem (by node or element) when it acts operators producing one field (for example RESOUDRE),
- or of the type RESULTAT to be strictly accurate which gathers assemblies of fields.

A field in a concept of the type RESULTAT is identified:

- by a variable of access which can be:
  - a simple sequence number referring to the order in which the fields were arranged,
  - a parameter preset according to the type of the result concept :
    - frequency or number of mode for a RESULTAT of the mode_meca type,
    - time for a RESULTAT of the evol_elas type, temper, dyna_trans or evol_noli.
- by a symbolic name of field referring to the type of the field: generalized displacement, velocity, stress state, forces,…

En plus des variables of access, other parameters can be attached to a kind of result concept .

The various fields are built-in a result concept:

- maybe by the operator who created the concept, a total command (MECA_STATIQUE, STAT_NON_LINE,…) or a simple instruction (MODE_ITER_SIMULT, DYNA_LINE_TRAN,…),
- that is to say during the execution of a command which makes it possible to add a computation option in the form of a field by element or of a field to nodes (CALC_CHAMP); it is then said explicitly that one enriches the concept:

```
resul = operator  (reuse=resul, RESULTAT =  resul…)   ;
```

4.5 To exploit the results

all the preceding commands have building permit various concepts which are exploitable by operators of postprocessing of computations:

- general operators of postprocessing (see booklet [U4.81]), for example CALC_CHAMP, POST_ELEM, POST_RELEVE_T,
- operators of fracture mechanics (see booklet [U4.82]), for example CALC_G,
- metallurgy operator: CALC_META,
- static mechanical postprocessing (see booklet [U4.83]), for example POST_FATIGUE, POST_RCCM,
- dynamic mechanical postprocessing (see booklet [U4.84]), for example POST_DYNA_ALEA, POST_DYNA_MODA_T.

- operators of extraction:
  - of a field in a result concept CREA_CHAMP [U4.72.04],
  - of a field in generalized coordinates for a dynamic computation with modal base RECU_GENE [U4.71.03],
  - of a function of evolution of a component starting from a result concept RECU_FONCTION [U4.32.03],
  - and of restitution of a dynamic response in physical base REST_GENE_PHYS [U4.63.31],

- an operator of postprocessing of functions or three-dimensions functions CALC_FONCTION which allows search of peaks, extremums, linear combinations,... [U4.32.04].

Lastly, two procedures IMPR_RESU [U4.91.01] and IMPR_FONCTION [U4.33.01] allow the printing and possibly the creation of exploitable files by other software packages, in particular of graphic visualization. One will retain in particular graphic visualization by IDEAS, GMSH, or GIBI whatever the tool for mesh used at the beginning.

5 Print files and Code_Aster

error messages writes the relative informations with computation in three files whose meaning is the following one.

File	Contenu
ERREUR	Erreurs met during execution
MESSAGE	Informations on the course of computation. Repetition of the command file, provided and its interpretation by *Aster*. Execution time of each command. Messages "system"
RESULTAT	Uniquement the expressly written results required at the request of the user and the error messages

Of other files are used for the interfaces with the programs of graphic examination.

One distinguishes various types of messages of ERREUR. The transmitted messages will be directed only according to their type:

Code	Type of Fichiers	message of output
F	fatal error message, the execution stops after various printings. The concept created by the current command is lost. Nevertheless, the product concepts previously are validated and GLOBALE data base is saved. It is used in the frame of the serious detection of error whom cannot allow the normal continuation of an ordering of *Code_Aster.*	ERREUR MESSAGE RESULTAT
E	error message, the execution continues a little bit: this kind of message makes it possible to analyze a series of errors before the program stop. *The emission of a message of the <E>* type *is always followed by the emission of a message of the <F>* type. The product concepts are validated and GLOBALE data base is available for a POURSUITE.	ERREUR MESSAGE RESULTAT
S	error message, the concepts created during the execution, including current command, are validated by the supervisor, the execution stops with "clean" closing of GLOBALE data base . It is thus reusable in POURSUITE. This message makes it possible in particular to be secured against a problem of convergence or time during an iterative process.	ERREUR MESSAGE RESULTAT
A	alarm message. The number of alarm messages is restricted automatically with 5 identical **successive** messages. **It is recommended to the users who have messages of the type A "to repair" their command file to make them disappear.**	MESSAGE RESULTAT
I	message of information of supervisor	MESSAGE

NB: The exceptions behave exactly like <S> errors. It is by way of <S> errors adapted to a typical case.