

Structure of Data sd_l_charge

Contents

1	General information.....	3
2	Object SD_L_CHARGES – Old version.....	3
2.1	Tree structure.....	3
2.2	Contents of the objects.....	3
2.2.1	Object .INFC.....	3
2.2.2	Object .LCHA.....	5
2.2.3	Object .FCHA.....	5
3	Object SD_L_CHARGES – New version.....	5
3.1	Principles.....	5
3.1.1	Is a loading, what it?.....	5
3.1.2	Multiplying function.....	5
3.1.3	Definition of the type of load.....	5
3.1.4	Concept of kind of load.....	6
3.1.5	Concept of keyword of load.....	6
3.1.6	Multiplicity of the kinds/keywords.....	6
3.1.7	Concept of object and prefix.....	7
3.2	Contents of the SD and access to information.....	7
3.2.1	Tree structure.....	7
3.2.2	Utility routines.....	8
3.2.3	Checks of the list of the loads.....	8
3.3	Calculation of the loadings.....	8
3.3.1	Principle general.....	8
3.3.2	Routines of calculation of the loadings.....	8
3.3.3	Routines of pre-assembly.....	8
3.4	Addition of a new loading.....	9

1 General information

An object of the type `sd_l_charges` is created and used in the total orders using the loads. He gives information about the loadings used in the order. This SD is useful at the same time in the operators of calculation but also in postprocessing. It is thus created on the total base or bird following the cases:

- For the operators of linear statics (`MECA_STATIQUE`) or `NON_LINEAIRE` (`STAT_NON_LINE`), for thermics (`THER_LINE` , `THER_NON_LINE` and `THER_LINE_MO`) and for dynamics `non_LINEAR` (`DYNA_NON_LINE`), the SD is created on the total basis, it is attached to the SD result produced by these operators;
- For the operators of linear dynamics (`DYNA_VIBRA`), for the breaking process (`CALC_G`), the SD is created on the volatile basis;

In postprocessing, it is either creates (keyword locally `EXCIT`), is recovered in the SD result storing it. It is the case of the operators `CALC_CHAMP` and `POST_ELEM` .

One also creates it in `LIRE_RESU` .

There exist currently two versions of this SD:

- The old version is most widespread;
- The new version, introduced into the operator `DYNA_VIBRA` has the role has to replace the old version in the long term;

2 Object `SD_L_CHARGES` – Old version

2.1 Tree structure

```
sd_l_charges (K19)  :: =record
(O)  \.INFC' : OJB  S  V  I
(O)  \.LCHA' : OJB  S  V  K24
(O)  \.FCHA' : OJB  S  V  K24
```

2.2 Contents of the objects

2.2.1 Object `.INFC`

That is to say `nchar` the number of loads used in the total order (many occurrences of the keyword `EXCIT`)

- dimension = 4 X `nchar` + 7 in mechanics
- dimension = 2 X `nchar` + 1 in thermics

In mechanics

`.INFC` (1) = `nchar`

Values `INFC` (2) with `INFC` (1+`nchar`) are reserved for the loads of Dirichlet type:

For $1 \leq \text{ichar} \leq \text{nchar}$

`INFC` (1+`ichar`) = code

- = 0 if not load of Dirichlet (dualized)
(or if the load contains only eliminated relations)
- = 1 if the load is resulting from `AFFE_CHAR_MECA`
- = 2 if the load is resulting from `AFFE_CHAR_MECA_F` and if it is independent time
- = 3 if the load is resulting from `AFFE_CHAR_MECA_F` and if it is dependent on time
- = 4 following force
- = 5 controlled force resulting from `AFFE_CHAR_MECA`
- = 6 controlled force resulting from `AFFE_CHAR_MECA_F`
- = -1 if the load is resulting from `AFFE_CHAR_CINE`
- = -2 if the load is resulting from `AFFE_CHAR_CINE_F` and if it is independent time

= -3 if the load is resulting from AFFE_CHAR_CINE_F and if it is dependent on time

Values `infc (2+nchar)` with `infc (1+2*nchar)` are reserved for the mechanical loads of Neuman type:

```
For 1 ≤ ichar ≤ nchar
  INFC (1+nchar+ichar)
    = 0 if not load
    = 1 if the load is resulting from AFFE_CHAR_MECA
    = 2 if the load is resulting from AFFE_CHAR_MECA_F and if it is independent of
      time
    = 3 if the load is resulting from AFFE_CHAR_MECA_F and if it is dependent on time
    = 4 if the load is following
    = 5 if the load is controlled
    = 6 if the load is of type ONDE_PLANE
    = 8 if the load is controlled and function
    = 9 if the load is controlled and following
    = 55 if the load is of type PRE_SIGM
    = 10 if the load corresponds to late elements of contact/friction: what makes it possible
to identify it LIGREL . This kind of load is strictly internal with STAT_NON_LINE and is never
defined outwards.
    = 11 if the load is controlled, function and following
    = 20 if the load is of type FORCE_SOL

.INFC (1+2*nchar+1) = unutilised
.INFC (1+2*nchar+2) = number of loads giving of the forces of Laplace.
```

Values `infc (3+3*nchar+1)` with `infc (3+4*nchar)` are reserved for the differential heads (DIDI):

```
For 1 ≤ ichar ≤ nchar
  INFC (3+3*nchar+ichar)
    = 1 if the load is differential (DIDI)
    = 0 if not
```

In thermics

```
.INFC (1) = nchar
```

Values `INFC (2)` with `INFC (1+nchar)` are reserved for the loads of the Dirichlet type:

```
For 1 ≤ ichar ≤ nchar

INFC (1+ichar) = code
  = 0 if not load of Dirichlet (dualized)
      (or if the load contains only eliminated relations)
  = 1 if the load is resulting from AFFE_CHAR_THER
  = 2 if the load is resulting from AFFE_CHAR_THER_F and if it is independent
      time
  = 3 if the load is resulting from AFFE_CHAR_THER_F and if it is dependent on
      time
  = 4 following force
  = -1 if the load is resulting from AFFE_CHAR_CINE
  = -2 if the load is resulting from AFFE_CHAR_CINE_F and if it is independent of
```

= -3 time
if the load is resulting from `AFFE_CHAR_CINE_F` and if it is dependent on
time

Values `INFC (1+nchar+1)` with `INFC (1+2*nchar)` are reserved for the loads of the Neuman type:

For $1 \leq \text{ichar} \leq \text{nchar}$

`INFC (1+nchar+ichar)`
= 0 if not load
= 1 if the load is resulting from `AFFE_CHAR_THER`
= 2 if the load is resulting from `AFFE_CHAR_THER_F` and if it is independent of
time
= 3 if the load is resulting from `AFFE_CHAR_THER_F` and if it is dependent on time

2.2.2 Object `.LCHA`

`.LCHA`: S V K24 dimension = `nchar`
`LCHA` the name of all the loads implied in the total order contains.

2.2.3 Object `.FCHA`

`.FCHA`: S V K24 dimension = `nchar`
`FCHA` the name of the multiplying function applied to the load contains.

3 Object `SD_L_CHARGES` – New version

3.1 Principles

3.1.1 Is a loading, what it?

A loading is in general defined in two times:

- Description of the loading in the operators `AFFE_CHAR_*` ;
- Application of the loading in the order (under the keyword `EXCIT/CHARGE`)

There exists nevertheless a second means of defining a loading: *

- Definition of one `CHAM_NO (VECT_ASSE)` or of one `VECT_ASSE_GENE` using the operators of handling of the vectors (`CALC_VECT_ELEM`, `ASSE_VECTEUR`, etc) or following a preceding calculation;
- Application of the loading in the order (under the keyword `EXCIT/VECT_ASSE`);

This second manner of making is very much used in dynamics.

In the keyword `EXCIT`, one can also define:

- a multiplying function of the time real or complex;
- A kind of application of the loading (m ot key `TYPE_CHARGE`) used especially in the non-linear operators;

3.1.2 Multiplying function

In-house of the orders, one always uses a multiplying function (`FONC_MULT` or `FONC_MULT_C`). If the user did not specify, it is a function constant definite unit in-house, or also a function not-unit with a coefficient given by the user by `COEF_MULT` or `COEF_MULT_C`. In the complex case, it is possible to give the phase (`PHAS_DEG`) and power of the pulsation (`PUIS_PULS`) complex loading, writing in the form of exponential complex.

3.1.3 Definition of the type of load

When the user defines a loading in `AFFE_CHAR_*`, it uses a particular keyword. `AFFE_CHAR_*` proceed then in several different ways:

1. Creation of one `MAP` containing the necessary information on T out the model. For example, if the loading is a pressure distributed, that wants to say that the pressure will be definite not-worthless only on the meshes affected by the loading. On the rest of `MODEL`, the pressure will be worthless¹ ;
2. Creation of one `MAP` on part of the model (`GROUP_MA` defined);
3. Creation of specific objects which are not cards.

Case 1 is most frequent. Case 2 is much rarer: it relates to in mechanics only the loadings of Dirichlet and `FORCE_NODALE`.

Case 3 relates to some loadings (in mechanics: `EVOL_CHAR`, `FORCE_ELEC` and `AFFE_CHAR_CINE` for example).

Information on the type of loading is recoverable by the name of the object (`MAP` or another object) created by `AFFE_CHAR_MECA`. But there are some ambiguous cases (lost information). E N any state of cause, the place of application of the load is lost because, in general, one definite `MAP` on all `MODEL` (case 2).

3.1.4 Concept of kind of load

One **kind** of load gathers the loadings having the following common points:

- Phenomenologic unit: Neumann/Dirichlet on one `PHENOMENON` given;
- Unit of description: the loading is described in the same operator (`AFFE_CHAR_*` for example) and is built in the same way: even object (map) and an associated parameter , one `LIGREL`, an option;
- Unit of programming: loading is calculated in only one routine

There are (currently) fifteen kinds of load:

- `DIRI_DUAL` : `AFFE_CHAR_MECA` with Dirichlet dualized;
- `DIRI_ELIM` : `AFFE_CHAR_CINE` ;
- `NEUM_MECA` : Loading of Neumann in mechanics;
- `PRE_SIGM` : as its name indicates it;
- `VITE_FACE` : as its name indicates it;
- `IMPE_FACE` : as its name indicates it;
- `EVOL_CHAR` : as its name indicates it;
- `SIGM_CABLE` : as its name indicates it;
- `FORCE_ELEC` : as its name indicates it;
- `INTE_ELEC` : as its name indicates it;
- `ONDE_FLUI` : as its name indicates it;
- `ONDE_PLANE` : as its name indicates it;
- `VECT_ASSE_CHAR` : `VECT_ASSE` defined by `AFFE_CHAR_MECA` ;
- `VECT_ASSE` : `CHAMNO` directly as starter of `EXCIT` ;
- `VECT_ASSE_GENE` : `VECT_ASSE_GENE` directly as starter of `EXCIT` ;

A kind gathers several **types** of loading. For example `NEUM_MECA` gather loadings of the Neumann type in mechanics, defined in `AFFE_CHAR_MECA*` and with keywords as various as `FORCE_NODALE` or `FORCE_COQUE`.

3.1.5 Concept of keyword of load

Identifiable by the name of the map or the object (except the ambiguous cases). Is used only in some cases (impressions as debugging, to locate particular loadings like gravity or orders `CALC_G`).

3.1.6 Multiplicity of the kinds/keywords

¹ A worthless pressure and not a not-definite pressure. What implies in practice that a finite element must be able at least to treat the case of a zero load.

In only one `AFFE_CHAR_*`, one can define several very different loadings (several kind and keywords). However the name of the load (concept of `AFFE_CHAR_*` or name of `VECT_ASSE/VECT_ASSE_GENE`) serves itérateurs in `sd_l_charges`.

There are as many loads as of occurrences of the keywords factor `EXCIT`. But there are several loadings by occurrence.

To identify the kind, a coded entirety is thus used and thus 30 different kinds of loads are possible. By decoding of this entirety, one can say if this kind or not is present in the occurrence of the load.

To identify the keyword of load, two coded entireties are thus used and thus 60 different keywords of loads are possible.

3.1.7 Concept of object and prefix

A loading is defined by one or more objects.

The name of the object is always prefixed same manner in the case of `AFFE_CHAR_*`. **prefix** `PREFOB` is built on the following basis:

- `PREFOB (1:8)` : name of the concept resulting from `AFFE_CHAR_*` ;
- `PREFOB (9:13)` : chain identifying the phenomenon is `.CHAC` , `CHME` or `.CHTH` (respectively, acoustics: `AFFE_CHAR_ACOU` , mechanics: `AFFE_CHAR_MECA` , or thermics: `AFFE_CHAR_THER`);

With this prefix:

- The object is identified: a map or another object . `E N` identifying the object, one can identify the kind and, possibly the keyword;
- For the loadings with `LIGREL` reduced, one can build the name of `LIGREL` starting from the prefix;

3.2 Contents of the SD and access to information

3.2.1 Tree structure

```
sd_l_charges (K19)  :: =record
(O)  \.NCHA' : OJB  S  V  K8      LONUTI=nchar
(O)  \.CODC' : OJB  S  V  I      LONUTI=nchar
(O)  \.TYPC' : OJB  S  V  K8      LONUTI=nchar
(O)  \.TYPA' : OJB  S  V  K16     LONUTI=nchar
(O)  \.PREO' : OJB  S  V  K24     LONUTI=nchar
(O)  \.NFON' : OJB  S  V  K8      LONUTI=nchar
(O)  \.TFON' : OJB  S  V  K16     LONUTI=nchar
(O)  \.VFON' : OJB  S  V  R      LONUTI=nchar
```

All these objects are subscripted by the number of load `ichar` , knowing that `nchar` corresponds to the number of occurrences of the keyword factor `EXCIT` .

- `\. NCHA` \ the name of the loads contains (concept resulting from `AFFE_CHAR_*` or `VECT_ASSE`) – Access in reading by `lislch.f` ;
- `\.CODC'` the kind of the loads contains (coded entirety) – Access in reading by `lislco.f` ;
- `\. TYPC` \ the type of the load contains (complex, real, function): `REALITY` , `COMP` , `FONC_F0` (function² unspecified) and `FONC_FT` (function of time) – Access in reading by `lisltc.f` ;
- `\. TYPIFIED` \ the type of application of the load contains (fixed loading, controlled, following, differential Dirichlet): `FIXE_CSTE` , `FIXE_PILO` , `SUIV` and `DIDI` – Access in reading by `lislta.f` ;
- `\. PREO` \ the prefix of the objects of the load – Access in reading contains by `lisl1c.f` ;
- `\. NFON` \ the name of the multiplying function – Access in reading contains by `lisl1nf.f` ;
- `\. TFON` \ the type of the multiplying function contains: function or real or complex constant (`FONCT_REEL` , `FONCT_COMP` , `CONST_REEL` and `CONST_COMP`) – Access in reading by `lisl1tf.f` ;
- `\. VFON` \ contains the parameters of exponential complexes in the case of one complex multiplying function – Access in reading by `readpcp.f` ;

2 The type of the load can be a function defined by `AFFE_CHAR_*_F` But in this case, it can be only a one function with variable *real* .

Warning : The translation process used on this website is a "Machine Translation". It may be imprecise and inaccurate in whole or in part and is provided as a convenience.

Copyright 2019 EDF R&D - Licensed under the terms of the GNU FDL (<http://www.gnu.org/copyleft/fdl.html>)

Important: it is interdict to reach directly the objects of the SD by their name, it is necessary to use the routines of access.

3.2.2 Utility routines

One creates the SD (empty objects) in the routine `liscrs.f`.
One prints his contents (mode `INFO=2`) grace has `lisimp.f`.
The keyword is read `EXCIT` and one fills the SD in `lislec.f`.

3.2.3 Checks of the list of the loads

Some checks are proposed out of standard in the SD. The routine making these checks is `lischk.f`, it is called systematically after the creation and the filling of the SD. These checks are:

- Coherence of the models: all the loads rest on the same model and are coherent with the model calculation (this limitation is provisional, while waiting to think of the case of the transients) – routine `liscom.f` ;
- Coherence enters the loadings and the phenomenon: all the loads rest on the same phenomenon and are coherent with the phenomenon of the operator – routine `lisccp.f` ;
- Coherence enters the loadings and the order: the kind of loading is calculable on the order – routine `lisccm.f` ;
- Prohibition of the doubled blooms: one prohibits that the same loading is present twice – routine `lisdbl.f` ;
- Checks of the types charges. For the moment, not automatic enough, some various checks (piloting, loadings following). In the long term, these incompatibilities will be automated probably – routine `lisver.f` ;

3.3 Calculation of the loadings

3.3.1 Principle general

To calculate a loading indeed (and thus to create the second member to integrate it in a resolution), there are two cases:

1. Standard loadings: they build themselves starting from the assembly in one `CHAM_NO` elementary vectors. There is thus a phase of calculation of these `VECT_ELEM` (call to `CALCULATION`) and an assembly run);
2. The loadings not-standards: `CHAM_NO` object already built is recopied in addition (`VECT_ASSE` for example) or it is a particular loading (contact for example);

3.3.2 Routines of calculation of the loadings

The most important routine is `vechme.f`. Calculate the kind `NEUM_MECA`, `EVOL_CHAR` and other things while glutant sometimes. In the new version, `vechme.f` is replaced by `vechms.f`, and only the kind calculates `NEUM_MECA`, it is separate in two pieces:

- Preparation of the inlet limits (standard with a map of parameter) – Routine `vechmp.f` ;
- Effective calculation of `VECT_ELEM` – Routine `vechmx.f` ;

For the sensitivity: one passes from `vechde.f` with `vechd2.f` (provisional before resorption sensitivity)

For Dirichlet dualized: one passes from `vedime.f` with `vedimd.f`

For `EVOL_CHAR`, one passes from `vechme.f` with `veevoc.f`. In practice one `EVOL_CHAR` contains loadings of the type `NEUM_MECA`. Therefore, for calculation `veevoc.f` will call `vechmp.f/vechmx.f` by building one `sd_l_charges` provisional.

Treatment of `VECT_ASSE/VECT_ASSE_GENE` is envisaged in `cnves1.f`.

Treatment of `VECT_ASSE` coming from `AFFE_CHAR_MECA` (kind `VECT_ASSE_CHAR`) is envisaged in `veassc.f`.

3.3.3 Routines of pre-assembly

Warning : The translation process used on this website is a "Machine Translation". It may be imprecise and inaccurate in whole or in part and is provided as a convenience.

Copyright 2019 EDF R&D - Licensed under the terms of the GNU FDL (<http://www.gnu.org/copyleft/fdl.html>)

As all the loadings are built using a multiplying function, the assembly in two sequences is made:

- Pre-assembly: `VECT_ELEM` are assembled in a list of `CHAM_NO`, in the routine `asvepr.f` who builds a list;
- Linear combination of `CHAM_NO` : `O N` combines them `CHAM_NO` pre-assembled `S` with the multiplying functions in the routine `ascomb.f`;

3.4 Addition of a new loading

To add a new loading, it is necessary:

- To make the impacts necessary in `AFFE_CHAR*` ;
- To identify the kind load by comparison with what already exists;
- To identify the operators allowing the use of this load;
- To write the routines which will calculate this load (§ 3.3.2 and possible elementary calculations) ;
- To impact the principal routine `lisdef.f` (to supplement them `DATED`) ;
- To add protections of use in `lischk.f` (to identify the operators allowing the use of this load for example, to make incompatible with piloting, etc...)