Data format sd_l_charge

Summarized:

Is contents

1 General information

an object of the sd_l_charges type is created and used in the global commands using the loads. He gives information about the loadings used in the command. This SD is useful at the same time in the operators of computation but also in postprocessing. It is thus created on the global database or bird following the cases:

•For the operators of linear static (MECA_STATIQUE) or NON_LINEAIRE (STAT_NON_LINE), for thermal (THER_LINE , THER_NON_LINE and THER_LINE_MO) and for the dynamics non_LINEAIRE (DYNA_NON_LINE), the SD is created on the global database, it is attached to the SD result produced by these operators;

•For the operators of linear dynamics (DYNA_VIBRA), for fracture mechanics (CALC_G), the SD is created on the volatile basis;

In postprocessing, it is either creates locally (keyword EXCIT), or recovered in the SD result storing it. It is the case of 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 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 Contained objects

2.2.1 Object .INFC

Is nchar the number of loads used in the global command (many occurrences of keyword EXCIT)

- dimension = 4 X nchar + 3 in mechanical
- dimension = 2 X nchar + 1 in thermal

In mechanics

.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 of 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 of time
= 3	if the load is resulting from AFFE_CHAR_MECA_F and if it is dependant on times
= 4	follower force
= 5	controlled 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 time
= -3	if the load is resulting from AFFE_CHAR_CINE_F and if it is dependant on time

Note:

** distinction -1, - 2, - 3 is used. One makes use of code < 0 to know if the load is a sd_char_cine.*
** a load can now contain linear relations dualized as well as eliminated relations. When it is the case, its "code" is that of the load dualized (≥ 0)*
*to know if a load contains eliminated relations, it is thus necessary to look at a little further which code < 0. (See for example the routine ascavc.f). **
a load of code 0 can contain eliminated relations.

The 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 of 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 dependant on time .INFC

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

The values infc (3+3*nchar+1) with infc (3+4*nchar) are reserved for the differential heads: For

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

thermal .INFC

(1) = nchar

values INFC (2) with INFC (1+nchar) is reserved for the loads of the Dirichlet type: For

1 ≤ ichar ≤ nchar INFC

(1+ichar) = code =
0 if not of 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 of time =
3 if the load is resulting from AFFE _CHAR_THER_F and if it is dependant on times
=
4 follower 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 time

=
-3 if the load is resulting from AFFE _CHAR_CINE_F and if it is dependant on time
Note:

See
the remarks concerning the mixture dualized relations and relations eliminated in the preceding
paragraph concerning the mechanical loads.

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

1 ≤ ichar ≤ nchar INFC

(1+nchar+ichar) =
0 if not of 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 dependant on time
Object

2.2.2 .LCHA .LCHA

: S V K24 dimension = nchar LCHA
contains the name of all the loads implied in the global command. Object

2.2.3 .FCHA .FCHA

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

3 SD_L_CHARGES – is News version Principles

3.1

3.1.1 a loading, what it?

A loading is in general defined in two times: D
•escription of the loading in operators AFFE _CHAR_*; Application
•of the loading in the command (under keyword EXCIT /CHARGE) There
exists nevertheless a second layer to define a loading: * Definition

•of a CHAM_NO (VECT_ASSE) or one VECT_ASSE_GENE using the operators of handling of vectors (CALC _VECT_ELEM, ASSE _VECTEUR, etc) or following a preceding computation; A
•pplication of the loading in the command (under keyword EXCIT /VECT_ASSE) ; This
second way make is very much used in dynamics. In

keyword EXCIT , one can also define:
•a multiplying function of the time real or complex; U
•N standard of application of the loading (m ot key TYPE_CHARGE) used especially in the nonlinear operators; In-house

3.1.2 multiplying function

of the commands, 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 NON-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 the power of pulsation (PUIS_PULS) of the complex loading, written in the form of exponential complex. Definition

3.1.3 of the type of load When

the user defines a loading in AFFE _CHAR_*, it uses a particular keyword. AFFE _CHAR_* proceeds then in several different ways: Creation
1. of a CARD containing the necessary information on T out the model . For example , if the loading is a distributed pressure, that wants to say that the pressure will be defined non-nulle only on meshes affected by the loading. On the rest of MODELE , the pressure will be null 1Une [1] Creation
2.of a CARD on part of model (GROUP_MA definite); Creation
3.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 recoverable by the name of the object (CARD or another object) is 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 the CARD on all the model (case 2).

3.1.4 The notion of kind of load A kind of

load gathers the loadings having the following common points: Phenomenologic
•unit: Neumann/Dirichlet on a PHENOMENE given ; Unit
•of description: the loading is described in same operator (AFFE _CHAR_* for example) and is built in the same way: even object (card) and an associated parameter, a 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 in entry of EXCIT; VECT_ASSE_GENE
• : VECT_ASSE_GENE directly in entry of EXCIT;

A kind gathers several types ***of*** loading. For example NEUM _MECA gathers loadings of the Neumann type in mechanics, defined in AFFE _CHAR_MECA* and with keywords as various as FORCE_NODALE or FORCE_COQUE .

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

3.1.5 Notion of Identifiable keyword of load

by the name of the card or the object (except the ambiguous cases). Is useful only in some cases (printings of debugging, to locate particular loadings like gravity). Multiplicity

3.1.6 of the kinds/keywords 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 the 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 integer is thus used and thus 30 different kinds of loads are possible. By decoding of this integer, one can say if this kind or not is present in the occurrence of the load. Notion

3.1.7 of object and prefix

a loading is defined by one or more objects.
The name of the object is always prefixed same way 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 , has coustic : AFFE _CHAR_ACOU, m ecanic : AFFE_CHAR_MECA , or T hermic : AFFE_CHAR_THER) ; With

this prefix: O
•N identifies the object: C arte or another object . E N identifying the object, one can identify the kind and , possibly the keyword; P
•our the loadings with reduced LIGREL , one can build the name of the LIGREL from the prefix ; Contents

3.2 of the SD and access to information Tree structure

3.2.1 sd

```
_l_charges (K  19)::   =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
    ) ".  TYPIFIED": 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 " contains N om S charges S (concept resulting from AFFE _CHAR_* or VECT_ASSE) – Access in reading by lislch.f ; ".CODC

•" contains G enre loads (coded integer) – Access in reading by lislco.f ; ". TYPC

•" contains the type of the load (complex, real, function): REEL, COMP , FONC_ F0 (function 2Le standard [2]) and FONC_FT (function of time) – Access in reading by lisltc.f; ". TYPIFIED" contains

•the type of application of the load (fixed loading, controlled, following, Dirichlet differential): FIXE_CSTE, FIXE_PILO , SUIV and DIDI – Access in reading by lislta.f; ". PREO" contains

•the prefix of the objects of the load – Access in reading by lisllc.f; ". NFON " contains

2 the load can be a function defined by AFFE_CHAR _*_F But in this case, it can be only a one function with real variable . *unspecified*

- the name of the multiplying function – Access in reading by lislnf.f; ". TFON" contains
- the type of the multiplying function: function or real or complex constant (FONCT_REEL, FONCT_ COMP, CONST _ REEL and CONST _COMP) – Access in reading by lisltf.f; ". VFON" contain
- the p arameters of exponential complexes a multiplying function in the case of complexes – Access in reading by lily pcp .f; Important : it is prohibited

to reach directly the objects of the SD by their name, it is necessary to use the routines of access. Utility routines One

3.2.2 creates the SD (empty

objects) in the routine liscrs.f. One prints his contents (mode INFO=2) grace has lisimp.f. One reads keyword EXCIT and one fills the SD in lislec.f . Checks of the list of the loads

3.2.3 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 computation (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 command
- : L E kind of loading is computable S ur the command – routine lisccm.f; Prohibition of duplicates : one prohibits that
- the same loading is present twice – routine lisdbl.f; Checks of the types charges . For time
- , not automatic enough , some various checks (control, loadings following). In the long term, these incompatibilities will be automated probably – routine lisver.f; Computation of the loadings general Principle To compute:

3.3 indeed

3.3.1 a loading (and thus

to create the second member to integrate it in a resolution), there are two cases: Standard loadings: they are built

1. from the assembly in an elementary CHAM_NO of vectors. I L thus has there a phase of computation of these VECT_ELEM (call to CALCUL) and an assembly run); Loadings not - standards: the CHAM_NO is recopied
2. object already in addition built (VECT_ASSE for example) or it is a particular loading (contact for example); The routines of computation of the loadings the

3.3.2 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 calculates only kind NEUM_MECA, it is separate in two pieces: Preparation of the inputted fields (standard with a card

- of parameter) – R outine vechmp.f; C alcul effective of the VECT_ELEM – Routine vechmx.f ; For the sensitivity
- : one passes from vechde.f to vechd2.f (provisional

before resorption sensitivity) For Dirichlet dualized : one passes from vedime.f to vedimd.f For EVOL_CHAR

, one passes from vechme.f to veevoc.f. In practice an EVOL_CHAR contains

loadings of the type NEUM_MECA . Therefore, for computation veevoc.f will call vechmp.f/vechmx.f by building a provisional sd_l_charges. The processing of VECT_ASSE /VECT_ASSE_GENE is envisaged in cnvesl.f. The processing of VECT_ASSE

coming from AFFE_CHAR_MECA (kind VECT_ ASSE_CHAR) is envisaged in veassc.f. The routines the pre one - assembly As all the loadings are built using a multiplying

3.3.3 function, one makes

the assembly in two sequences: P re-assembly: the VECT_ELEM are assembled S in a list of CHAM_NO, the routine
- asvepr .f which builds a list ; Linear combination of the CHAM_NO : O N combines the CHAM_NO pre-assembled S with the functions
- multip L I catrices in al routine ascomb.f; Addition of a new loading to add a new loading , it is necessary : F area impacts necessary

3.4 in AFFE_CHAR_*

; I dentifier the kind of the load by comparison
- with what already exists; Id entifier operators
- allowing the use of this load; To write the routines which will calculate
- this load (§ 3.3.2 and possible elementary computations)
- ; I mpacter the principal routine lisdef.f (to supplement 8); A jouter of protections of use
- in lischk.f (to identify the operators allowing the use of
- this load for example, to make incompatible with control, etc…)