Mise en œuvre de STAT_NON_LINE et de DYNA_NON_LINE

Résumé :

On décrit ici la mise en œuvre informatique de l'algorithme de résolution des problèmes quasi-statiques non linéaires et dynamiques non-linéaires. Les documents décrivant en détail ces algorithmes sont supposés connus ([R5.03.01] et [R5.05.05]), on n'en rappellera que les principales étapes. On trouvera dans ce document un rappel des notations, l'organigramme simplifié de la routine op0070, permettant de discerner les principales articulations logiques des opérateurs STAT_NON_LINE et DYNA_NON_LINE de *Code_Aster*, l'arbre d'appel, une description des objets informatiques et des principales routines, et quelques pièges à éviter lors du développement dans cet opérateur.

Table des matières

1 Introduction

L'opérateur op0070 est constitué de plusieurs parties :

1. Lecture des données ;
2. Initialisation des données ;
3. Construction et résolution d'un système linéaire ;
4. Mise-à-jour des champs ;
5. Archivage des résultats, post-traitements.

Le tout étant encapsulé dans trois niveaux de boucle : pas de temps, boucles de point fixe (pour le contact) et itérations de Newton avec une gestion de type événementielle.

Ce document propose la description de ces différentes zones de l'opérateur, en insistant particulièrement sur le côté structurant des SD (§2), des événements (§ 3.3) et sur l'algorithme général, boucle par boucle (§ 4).

À noter que pour aider au débogage, il est possible de sauvegarder les champs de déplacements dans un fichier MED, à chaque itération de Newton. Pour cela, il faut surcharger la routine dbgcha en changeant le booléen dbg=.false. en dbg=.true. Les champs de déplacements seront alors sauvegardés dans un fichier au format MED, sur l'unité logique 80 (ne pas oublier d'ajouter un fichier sur cette unité en sortie dans astk).

2 Structures de données

Une SD passe par plusieurs phases :
- Création de la SD ;
- Lecture des informations de l'utilisateur et stockage dans la SD ;
- Initialisations de la SD ;
- Destruction de la SD.
  Les SD ne passent pas nécessairement par toutes ces phases.

2.1 Les types de SD

On introduit trois niveaux de SD :
1. Les concepts provenant d'autres commandes ou produits par la commande elle-même ;
2. Les SD « moyen-niveau » sont des variables Fortran (des tableaux de type simple) disposant de routines d'accès. Dans cette catégorie, on trouvera le vecteur des fonctionnalités activées `list_func_acti` et les variables-chapeaux qui sont des tableaux de chaînes de caractères contenant des objets plus complexes (matrices, vecteurs, etc) ;
3. Les SD « haut-niveau » sont
   1. Soit des objets JEVEUX spécifiques à l'opérateur (avec routines d'accès)
   2. Soit des SD standards (maillage, modèle, …) de Code_Aster qui resteront internes à l'opérateur
   3. Soit des SD décrites à l'aide des types dérivés Fortran ;

2.1.1 SD de type concept

Les concepts proviennent d'un autre opérateur ou sont construits par l'opérateur pour être utilisés dans d'autres commandes. Elles sont construites exclusivement sur des SD standards dans *Code_Aster* (champs, numérotations, etc.). Leur contenu est décrit dans les documentations de type D4. Voici la liste exhaustive de ces SD dans `op0070.F90` :

Description	**Type**	**Nom généralement utilisé**
Maillage	`sd_mailla`	`MAILLA`
Modèle	`sd_modele`	`MODELE`
Résultat	`sd_resultat`	`RESULT`
Caractéristiques élémentaires	`sd_cara_elem`	`CARELE`
Chargements	`sd_l_charges`	`LISCHA`
Champ de matériaux et variables de commande	`sd_cham_mater`	`MATE`
Définition des contacts	`sd_contact`	`DEFICO`
Définition des liaisons unilatérales	`sd_contact`	`DEFICU`

2.1.2 SD de niveau moyen

Les SD de niveau moyen sont intermédiaires entre les SD « bas-niveau » et les SD « haut-niveau ». En effet, ces variables sont des objets Fortran simples mais disposent de routines d'accès. Elles sont donc « bas-niveau » du point de vue du stockage, mais leur accès se fait par des routines spécifiques, tout comme les SD de « haut-niveau ».

Description	Type	Nom
Fonctionnalités activées	INTEGER(100)	list_func_acti
Variable-chapeau – Matrices élémentaires	CHAR*19 (ZMEELM)	MEELEM
Variable-chapeau – Matrices assemblées	CHAR*19 (ZMEASS)	MEASSE
Variable-chapeau – Vecteurs élémentaires	CHAR*19 (ZVEELM)	VEELEM
Variable-chapeau – Vecteurs assemblés	CHAR*19 (ZVEASS)	VEASSE
Variable-chapeau – Champs solutions	CHAR*19 (ZSOLAL)	SOLALG
Variable-chapeau – Champs solutions incrémentales	CHAR*19 (ZVALIN)	VALINC

2.1.3 SD de haut-niveau

Toutes ces SD sont construites sur des objets JEVEUX ou des types dérivés et leur accès se fait par l'intermédiaire de routines dédiées (encapsulation des données) quand les SD sont spécifiques. On ne revient pas sur toutes, car ce sont des concepts contenus dans les mot-clefs de l'opérateur ou produits par l'opérateur (§ 2.1.1).

Description	Type	Nom
Carte de comportement	sd_carte	COMPOR
Solveur	sd_solveur	SOLVEU
Matrice de préconditionnement	sd_matr_asse	MAPREC
Matrice de résolution assemblée	sd_matr_asse	MATASS
Carte des critères de convergence pour le comportement	sd_carte	CARCRI
Carte des variables de commande de référence	sd_carte	COMREF
Carte des codes-retours erreur du comportement	sd_carte	CODERE
Numérotation	sd_nume_ddl	NUMDDL
Numérotation fixe (utilisée pour le contact méthode continue)	sd_nume_ddl	NUMFIX
Gestion des impressions	spécifique à op0070	ds_print
Gestion des mesures de temps et des statistiques	spécifique à op0070	ds_measure
Gestion des erreurs de l'algorithme	spécifique à op0070	SDERRO
Gestion de l'archivage et de l'état initial (IN et OUT)	spécifique à op0070	ds_inout
Gestion de la convergence	spécifique à op0070	ds_conv
Gestion du SUIVI_DDL	spécifique à op0070	SDSUIV
Gestion des critères de qualité	spécifique à op0070	SDCRIQ
Gestion du pilotage	spécifique à op0070	SDPILO
Gestion des numérotations	spécifique à op0070	SDNUME
Gestion de la dynamique	spécifique à op0070	SDDYNA
Gestion de la discrétisation temporelle	spécifique à op0070	SDDISC
Gestion des critères de convergence	spécifique à op0070	SDCRIT
Gestion de l' OBSERVATION	spécifique à op0070	SDOBSE
Gestion du post-traitement (MODE_VIBR et CRIT_STAB)	spécifique à op0070	SDPOST
Gestion de l'énergie	spécifique à op0070	SDENER
Paramètres de l'algorithme	spécifique à op0070	ds_algopara

Le reste du document va décrire essentiellement le contenu, l'utilisation et l'accès aux SD spécifiques à op0070.

Certaines SD sont construites à l'aide de type dérivés Fortran. Elles obéissent à des règles communes :

- Le module décrivant ces types est NonLin_Datastructure_Type.F90 ;
- Les types sont préfixés par la chaîne NL_DS_. Par exemple type(NL_DS_Print) pour la gestion des impressions ;
- En général, il n'y a qu'une variable de ce type dans l'opérateur, cette variable est passée en argument dans les sous-routines qui en ont besoin (ce ne sont pas des variables globales) ;
- La gestion de ces SD est standardisée :
  - La création de ces SD est faite dans la routine nmini0.F90 par des routines de la forme Create**Type**DS.F90 avec **Type** le nom du type dérivé ;
  - La lecture des données utilisateurs à partir du fichier de commande est faite dans nmdata.F90, en général dans des routines de la forme Read**Type**.F90 avec **Type** le nom du type dérivé ;
  - L'initialisation des données est faite dans nminit.F90 en général, sauf pour certains types qui ont besoin d'une initialisation plus spécifique (à chaque pas de temps par exemple). Les routines réalisant ces initialisations sont en général de la forme Init**Type**.F90 avec **Type** le nom du type dérivé.

  À terme, l'ensemble des structures de données de l'opérateur sera de cette forme.

2.2 Création des SD

Le tableau suivant résume l'origine de la création des SD.

Description	Type	Création
Maillage	sd_mailla	vient du .comm
Modèle	sd_modele	vient du .comm
Résultat	sd_resultat	nmnoli
Caractéristiques élémentaires	sd_cara_elem	vient du .comm
Carte de comportement	sd_carte	nmdocc
Chargements	sd_l_charges	nmdoch
Matériau codé	sd_cham_mater	vient du .comm
Définition des contacts	sd_contact	vient du .comm
Définition des liaisons unilatérales	sd_contact	vient du .comm
Fonctionnalités activées	INTEGER(100)	op0070
Variable-chapeau – Matrices élémentaires	CHAR*19 (ZMEELM)	op0070
Variable-chapeau – Matrices assemblées	CHAR*19 (ZMEASS)	op0070
Variable-chapeau – Vecteurs élémentaires	CHAR*19 (ZVEELM)	op0070
Variable-chapeau – Vecteurs assemblés	CHAR*19 (ZVEASS)	op0070
Variable-chapeau – Champs solutions	CHAR*19 (ZSOLAL)	op0070
Variable-chapeau – Champs solutions incrémentales	CHAR*19 (ZVALIN)	op0070
Solveur	sd_solveur	nmlect
Matrice de préconditionnement	sd_matr_asse	pendant l'algo
Matrice de résolution assemblée	sd_matr_asse	pendant l'algo
Carte des critères de convergence pour le comportement	sd_carte	nmdocr

Carte des variables de commande de référence	sd_carte	nmvcre
Carte des codes-retours erreur du comportement	sd_carte	pendant l'algo
Numérotation	sd_nume_ddl	nmprof
Numérotation fixe (utilisée pour le contact méthode continue)	sd_nume_ddl	nmpro2
Gestion des impressions	Type dérivé	CreatePrintDS
Gestion des mesures de temps	Type dérivé	nmcrti
Gestion des erreurs de l'algorithme	spécifique	nmcrga
Gestion de l'archivage et de l'état initial (IN et OUT)	Type dérivé	nmetcr ntetcr
Gestion des statistiques	Type dérivé	nmcrst
Gestion de la convergence	Type dérivé	CreateConvDS
Gestion du SUIVI_DDL	spécifique	nmcrdd
Gestion des critères de qualité	spécifique	nmcrer
Gestion du pilotage	spécifique	nmdopi
Gestion des numérotations	spécifique	nmnume
Gestion de la dynamique	spécifique	ndcrdy
Gestion de la discrétisation temporelle	spécifique	nmcrli
Gestion de l' OBSERVATION	spécifique	nmcrob
Gestion du post-traitement (MODE_VIBR et CRIT_STAB)	spécifique	nmdopo
Gestion de l'énergie	Type dérivé	eninit
Résolution du CONTACT	Type dérivé	cfmxsd
Résolution de LIAISON_UNIL	Type dérivé	cfmxsd
Paramètres de l'algorithme	Type dérivé	CreateAlgoParaDS

2.3 Lecture des données utilisateurs

La lecture des données utilisateurs se fait majoritairement sous la routine nmdata, cette routine lit les données utilisateurs et crée éventuellement les SD nécessaires. La routine nmdome lit les caractéristiques données par les mots-clefs MODELE, CHAM_MATER, CARA_ELEM et EXCIT en traitant également le cas des reprises de calcul (reuse) qui utilise les informations stockées dans la SD résultat. Le tableau ci-dessous rassemble toutes les SD qui vont lire directement des informations dans le fichier de commande (et donc utilisant les routines getxxx de la communication entre le Fortran et le superviseur Python).

Description	**Mots-clefs**	**Routine de lecture**
Modèle	MODELE	nmdome
Résultat	*concept créé*	nmnoli
Caractéristiques élémentaires	CARA_ELEM	nmdome
Carte de comportement	COMPORTEMENT	nmdocc
Chargements	EXCIT	nmdome
Matériau codé	CHAM_MATER	nmdome
Définition des contacts	CONTACT	cfmxsd
Définition des liaisons unilatérales	CONTACT	cfmxsd

Gestion de la convergence	CONVERGENCE	ReadConv
Solveur	SOLVEUR	cresol
Carte des critères de convergence pour le comportement	COMPORTEMENT	nmdocr
Gestion des impressions	IMPRESSION	ReadPrint
Gestion de l'archivage et de l'état initial (IN et OUT)	ARCHIVAGE ETAT_INIT	nmetcr nmcrar nmdoet
Gestion du SUIVI_DDL	SUIVI_DDL	nmcrdd
Gestion des critères de qualité	CRIT_QUALITE	nmcrer
Gestion du pilotage	PILOTAGE	nmdopi
Gestion de la dynamique	SCHEMA_TEMPS MASS_DIAG PROJ_MODAL MODE_STAT AMOR_MODAL	ndlect
Gestion de la discrétisation temporelle	DISCRETISATION	nmcrsu
Gestion de l' OBSERVATION	OBSERVATION	nmcrob
Gestion du post-traitement (MODE_VIBR et CRIT_STAB)	CRIT_STAB MODE_VIBR	nmdopo
Gestion de l'énergie	ENERGIE	eninit
Résolution du CONTACT	CONTACT	cfmxsd
Résolution de LIAISON_UNIL	CONTACT	cfmxsd
Paramètres de l'algorithme	METHODE RECH_LINEAIRE	nmdomt nmdomt_ls

2.4 Description des SD

2.4.1 SD de niveau moyen

2.4.1.1 Fonctionnalités activées – list_func_acti

Cette SD permet de savoir quelles fonctionnalités sont actives dans l'algorithme à n'importe quel moment. L'idée principale de cette SD est un dismoi très rustique permettant de répondre à une question simple sur les fonctionnalités actives dans op0070. Par exemple :

- Il y a du contact : isfonc(list_func_acti,'CONTACT') est vrai ;
- La recherche linéaire est activée : isfonc(list_func_acti,'RECH_LINEAIRE') est vrai ;

On ne fera pas la liste des fonctionnalités interrogeables par ce mécanisme dans le présent document car ce vecteur est souvent modifié, il suffit de lire la routine isfonc. Même si cet objet est bas-niveau (simple tableaux d'INTEGER), on doit y accéder par l'intermédiaire de trois routines uniquement :

Opération sur le vecteur des fonctionnalités activées – list_func_acti	**Routine**
Préparation des fonctionnalités activées	nmfonc
Interrogation d'une fonctionnalité activée	isfonc
Règles d'exclusion de certaines fonctionnalités	exfonc

Il est **impératif** de modifier le vecteur list_func_acti uniquement dans la routine nmfonc (appelée dans nminit) et de toujours prévoir la question correspondant à cette fonctionnalité dans isfonc. De même, c'est

ce vecteur qu'on utilisera de manière prioritaire pour tester les compatibilités entre certaines fonctionnalités (routine exfonc, appelée par nmfonc).

2.4.1.2 Variable-chapeau – Matrices élémentaires MEELEM

Cette variable-chapeau contient le nom de toutes les matrices élémentaires utilisables dans op0070. Il s'agit donc d'une liste de SD de type sd_matr_elem (sd_resu_elem). Le code de repérage dans les variables chapeaux MEELEM/MEASSE est le même s'il s'agit des mêmes objets.

Variable-chapeau – Matrices élémentaires MEELEM	Code de repérage
Matrices élémentaires de rigidité	MERIGI
Matrices élémentaires des conditions limites de Dirichlet dualisées	MEDIRI
Matrices élémentaires de masse	MEMASS
Matrices élémentaires d'amortissement	MEAMOR
Matrices élémentaires des chargements suiveurs	MESUIV
Matrices élémentaires des sous-structures (macro-éléments)	MESSTR
Matrices élémentaires de rigidité géométrique	MEGEOM
Matrices élémentaires de contact (méthode CONTINUE et XFEM)	MEELTC
Matrices élémentaires de frottement (méthode CONTINUE et XFEM)	MEELTF

2.4.1.3 Variable-chapeau – Matrices assemblées MEASSE

Cette variable-chapeau contient le nom de toutes les matrices assemblées utilisables dans op0070. Il s'agit donc d'une liste de SD de type sd_matr_asse. Le code de repérage dans les variables chapeaux MEELEM/MEASSE est le même s'il s'agit des mêmes objets.

Variable-chapeau – Matrices assemblées MEASSE	Code de repérage
Matrice assemblée de rigidité	MERIGI
Matrice assemblée de masse	MEMASS
Matrice assemblée d'amortissement	MEAMOR
Matrice assemblée des sous-structures (macro-éléments)	MESSTR

2.4.1.4 Variable-chapeau – Vecteurs élémentaires VEELEM

Cette variable-chapeau contient le nom de tout les vecteurs élémentaires utilisables dans op0070. Il s'agit donc d'une liste de SD de type sd_vect_elem (sd_resu_elem). Le code de repérage dans les variables chapeaux VEELEM/VEASSE est le même s'il s'agit des mêmes objets.

Variable-chapeau – Vecteurs élémentaires VEELEM	Code de repérage
Vecteur élémentaire des forces internes	CNFINT
Vecteur élémentaire des réactions d'appui pour les conditions limites de Dirichlet dualisées	CNDIRI
Vecteur élémentaire des conditions limites de Dirichlet dualisées	CNBUDI
Vecteur élémentaire des forces nodales	CNFNOD
Vecteur élémentaire des conditions limites de Dirichlet données	CNDIDO
Vecteur élémentaire des conditions limites de Dirichlet pilotées	CNDIPI

Vecteur élémentaire des conditions limites de Neumann données	CNFEDO
Vecteur élémentaire des conditions limites de Neumann pilotées	CNFEPI
Vecteur élémentaire des conditions limites de type Laplace	CNLAPL
Vecteur élémentaire des conditions limites de type onde plane	CNONDP
Vecteur élémentaire des conditions limites de Neumann suiveuses et données	CNFSDO
Vecteur élémentaire des conditions limites de type impédance (en prédiction)	CNIMPP
Vecteur élémentaire des conditions limites de Dirichlet différentielles	CNDIDI
Vecteur élémentaire des forces sur les sous-structures	CNSSTF
Vecteur élémentaire des forces de contact (méthode CONTINUE et XFEM)	CNELTC
Vecteur élémentaire des forces de frottement (méthode CONTINUE et XFEM)	CNELTF
Vecteur élémentaire des forces de référence (RESI_REFE_RELA)	CNREFE
Vecteur élémentaire des variables de commande pour l'état initial	CNVCF1
Vecteur élémentaire des variables de commande pour la convergence	CNVCF0
Vecteur élémentaire des conditions limites de type impédance (en correction)	CNIMPC

2.4.1.5 Variable-chapeau – Vecteurs assemblés VEASSE

Cette variable-chapeau contient le nom de tous les vecteurs assemblés utilisables dans op0070. Il s'agit donc d'une liste de SD de type sd_cham_no. Ces vecteurs sont essentiellement utilisés dans la construction des seconds membres et dans l'évaluation de la convergence. Le code de repérage dans les variables chapeaux VEELEM/VEASSE est le même s'il s'agit des mêmes objets.

Variable-chapeau – Vecteurs assemblés VE ASSE	**Code de repérage**
Vecteur assemblé des forces internes	CNFINT
Vecteur assemblé des réactions d'appui pour les conditions limites de Dirichlet dualisées	CNDIRI
Vecteur assemblé des conditions limites de Dirichlet dualisées	CNBUDI
Vecteur assemblé des forces nodales	CNFNOD
Vecteur assemblé des conditions limites de Dirichlet données	CNDIDO
Vecteur assemblé des conditions limites de Dirichlet pilotées	CNDIPI
Vecteur assemblé des conditions limites de Neumann données	CNFEDO
Vecteur assemblé des conditions limites de Neumann pilotées	CNFEPI
Vecteur assemblé des conditions limites de type Laplace	CNLAPL
Vecteur assemblé des conditions limites de type onde plane	CNONDP
Vecteur assemblé des conditions limites de Neumann suiveuses et données	CNFSDO
Vecteur assemblé des conditions limites de type impédance (en prédiction)	CNIMPP
Vecteur assemblé des conditions limites de Dirichlet différentielles	CNDIDI
Vecteur assemblé des forces sur les sous-structures	CNSSTF
Vecteur assemblé des forces de contact (méthode CONTINUE et XFEM)	CNELTC
Vecteur assemblé des forces de frottement (méthode CONTINUE et XFEM)	CNELTF
Vecteur assemblé des forces de référence (RESI_REFE_RELA)	CNREFE
Vecteur assemblé des variables de commande pour l'état initial	CNVCF1

Vecteur assemblé des variables de commande pour la convergence	CNVCF0
Vecteur assemblé des conditions limites de type impédance (en correction)	CNIMPC
Vecteur assemblé des conditions limites de Dirichlet éliminées	CNCINE
Vecteur assemblé des sous-structures	CNSSTR
Vecteur assemblé des forces de frottement (méthode DISCRETE)	CNCTDF
Vecteur assemblé des variables de commande pour le calcul	CNVCPR
Vecteur assemblé des forces dynamiques	CNDYNA
Vecteur assemblé de l'amortissement modal (en prédiction)	CNMODP
Vecteur assemblé de l'amortissement modal (en correction)	CNMODC
Vecteur assemblé des forces de contact (méthode DISCRETE)	CNCTDC
Vecteur assemblé des forces unilatérales (méthode LIAISON_UNIL)	CNUNIL
Vecteur assemblé des forces extérieures	CNFEXT
Vecteur assemblé des conditions limites de type VECT_ISS	CNVISS

2.4.1.6 Variable-chapeau – Vecteurs des solutions SOLALG

Cette variable-chapeau contient le nom des champs aux nœuds qui servent dans l'algorithme pour calculer la solution.

Variable-chapeau – Vecteurs des solutions SOLALG	Code de repérage
Solution en déplacement de l'itération de Newton courante	DDEPLA
Déplacement cumulé depuis le début du pas de temps	DEPDEL
Incrément de déplacement du pas de temps précédent	DEPOLD
Solution en déplacement de la prédiction	DEPPR1
Solution en déplacement de la prédiction (partie pilotée)	DEPPR2
Solution en vitesse de l'itération de Newton courante	DVITLA
Vitesse cumulée depuis le début du pas de temps	VITDEL
Incrément de vitesse du pas de temps précédent	VITOLD
Solution en vitesse de la prédiction	VITPR1
Solution en vitesse de la prédiction (partie pilotée)	VITPR2
Solution en accélération de l'itération de Newton courante	DACCLA
Accélération cumulée depuis le début du pas de temps	ACCDEL
Incrément d'accélération du pas de temps précédent	ACCOLD
Solution en accélération de la prédiction	ACCPR1
Solution en accélération de la prédiction (partie pilotée)	ACCPR2
Solution du système	DEPSO1
Solution du système (partie pilotée)	DEPSO2

2.4.1.7 Variable-chapeau – Solutions incrémentales VALINC

Cette variable-chapeau contient le nom des champs aux nœuds ou des champs de type ELGA qui seront les solutions du problème non-linéaire.

Variable-chapeau – Solutions incrémentales VALINC	Code de repérage
Déplacements au début du pas de temps	DEPMOI
Contraintes au début du pas de temps	SIGMOI
Variables internes au début du pas de temps	VARMOI
Vitesses au début du pas de temps	VITMOI
Accélérations au début du pas de temps	ACCMOI
Variables de commande au début du pas de temps	COMMOI
Variables pour les éléments multifibres au début du pas de temps	STRMOI
Forces extérieures au début du pas de temps (pour le calcul des énergies)	FEXMOI
Forces d'amortissement au début du pas de temps (pour le calcul des énergies)	FAMMOI
Forces de liaison au début du pas de temps (pour le calcul des énergies)	FLIMOI
Forces nodales au début du pas de temps (pour le calcul des énergies)	FNOMOI
Déplacements à la fin du pas de temps	DEPPLU
Contraintes à la fin du pas de temps	SIGPLU
Variables internes à la fin du pas de temps	VARPLU
Vitesses à la fin du pas de temps	VITPLU
Accélérations à la fin du pas de temps	ACCPLU
Variables de commande à la fin du pas de temps	COMPLU
Variables pour les éléments multifibres à la fin du pas de temps	STRPLU
Forces extérieures à la fin du pas de temps (pour le calcul des énergies)	FEXPLU
Forces d'amortissement à la fin du pas de temps (pour le calcul des énergies)	FAMPLU
Forces de liaison à la fin du pas de temps (pour le calcul des énergies)	FLIPLU
Forces nodales à la fin du pas de temps (pour le calcul des énergies)	FNOPLU
Contraintes extrapolées (méthode IMPLEX)	SIGEXT
Déplacements à l'itération de Newton courante (gestion des grandes rotations)	DEPKM1
Vitesses à l'itération de Newton courante (gestion des grandes rotations)	VITKM1
Accélérations à l'itération de Newton courante (gestion des grandes rotations)	ACCKM1
Rotations à l'itération de Newton précédente (gestion des grandes rotations)	ROMK
Rotations à l'itération de Newton courante (gestion des grandes rotations)	ROMKM1

2.4.1.8 Accès aux variables-chapeaux

L'accès aux variables-chapeaux se fait par l'intermédiaire de cinq routines.

Opération sur la variable-chapeau	Routine
Création d'une variable-chapeau	nmcha0
Récupération de l'index où est stocké le nom de la variable dans la variable-chapeau	nmchai
Recopie d'une variable-chapeau	nmchcp
Récupération du nom de la variable dans la variable-chapeau	nmchex
Recopie une variable-chapeau en changeant éventuellement un nom de variable	nmchso
Création des CHAM_NO pour VALINC , SOLAG et VEASSE	nmcrch

La taille de ces SD est indiquée de la même manière que les SD bas-niveau. Néanmoins, il convient de répercuter un changement de taille sur la routine principale nmchai. Si on veut modifier le contenu d'une variable-chapeau (ajouter, supprimer ou modifier le contenu d'une variable-chapeau), il faut :

- Impacter éventuellement la longueur dans op0070, nmini0 (ASSERT de protection) et nmchai ;
- Modifier dans nmchai ;
- Créer la variable-chapeau (voir § 2.2) ;
- Initialiser éventuellement le contenu de la variable-chapeau ;

Ces routines se contentent de gérer les variables-chapeaux en tant que liste de noms, le contenu proprement dit de ces variables-chapeaux ne dépend pas d'elles. Par exemple, la variable-chapeau VEASSE contient des vecteurs assemblés. Aucune des routines du tableau précédent ne s'occupent de gérer la SD CHAM_NO des objets contenus dans la variable-chapeau, juste leur nom. Pour les trois variables-chapeaux définissant des CHAM_NO, les champs sont créés dans la routine nmcrch, en utilisant les informations sur les fonctionnalités actives list_func_acti.

2.4.2 SD de haut-niveau

2.4.2.1 Gestion des impressions – NL_DS_Print

L'impression comporte trois types d'objets :

- Les impressions « standards » à base d'utmess ;
- L'impression du tableau de convergence (dans le fichier .mess et éventuellement en export dans un fichier de type csv) ;
- Les lignes de séparation dans le fichier message.
  La principale difficulté dans la gestion de cette structure de données vient du fait que le tableau de convergence est dynamique car l'affichage des colonnes dépend à la fois des fonctionnalités activées (recherche linéaire, type de résidu, contact, etc.) définies via l'objet list_func_acti (voir §2.4.1.1), mais aussi d'options définies par l'utilisateur (INFO_RESIDU et INFO_TEMPS dans le mot-clef facteur AFFICHAGE mais aussi monitoring de degrés de liberté dans le mot-clef facteur SUIVI_DDL).

Une seule variable ds_print de type NL_DS_Print gère l'ensemble des impressions (sauf les utmess standards). Elle contient différentes informations :

- les informations récupérées dans le mot-clef AFFICHAGE ;
- un drapeau indiquant si on doit faire de 'l'affichage dans le fichier message (évalué à partir du paramètre PAS) ;
- un tableau de convergence ;
- une chaîne de caractères (de la largeur du tableau de convergence) contenant la ligne de séparation (avec des "-") ;

Actuellement, op0070 n'a qu'un tableau de convergence, mais la définition de ce qu'est un tableau sous forme générique (voir §2.4.2.2) permettra à terme d'ajouter d'autres tableaux (comme l'énergie par exemple).

Le type NL_DS_Print a la structure suivante :

Type	Nom	Description
aster_logical	l_print	drapeau pour savoir si on affiche ou pas
type(NL_DS_Table)	table_cvg	tableau de convergence
aster_logical	l_info_resi	paramètre INFO_RESIDU dans AFFICHAGE
aster_logical	l_info_time	paramètre INFO_TEMPS dans AFFICHAGE
aster_logical	l_tcvg_csv	drapeau pour savoir si on sort le tableau de convergence dans un fichier CSV
integer	tcvg_unit	paramètre UNITE dans AFFICHAGE (tableau de convergence dans un fichier CSV)
integer	reac_print	paramètre PAS dans AFFICHAGE

character(len=255)	sep_line	ligne de séparation (---), de la largeur du tableau de convergence

La gestion de la SD se fait en trois temps :

1. Création de la SD dans la routine CreatePrintDS dans nmini0. Cette création comprend en particulier la création de toutes les colonnes possibles dans le tableau de convergence. Leur affichage effectif dépendra de l'état du drapeau de leur activation.
2. Lecture des informations données par l'utilisateur (mot-clef AFFICHAGE). Routine ReadPrint dans nmdata. Les informations ont stockées dans ds_print.
3. Initialisation de la SD. Se fait en deux temps :
   1. Ajout des colonnes pour le SUIVI_DDL dans la routine InitPrint dans nminit. En effet, ces colonnes (dont leur titre) va dépendre de ce qu'a demandé l'utilisateur dans le mot-clef facteur SUIVI_DDL ;
   2. Activation des colonnes suivant les fonctionnalités à chaque pas de temps dans nmimin (routine nmnpas).

Ensuite, au niveau de l'utilisation dans l'algorithmie, elle se fait en deux temps. Le développeur affecte les valeurs des colonnes au bon moment grâce à la routine SetCol. L'algorithme général concatène les informations, crée le tableau de convergence et l'affiche de manière régulière à chaque itération de Newton. Ces routines utilitaires de manipulation de la SD ont en général un identificateur commençant par nmimpX. On n'en fait pas la liste exhaustive.

2.4.2.2 Gestion des tableaux – NL_DS_Table

La structure de données NL_DS_Table est un objet qui a une double-fonction :

- il gère une table (au sens Code_Aster) qui est attachée à la structure de données résultat comme l'est la table d'observation (voir §2.4.2.13) ou des statistiques (voir §2.4.2.3) ;
- il gère un tableau qui sera affichable dans le fichier mess ou exportable au format csv (tableau de convergence ou des énergies par exemple)

Ce tableau est défini par :

- ses colonnes : nombre, identifiant et type (entier, réel, chaîne ou complexe) ;
- ses lignes : pour chaque ligne, la valeur affectée et un indicateur d'affectation ;
- son entête : titre (trois lignes au maximum) ;
- ses éléments graphiques : ligne de séparation (par exemple lors de passage des boucles de points fixes en contact) ;
- ses dimensions : sa largeur totale (qui est une fonction du nombre de colonnes et de la largeur de chaque colonne).

Une des difficultés pour gérer ce tableau est de définir dynamiquement les colonnes (selon les fonctionnalités par exemple.) Pour cela, on gère une double liste :

- la liste exhaustive des colonnes possibles ;
- un indicateur de l'activation d'une colonne.

Le type tableau générique est défini par NL_DS_Table qui a la structure suivante :

Type	Nom	Description
integer	nb_cols	Nombre de colonnes
integer	nb_cols_maxi	Nombre maximum de colonnes d'un tableau
type(NL_DS_Column)	cols(max)	Liste des colonnes
aster_logical	l_cols_acti(max)	Drapeau indiquant que la colonne est active ou pas
integer	width	Largeur totale du tableau
integer	title_height	Hauteur du titre (3 par le tableau de convergence)
character(len=255)	sep_line	Ligne de séparation dans le tableau
aster_logical	l_csv	Drapeau pour dire que ce tableau est aussi imprimé dans un

		fichier externe csv
integer	unit_csv	Unité logique pour l'export au format csv
type(NL_DS_TableIO)	table_io	Structure de données table_container
integer	indx_vale(max)	Table d'indirection entre les colonnes et l'objet s'y rattachant

Les objets table_name, nb_para, list_para et type_para permettent d'utiliser directement les utilitaires de gestion de table. Par exemple :

- call tbajpa(tble%table_name, tble%nb_para, tble%list_para, tble%type_para)
- call tbajli(tble%table_name, tble%nb_para, tble%list_para, …)

Sans avoir besoin de reconstruire les listes de paramètres à chaque fois.

Pour des raisons d'efficacité, le nombre maximum de colonnes n'est pas évalué dynamiquement mais donné par la variable nb_cols_maxi. On l'utilise pour les variables cols(*) et l_cols_acti(*).

Les utilitaires disponibles pour la SD sont les suivants :

- CreateTable.F90 : crée une table dans la SD résultat
- CreateVoidTable.F90 : crée une table vide (initialisation de tous les objets)
- ComputeTableHead.F90 : crée les chaînes permettant d'écrire le titre de la table
- ComputeTableWidth.F90 : calcule la largeur totale de la table (selon les colonnes actives)
- PrepareTableLine.F90 : crée la chaîne correspondant à une ligne (vide) de la table
- PrintTableLine.F90 : crée la chaîne (avec les valeurs et les marques) et l'imprime dans une unité logique
- SetTableColumn.F90 : affecte une colonne dans une table
- SetTablePara.F90 : prépare les objets list_para et type_para

Une colonne est définie par :

- un identifiant sous forme de chaîne ;
- trois chaînes définissant le titre de la colonne (on peut en utiliser 1, 2 ou 3 selon la définition du tableau) ;
- un drapeau pour dire si une valeur est affectée ou pas (permet de ne rien afficher si la valeur n'est pas définie) ;
- quatre drapeaux pour donner le type de valeur contenue dans la colonne: entier, chaîne, complexe ou réel ;
- quatre variables (entier, chaîne, complexe, réel) contenant la valeur à afficher ;
    - la possibilité d'afficher une « marque » à côté d'une valeur dans une colonne. Cette marque sert par exemple à dire sur quel résidu on converge (un « X ») ou si l'on a atteint une borne dans le cas du pilotage. Remarque: la marque n'est autorisée qu'avec des colonnes de type entier ou réel, pas avec des chaînes de caractères.

Une colonne est définie par le type NL_DS_Column qui a la structure suivante :

Type	Nom	Description
aster_logical	l_vale_affe	drapeau pour dire qu'on a affecté une valeur dans la colonne
aster_logical	l_vale_inte	drapeau pour dire que la colonne contient un entier
aster_logical	l_vale_real	drapeau pour dire que la colonne contient un réel
aster_logical	l_vale_cplx	drapeau pour dire que la colonne contient un complexe
aster_logical	l_vale_strg	drapeau pour dire que la colonne contient une chaîne
integer	vale_inte	valeur de la colonne si c'est un entier
real(kind=8)	vale_real	valeur de la colonne si c'est un réel
complex(kind=8)	vale_cplx	valeur de la colonne si c'est un complexe
character(len=24)	vale_strg	valeur de la colonne si c'est une chaîne
character(len=9)	name	nom de la colonne (identificateur unique)

character(len=16)	title(3)	titre de la colonne (jusqu'à trois lignes de titre)
character(len=1)	mark	éventuelle marque à côté de la valeur dans la colonne (X, B, etc.)

2.4.2.3 Gestion des mesures de temps et statistiques – NL_DS_Measure

La structure de données permet de gérer les mesures diverses réalisées lors d'un calcul. Elle stocke les temps mais aussi diverses informations comme le nombre d'itérations de Newton ou le nombre de nœuds en contact. On se base pour cela sur trois structures de données :

- la SD NL_DS_Measure est representée par une seule variable de nom ds_measure. C'est un aggrégateur des différentes mesures ;
- la SD NL_DS_Timer gère les différents timers ;
- la SD NL_DS_Device est l'objet qui réalise les mesures.

1.

La structure de données NL_DS_Timer est un objet assez simple qui permet de gérer les appel aux utilitaires de type uttcpu.F90. Elle a la structure suivante :

Type	Nom	Description
character(len=9)	type	nom du timer (identificateur unique)
character(len=24)	cpu_name	nom pour les utilitaires uttcpu
real(kind=8)	time_init	temps initial sauvegardé

La routine nmtime.F90 gère ces timers (démarrage, arrêt, mesure et ré-initialisation).
Pour lancer un chronomètre :

```
call nmtime(ds_measure, 'Launch', 'Time_Step')
```

Pour l'arrêter (et mesurer) :

```
call nmtime(ds_measure, 'Stop', 'Time_Step')
```

Ensuite, la structure de données NL_DS_Device gère les mesures. Chaque *device* permet de mesurer une opération pendant le calcul. La SD a la structure suivante :

2.4.2.4

Type	Nom	Description
character(len=9)	type	Nom du device (identificateur unique)
character(len=9)	timer_name	Nom du timer attaché au device
real(kind=8)	time_iter	Temps mesuré pour une itération de Newton
real(kind=8)	time_step	Temps mesuré pour un pas de calcul
real(kind=8)	time_comp	Temps mesuré pour tout le calcul
integer	time_indi_step	Index dans le catalogue des messages measure.py pour afficher le temps pour ce device à chaque pas de temps (géré dans nmimpr_mess.F90)
integer	time_indi_comp	Index dans le catalogue des messages measure.py pour afficher le temps pour ce device à la fin du calcul (géré dans nmimpr_mess.F90)
aster_logical	l_count_add	Drapeau pour cumuler le nombre d'occurrences à chaque étape
integer	count_iter	Compteur d'occurrences pour une itération de Newton
integer	count_step	Compteur d'occurrences pour un pas de calcul
integer	count_comp	Compteur d'occurrences pour tout le calcul
integer	count_indi_step	Index dans le catalogue des messages measure.py pour afficher le compteur pour ce device à chaque pas de temps

		(géré dans nmimpr_mess.F90)
integer	count_indi_comp	Index dans le catalogue des messages measure.py pour afficher le compteur pour ce device à la fin du calcul (géré dans nmimpr_mess.F90)

La routine nmrvai.F90 permet de gérer les compteurs.
Pour incrémenter un compteur (nombre de pas de temps par exemple) :

```
call nmrinc(ds_measure, 'Time_Step')
```

Pour donner la valeur d'un compteur :

```
call nmrvai(ds_measure, 'Contact_NumbCont', input_count = nbliac)
```

En fait nmrinc.F90 c'est nmrvai.F90 avec input_count = 1.

Enfin, la structure de données NL_DS_Measure gère l'ensemble des SD permettant les mesures (temps et statistiques diverses). Elle a la structure suivante :

Type	Nom	Description
aster_logical	l_table	.true. quand l'utilisateur a écrit TABLE='OUI' dans le mot-clef MESURE
type(NL_DS_Table)	table	Table en sortie pour les statistiques
integer	nb_device	Nombre de device utilisés
integer	nb_device_maxi	Nombre de device maximum
type(NL_DS_Device)	device(max_{device})	Liste des device
aster_logical	l_device_acti(max_{device})	Liste des device actifs
integer	indx_cols(2*max_{device})	Référence des colonnes de la table vers les device
integer	nb_timer	Nombre de timer utilisés
integer	nb_timer_maxi	Nombre de timer maximum
type(NL_DS_Timer)	timer(2*max_{timer})	Liste des timer
real(kind=8)	store_mean_time	Temps moyen pour l'archivage
real(kind=8)	iter_mean_time	Temps moyen par itération de Newton
real(kind=8)	step_mean_time	Temps moyen par pas de temps
real(kind=8)	iter_remain_time	Temps restant pour l'itération
real(kind=8)	step_remain_time	Temps restant pour le pas de temps

L'ensemble des données de cette structure est créé dans la routine principale nmcrti.F90. Cette routine utilise la routine ActivateDevice.F90 qui permet d'activer un device selon la disponibilité de certaines fonctionnalités.

2.4.2.5 Gestion du calcul des énergies – NL_DS_Energy

Pour mesurer les énergies, on utilise une structure de données de type NL_DS_Energy qui a la structure suivante :

Type	Nom	Description
aster_logical	l_comp	.true. quand on veut mesurer les énergies
type(NL_DS_Table)	table	table en sortie pour les énergies (voir §16)
character(len=16)	command	Commande appelante

2.4.2.6 Gestion des erreurs de l'algorithme – SDERRO

Cette SD s'occupe de gérer les erreurs de l'algorithme. Elle contient sept objets de même taille, celui du nombre d'événements (variable ZEVEN) traitables par l'algorithme (modifiable dans nmcrga , cf § 3.4.1)

SDERRO – Objets	
Nom	**Description**
SDERRO(1:19)//'.ENOM'	Nom de l'événement
SDERRO(1:19)//'.ECOV'	Valeur du code-retour lié à l'événement
SDERRO(1:19)//'.ECON'	Nom du code-retour lié à l'événement
SDERRO(1:19)//'.ENIV'	Type et niveau de déclenchement de l'événement
SDERRO(1:19)//'.EFCT'	Fonctionnalité activant un événement de type convergence ou divergence
SDERRO(1:19)//'.EACT'	État de l'événement (activé ou non)
SDERRO(1:19)//'.EMSG'	Code du message à afficher quand l'événement se déclenche

Les deux autres objets permettent de gérer l'état de la boucle et de stocker le dernier événement déclenché (servira aux affichages).

SDERRO – Objets	
Nom	**Description**
SDERRO(1:19)//'.CONV'	État de la boucle
SDERRO(1:19)//'.EEVT'	Information sur le dernier événement déclenché

La SD ERRO s'utilise en utilisant certaines informations contenues dans la SDDISC (§2.4.2.18), provenant des définitions des événements de la commande DEFI_LIST_INST.

Opération sur la gestion des erreurs de l'algorithme – SDERRO	Routine
Création de la SDERRO	nmcrga
Enregistrement d'un événement	nmcrel
Enregistrement d'un événement à partir d'un code-retour	nmcret
Changement de l'état d'une boucle	nmeceb
Lecture de l'état d'une boucle	nmleeb
Remise à zéro des événements	nmeraz
Retourne l'état d'un événement (actif ou non) suivant son nom	nmerge
Émission du message d'information sur l'événement	nmevim
Évaluation de l'état de convergence d'une boucle	nmevcv
Retourne l'état d'un événement (actif ou non) suivant son type	nmltev

L'utilisation de ces routines dans le cadre de la gestion des événements est détaillée dans le § 3.4 .

2.4.2.7 Gestion des entrées/sorties – NL_DS_InOut

On appelle entrées/sorties des opérateurs non-linéaires l'ensemble des opérations liées aux fonctionnalités suivantes :

- Lecture d'un état initial (mot-clef facteur ETAT_INIT) ;
- Extraction des résultats dans une table pendant le calcul (mot-clef facteur OBSERVATION) et monitoring en temps réel dans le tableau de convergence (mot-clef facteur SUIVI_DDL) ;
- Archivage des résultats dans la SD résultat (mot-clef facteur ARCHIVAGE).

Il y a deux types dérivés pour cette fonctionnalité : NL_DS_Inout et NL_DS_Field.

Une seule variable ds_inout de type NL_DS_Inout gère l'ensemble des entrées-sorties. L'objet principal ds_inout contient des paramètres issus de l'utilisateur (mot-clefs ETAT_INIT et ARCHIVAGE) ainsi qu'une liste de champs et leur comportement en entrée/sortie.
Cette SD ne gère pas encore le cas des _paramètres_ des SD résultats (sauf la liste des charges), mais seulement la liste des champs. Elle ne gère pas non plus les utilitaires liés au cadencement de l'archivage (pour l'instant dans la SDDISC voir §2.4.2.18) ni les informations liées à la manière de réaliser une OBSERVATION (pour l'instant dans la SDOBSE voir §2.4.2.13). Le type NL_DS_Inout a la structure suivante :

Type	Nom	Description
character(len=8)	result	nom de la SD résultat pour archiver
integer	nb_field	nombre de champs gérés
integer	nb_field_maxi	nombre maximum de champs gérables
type(NL_DS_Field)	field (nb_field_maxi)	Liste des champs
character(len=8)	stin_evol	nom de la SD résultat dans ETAT_INIT
aster_logical	l_stin_evol	drapeau pour la présence d'une SD résultat dans ETAT_INIT
aster_logical	l_field_acti (nb_field_maxi)	drapeaux des champs actifs (dépend des fonctionnalités)
aster_logical	l_field_read (nb_field_maxi)	drapeaux pour indiquer qu'un champ est à considérer dans l'état initial
aster_logical	l_state_init	drapeau pour indiquer la présence d'un état initial
aster_logical	l_reuse	drapeau pour indiquer qu'on est en mode reuse
integer	didi_nume	numéro d'ordre pour les chargements DIDI (NUME_DIDI dans ETAT_INIT)
character(len=8)	criterion	valeur de CRITERE dans ETAT_INIT (pour sélection par un instant)
real(kind=8)	precision	valeur de PRECISION dans ETAT_INIT (pour sélection par un instant)
real(kind=8)	user_time	valeur de l'état initial donnée par un instant dans ETAT_INIT
aster_logical	l_user_time	drapeau pour dire que la valeur de l'état initial est donnée par un instant dans ETAT_INIT
integer	user_nume	valeur de l'état initial donné par un numéro d'ordre dans ETAT_INIT
aster_logical	l_user_nume	drapeau pour dire que la valeur de l'état initial est donnée par un numéro d'ordre dans ETAT_INIT
real(kind=8)	stin_time	valeur de l'état initial défini par INST_ETAT_INIT dans ETAT_INIT
aster_logical	l_stin_time	drapeau pour dire que la valeur de l'état initial est définie par INST_ETAT_INIT dans ETAT_INIT
real(kind=8)	init_time	valeur de l'instant initial
integer	init_nume	valeur de du numéro d'ordre initial
character(len=19)	list_load_resu	nom de l'objet JEVEUX stockant la liste des chargements dans la SD résultat
aster_logical	l_init_stat	drapeau pour dire que l'état initial est stationnaire (pour la thermique)
aster_logical	l_init_vale	drapeau pour dire que l'état initial est une valeur (pour la

		thermique)
real(kind=8)	temp_init	température donnée par l'état initial quand c'est une valeur (pour la thermique)
type(NL_DS_TableIO)	table_io	Structure de données pour la table_container PARA_CALC

Cet objet rassemble donc à la fois des informations issus de l'utilisateur, mais aussi la définition des champs d'entrée-sortie et de leur comportement, défini par le développeur. Un champ a plusieurs états :

- Il peut définir un état initial ;
- Il peut être observé ;
- Il peut être archivé.

Ces trois états ne sont pas forcément indépendants. Par exemple, un champ qui se lit dans l'état initial est forcément archivable mais l'inverse n'est pas vrai (il existe des champs devant être archivés mais n'étant pas dans l'état initial comme COMPORTEMENT , CONT_NOEU , CRIT_STAB , etc.).
Un champ est utilisable (état initial, observation et archivage) seulement si certaines fonctionnalités sont actives (par exemple, le champ de vitesse en dynamique ou les statuts de contact pour le contact)
Si un champ est définit dans l'état initial, il pourrait être lu dans ETAT_INIT (soit de manière individuelle, par champ, soit dans la SD résultat donné dans ETAT_INIT), ou créé (égal à zéro) par la commande (dans ce cas, il faudra définir le nom de ce champ nul et le créer).
On a donc une liste de champs (variable field) définis par le type dérivé NL_DS_Field. Cet objet a la structure suivante :

Type	Nom	Description
character(len=16)	type	identificateur unique du champ. C'est aussi le nom symbolique défini dans la SD résultat
character(len=8)	gran_name	Type de grandeur (DEPL_R, SIEF_R, etc ...)
character(len=8)	field_read	nom du champ donné par l'utilisateur dans ETAT_INIT
character(len=4)	disc_type	type de discrétisation (NOEU, ELGA, ELNO, ...) du champ donné par l'utilisateur dans ETAT_INIT
character(len=8)	init_keyw	mot-clef correspondant à ce champ pour ETAT_INIT
character(len=16)	obsv_keyw	mot-clef correspondant à ce champ pour OBSERVATION (et SUIVI_DDL)
aster_logical	l_read_init	drapeau pour dire que ce champ doit être défini en état initial
aster_logical	l_store	drapeau pour dire que ce champ doit être archivé dans la SD résultat
aster_logical	l_obsv	drapeau pour dire que ce champ est « observable » (présent dans OBSERVATION et SUIVI_DDL)
character(len=24)	algo_name	nom de l'objet JEVEUX correspondant à ce champ dans l'algorithme
character(len=24)	init_name	nom de l'objet JEVEUX correspondant au champ initial nul dans l'algorithme
character(len=4)	init_type	dit comment ce champ a été initialisé (lu dans ETAT_INIT, champ par champ, dans la SD résultat, etc).

La gestion de la SD se fait en trois temps :

- Création de la SD dans la routine CreateInOutDS . Cette création comprend en particulier la création de tous les champs possibles. C'est dans cette routine que le développeur peut ajouter et définir le comportement (état initial, archivable, observable, etc..) d'un nouveau champ ;
- Lecture des informations données par l'utilisateur (mot-clef ETAT_INIT). Routine ReadInOut dans nmdata ;
- Initialisation de la SD : routine nmetcr pour la mécanique et ntetcr pour la thermique . Ces routines vont activer les champs suivant les fonctionnalités présentes

Les routines utilitaires d'accès à la SD sont les suivantes :
- GetIOField => récupérer les informations d'un champ donné
- SetIOField => donner les informations d'un champ donné

Ces routines utilise le type du champ en entrée (variable type de la SD NL_DS_Field).

Contexte d'utilisation : le développeur veut ajouter un champ, le rendre utilisable dans OBSERVATION, ARCHIVAGE, SUIVI_DDL ou ETAT_INIT.
1. Il commence par éditer la routine CreateInOutDS_M ou CreateInOutDS_M selon que l'on soit en mécanique ou en thermique. Il suffit de compléter exhaustivement les informations au début de cette routine. Si l'on ajoute un nouveau champ, il conviendra de modifier le nombre total (paramètre nb_field_maxi);
2. S'il doit ajouter un champ dans la SD résultat, il est nécessaire par ailleurs de modifier rscrsd ;
3. Le champ est désormais _possiblement_ activable. Mais on doit l'activer (par exemple sous condition d'une fonctionnalité active) dans la routine nmetac ;
4. Enfin, s'il est nécessaire d'avoir un état initial vierge, il convient de modifier la routine nmetc0 pour créer ce champ.

2.4.2.8 Gestion du contact – NL_DS_Contact

Le contact est géré par deux types de structures de données :
- Pour la *définition* du contact et des liaisons unilatérales[1] (opérateur DEFI_CONTACT), il faut se référer à la documentation [D4.06.14] de la sd_contact. Cette SD n'est pas concernée dans ce document ;
- Pour la *résolution* du contact et des liaisons unilatérales, on introduit un type dérivé appelé NL_DS_Contact.

L'objet principal ds_contact (de type NL_DS_Contact) contient les paramètres utiles à la résolution du problème de contact/frottement. Un seul objet ds_contact de type NL_DS_Contact gère l'ensemble du contact/frottement. Il a la structure suivante :

Type	Nom	Description
aster_logical	l_contact	drapeau pour dire que le contact ou le suintement est activé dans l'opérateur
aster_logical	l_meca_cont	drapeau pour dire que le contact mécanique est activé dans l'opérateur
aster_logical	l_meca_unil	drapeau pour dire que le suintement est activé dans l'opérateur
character(len=8)	sdcont	nom du concept DEFI_CONTACT renseigné dans STAT_NON_LINE/CONTACT
character(len=24)	sdcont_defi	nom de l'objet JEVEUX pour la définition du contact
character(len=24)	sdcont_solv	préfixe des objets JEVEUX pour la résolution du contact
character(len=24)	sdunil_defi	nom de l'objet JEVEUX pour la définition de LIAISON_UNIL
character(len=24)	sdunil_solv	préfixe des objets JEVEUX pour la résolution de LIAISON_UNIL
aster_logical	l_form_cont	drapeau pour dire qu'on est en FORMULATION='CONTINUE'
aster_logical	l_form_disc	drapeau pour dire qu'on est en FORMULATION='DISCRET'
aster_logical	l_form_xfem	drapeau pour dire qu'on est en FORMULATION='XFEM'
aster_logical	l_form_lac	drapeau pour dire qu'on est en FORMULATION='LAC'
aster_logical	l_elem_slav	drapeau pour dire la présence d'éléments esclaves de contact (CONTINUE/LAC/XFEM)

1 Les conditions unilatérales (LIAISON_UNILATER) gèrent les conditions de succion et de suintement pour les problèmes en THM

character(len=8)	ligrel_elem_slav	nom du <LIGREL> pour les éléments esclaves (créés dans DEFI_CONTACT)
aster_logical	l_elem_cont	drapeau pour dire la présence d'éléments de contact (CONTINUE/LAC/XFEM)
character(len=19)	ligrel_elem_cont	nom du <LIGREL> pour les éléments de contact (créés dans STAT_NON_LINE)
aster_logical	l_iden_rela	drapeau pour dire la présence de relations d'identité à traiter par modification de la matrice (XFEM avec ELIM_ARETE ou méthode LAC)
character(len=24)	iden_rela	nom de la SD définissant des relations d'identité
aster_logical	l_dof_rela	drapeau pour dire la présence de relations linéaires entre DDL (QUAD8 pour les méthodes discrètes ou XFEM, créées dans DEFI_CONTACT)
character(len=8)	ligrel_dof_rela	nom de la SD définissant des relations linéaires entre DDL
character(len=19)	field_input	nom du CHAM_ELEM en entrée des TE pour les méthodes XFEM/CONTINUE/LAC
character(len=14)	nume_dof_frot	nom du NUME_DDL pour la matrice de frottement discret
character(len=19)	field_cont_node	nom du champ aux noeuds CONT_NOEU pour le post-traitement
character(len=19)	fields_cont_node	nom du champ aux noeuds simple CONT_NOEU pour le post-traitement
character(len=19)	field_cont_perc	nom du champ aux noeuds des percussions/impacts pour le post-traitement
integer	nb_loop	nombre effectif de boucles de traitement du contact à gérer
integer	nb_loop_maxi	nombre maximum de boucles de traitement du contact à gérer
NL_DS_Loop	loop (nb_loop_maxi)	pointeurs vers les boucles de traitement du contact à gérer
aster_logical	l_renumber	drapeau pour renumérotation de la matrice
real(kind=8)	geom_maxi	valeur pour contrôler la boucle sur la géométrie
aster_logical	l_getoff	Drapeau pour l'indicateur de décollement (thêta-schéma)
aster_logical	l_first_geom	Drapeau pour dire que c'est la première itération de boucle géoémtrique
aster_logical	l_pair	Drapeau pour dire qu'il faut apparier
aster_logical	l_wait_conv	Drapeau pour dire qu'il faut attendre le point fixe des sous-itérations de contact (frottement) discret

Pour gérer les boucles de point fixe du contact (géométrie, frottement et statuts de contact), on a crée le type dérivé NL_DS_Loop dont voici la structure :

Type	Nom	Description
character(len=4)	type	Chaîne identifiant le type de la boucle (Geom, Fric et Cont)
integer	counter	itérateur pour le compteur de boucle
aster_logical	conv	rapeau pour dire que cette boucle est convergée
aster_logical	error	drapeau pour dire qu'il y a eu une erreur lors de l'évaluation de la boucle

real(kind=8)	vale_calc	valeur de convergence de la boucle
character(len=16)	locus_calc	endroit de convergence de la boucle

La gestion de la SD se fait en trois temps :
- Création de la SD dans la routine CreateContactDS ;
- Lecture des informations données par l'utilisateur (mot-clef CONTACT). Routine ReadContact dans nmdata ;
- Initialisations de la SD :
  - Routine InitContact dans nminit . Cette phase comprend en particulier la lecture du type de contact pour les LIGREL à gérer (voir routine nmdoct) ;
  - Routine cfmxsd dans nminit. Création des objets nécessaires pour la résolution du contact (essentiellement discret). Ce sont des objets dynamiques (dépendant du nombre de noeuds en contact) et il est donc nécessaire d'en faire des objets JEVEUX ;
  - Routine cucrsd dans nminit. Création des objets nécessaires pour la résolution des liaisons unilatérales. Ce sont des objets dynamiques(dépendant du nombre de noeuds) et il est donc nécessaire d'en faire des objets JEVEUX ;
  - Routine nmdoct : gestion des LIGREL tardifs

L'utilisation de la SD est directe par appel à ses sous-objets (usage de %) ou par des routines utilitaires déjà existantes (comme les routines cfdis* et mminf*), déjà utilisées pour l'accès à la sdcont provenant de DEFI_CONTACT.

2.4.2.9 Gestion de la convergence – NL_DS_Conv

Il s'agit ici de gérer les informations et l'algorithme concernant la gestion de la convergence, ce qui comprend :
- la gestion et le calcul des résidus d'équilibre ;
- la gestion des options venant du mot-clef facteur CONVERGENCE.

On crée les types dérivés NL_DS_Resi , NL_DS_ResiRefe et NL_DS_Conv. Un seul objet ds_conv de type NL_DS_Conv gère l'ensemble de la convergence. Il contient différentes informations :
- les informations récupérées dans le mot-clef CONVERGENCE ;
- un emplacement pour stocker la valeur ayant déclenché la bascule RESI_GLOB_RELA vers RESI_GLOB_MAXI ;
- deux variables pour gérer la recherche linéaire.

Il a la structure suivante :

Type	Nom	Description
integer	nb_resi	nombre de résidus
integer	nb_resi_maxi	nombre de résidus au maximum
type(NL_DS_Resi)	list_resi(max)	liste des résidus
aster_logical	l_resi_test(max)	drapeau pour dire si on doit utiliser ce résidu pour la convergence
integer	nb_refe	nombre de composantes de type RESI_REFE_RELA
integer	nb_refe_maxi	nombre maximum de composantes de type RESI_REFE_RELA
type(NL_DS_RefeResi)	list_refe(max2)	liste des composantes
aster_logical	l_refe_test(max2)	drapeau pour dire si la composante est active ou pas
integer	iter_glob_maxi	paramètre ITER_GLOB_MAXI
integer	iter_glob_elas	paramètre ITER_GLOB_ELAS
aster_logical	l_stop	paramètre ARRET="NON"
aster_logical	l_iter_elas	drapeau pour dire que l'utilisateur a renseigné

		explicitement ITER_GLOB_ELAS
real(kind=8)	swap_trig	valeur ayant déclenché la bascule RESI_GLOB_RELA vers RESI_GLOB_MAXI
real(kind=8)	line_sear_coef	coefficient de la recherche linéaire
integer	line_sear_iter	nombre d'itérations de recherche linéaire

Cet objet contient la liste des résidus possibles (actuellement, il y a six résidus entrant dans l'évaluation de la convergence). Chaque résidu est défini lui-même par le type dérivé NL_DS_Resi qui a la structure suivante :

Type	Nom	Description
character(len=16)	type	identifiant du type de résidu
character(len=16)	col_name	identifiant de la colonne du tableau de convergence stockant la valeur du résidu
character(len=16)	col_name_locus	identifiant de la colonne du tableau de convergence stockant l'emplacement de la norme maxi du résidu (quand INFO_RESIDU='OUI')
character(len=16)	event_type	identifiant de l'événement de divergence du résidu
real(kind=8)	vale_calc	valeur de la norme maxi du résidu
character(len=16)	locus_calc	emplacement de la norme maxi du résidu
real(kind=8)	user_para	valeur du critère donnée par l'utilisateur
aster_logical	l_conv	drapeau pour indiquer que vale_calc < user_para

Enfin, l'objet ds_conv contient également la liste des objets définissant le résidu de type RESI_REFE_RELA. Actuellement, onze types de composantes existent. Un objet de type NL_DS_ResiRefe a la structure suivante :

Type	Nom	Description
character(len=16)	type	type de la composante
real(kind=8)	user_para	valeur de la composante donnée par l'utilisateur
character(len=8)	cmp_name	nom de la composante dans la grandeur pour le TE

Cette SD est principalement utilisée pour l'évaluation des résidus. La routine qui évalue leur convergence est nmcore. Désormais, elle est autonome dans le sens où l'ajout d'un nouveau type de résidu ne nécessite pas d'impact spécifique pour cette partie (par contre, il faudra réaliser le calcul effectif de ce résidu, qui est actuellement, principalement réalisé dans nmresi).
Pour faciliter la maintenance et la lisibilité, on a également extrait la partie qui s'occupe des « bascules » :

- Passage de RESI_GLOB_RELA à RESI_GLOB_MAXI quand le chargement extérieur est nul ;
- Passage de RESI_COMP_RELA à RESI_GLOB_RELA au premier instant

Remarque :
Il est possible de faire une double bascule RESI_COMP_RELA => RESI_GLOB_RELA => RESI_GLOB_MAXI.

La gestion de la SD se fait en trois temps :

1. Création de la SD dans la routine CreateConvDS dans nmini0. Cette création comprend en particulier la création de tous les résidus disponibles. Leur utilisation effective dans l'évaluation de la convergence dépendra de l'état du drapeau de leur activation ;
2. Lecture des informations données par l'utilisateur (mot-clef AFFICHAGE). Routine nmdocn dans nmdata . Les informations ont stockées dans ds_conv ;

3. Initialisation de la SD dans `InitConv` . Cette routine gère en particulier les alarmes (`ARRET='NON'` ou résidu trop lâche) et active les résidus spécifiques au contact (car ils sont lus dans `nmdoct` et non dans `nmdocn`).

Pour manipuler la SD, on a les paires de routines `SetResi/GetResi` et `SetResiRefe`.

2.4.2.10 Gestion des paramètres de l'algorithme – `NL_DS_AlgoPara`

Cet objet stocke deux types d'informations :

- les données issues des options du mot-clef facteur `NEWTON` (`REAC_ITER`, types de matrices, options pour la recherche linéaire, etc.) ;
- les résultats intermédiaires de l'algorithme : résultats de la recherche linéaire et valeurs des résidus de convergence calculé.

  On crée les types dérivés `NL_DS_LineSearch` et `NL_DS_AlgoPara` . Un objet de type `NL_DS_AlgoPara` gère les paramètres de l'algorithme, il a la structure suivante :

Type	Nom	Description
`character(len=16)`	`method`	mot-clef `METHODE` (Newton, Newton-Krylov, IMPLEX)
`character(len=16)`	`matrix_pred`	type de matrice en prédiction
`character(len=16)`	`matrix_corr`	type de matrice en correction
`integer`	`reac_incr`	valeur de `REAC_INCR`
`integer`	`reac_iter`	valeur de `REAC_ITER`
`real(kind=8)`	`pas_mini_elas`	valeur de `PAS_MINI_ELAS`
`integer`	`reac_iter_elas`	valeur de `REAC_ITER_ELAS`
`aster_logical`	`l_line_search`	drapeau pour dire que la recherche linéaire est activé
`type(NL_DS_LineSearch)`	`line_search`	objet pour décrire les paramètres de la recherche linéaire
`character(len=8)`	`result_prev_disp`	SD résultat pour `DEPL_CALCULE/EXTRAPOLE`
`aster_logical`	`l_matr_rigi_syme`	Drapeau pour symétriser la matrice de rigidité (`MATR_RIGI_SYME`)

L'objet `line_search` est un type dérivé de type `NL_DS_LineSearch` dont voici la structure :

Type	Nom	Description
`character(len=16)`	`method`	type de recherche linéaire
`real(kind=8)`	`resi_rela`	tolérance pour la recherche linéaire
`integer`	`iter_maxi`	nombre maxi d'itérations de recherche linéaire
`real(kind=8)`	`rho_mini`	valeur minimale du coef. de recherche linéaire
`real(kind=8)`	`rho_maxi`	valeur maximale du coef. de recherche linéaire
`real(kind=8)`	`rho_excl`	valeur à exclure pour le coef. de recherche linéaire

On définit une seule variable `ds_algopara` de type `NL_DS_AlgoPara` qui sera transmise comme argument dans les routines qui en ont besoin. La gestion de la SD se fait en trois temps :

1. Création de la SD dans la routine `CreateAlgoParaDS` dans `nmini0` ;
2. Lecture des informations données par l'utilisateur (mot-clef `NEWTON`). Routine `nmdomt` ;
3. Lecture des informations données par l'utilisateur (mot-clef `RECH_LINEAIRE`). Routine `nmdomt_ls` ;
4. Initialisation de la SD par la routine `InitParaAlgo`.

2.4.2.11 Gestion de l'extraction de champs – `SDEXTR`

Le SUIVI_DDL partage avec l'OBSERVATION (voir §2.4.2.13) une SD commune appelé SDEXTR, routine d'extraction des valeurs des champs. On va donc commencer par décrire cette dernière SD. Les trois premiers objets sont généraux et leur longueur est proportionnelle au nombre d'occurrences du mot-clef SUIVI_DDL ou OBSERVATION.

SDEXTR – Objets	
Nom (attention aux blancs!)	**Description**
SDEXTR(1:14)//' .INFO'	Informations diverses sur les données d'extraction
SDEXTR(1:14)//' .EXTR'	Type d'extraction • Sur le champ (NOEU et ELGA) • Sur la maille (si ELGA) • Sur les composantes ou formule entre les composantes
SDEXTR(1:14)//' .ACTI'	Extraction active ou pas

On ne peut avoir plus de 99 occurrences des mots-clefs car on construit le nom de certains objets à partir de ce numéro d'occurrence OCC. Ces objets sont les suivants :

SDEXTR – Objets	
Nom (attention aux blancs!)	**Description**
SDEXTR(1:14)//OCC//' .NOEU'	Nœuds concernés par l'extraction
SDEXTR(1:14)//OCC//' .MAIL'	Mailles concernées par l'extraction
SDEXTR(1:14)//OCC//' .POIN'	Points d'intégration concernés par l'extraction
SDEXTR(1:14)//OCC//' .SSPI'	Sous-points d'intégration concernés par l'extraction
SDEXTR(1:14)//OCC//' .CMP'	Composantes concernées par l'extraction

2.4.2.12 Gestion du SUIVI_DDL – SDSUIV

Cette SD sert à gérer la fonctionnalité SUIVI_DDL. En plus des références à l'extraction des champs (SDEXTR, §2.4.2.11), nous avons un objet spécifique pour le SUIVI_DDL.

SDSUIV – Objets	
Nom (attention aux blancs!)	**Description**
SD SUIV (1:14)//' . TITR '	Titres des colonnes

2.4.2.13 Gestion de l'OBSERVATION – SDOBSE

Cette SD sert à gérer la fonctionnalité OBSERVATION. L'OBSERVATION partage avec le SUIVI_DDL (voir §2.4.2.12) une SD commune appelé SDEXTR, déjà décrite dans (§2.4.2.11).

Ensuite, nous avons des objets spécifiques pour l'OBSERVATION. Un qui va donner le nom de la table et un qui stocke les informations relatives à la fréquence d'observation (objet utilitaire SDSELI, voir §2.4.2.19), indicé par le numéro d'occurrence OCC du mot-clef OBSERVATION.

SDOBSE – Objets	
Nom (attention aux blancs!)	**Description**
SDOBSE(1:14)//' .TABL'	Nom de la table
SDOBSE(1:14)//OCC//'.LI'	Accès à la SDSELI, liste des instants à observer

2.4.2.14 Gestion des critères de qualité – SDCRIQ

Cette SD sert à évaluer les critères de qualité (mot-clef CRIT_QUALITE). Elle est très simple et ne comporte qu'un objet.

SDCRIQ – Objets	
Nom	**Description**
SDSUIV (1:1 9)//'.ERRT'	Valeur des erreurs THM espace et temps, coefficient THETA

2.4.2.15 Gestion du pilotage – SDPILO

Cette SD sert au pilotage (méthode de continuation). La plupart de ces objets sont regroupés dans une logique de type (réels avec réels, chaînes avec chaînes), plutôt que dans une logique de fonctionnalité.

SDPILO – Objets	
Nom	**Description**
SDPILO(1:19)//'.PLTK'	Contient les paramètres du pilotage (de type chaîne) ou les noms d'objets nécessaires au pilotage
SDPILO(1:19)//'.PLIR'	Contient les paramètres du pilotage (de type réel)
SDPILO(1:14)//'.PLCR'	Liste des coefficients pour le pilotage
SDPILO(1:14)//'.PLSL'	Liste des DDL de type DX, DY et DZ pour le pilotage
SDPILO(1:14)//'.PLCI'	Liste des coefficients pour le pilotage – Cas XFEM

Il n'y a pas de routines d'accès de haut-niveau, La SD est créée dans la routine nmdopi. Il faut attaquer directement les SD.

2.4.2.16 Gestion des numérotations – SDNUME

Cette SD vient en complément des deux SD NUMDDL et NUMFIX pour gérer la numérotation des équations. NUMDDL est la numérotation des équations, qui peut être variable au cours du transitoire, quand on utilise des fonctionnalités comme le contact CONTINUE ou le contact XFEM. NUMFIX est la numérotation fixe, elle est utile dans le cas des macro-éléments, pour pouvoir combiner les matrices. La SDNUME contient trois autres objets :

SDNUME – Objets	
Nom	**Description**
SDNUME(1:19)//'.NDRO'	Repérage des DDL pour les grandes rotations
SDNUME(1:19)//'.NUCO'	Repérage des DDL pour les Lagrangiens de contact et frottement
SDNUME(1:19)//'.ENDO'	Repérage des DDL pour l'endommagement aux nœuds

Ces trois objets ont la même logique : ils sont dimensionnés au nombre total de degrés de liberté de la structure, avec la même numérotation que les matrices et les CHAM_NO utilisés dans op0070. Une valeur particulière (en général 1), sert à repérer a priori des DDL spécifiques : ceux correspondant aux grandes rotations, ceux correspondant aux lagrangiens de contact/frottement et ceux correspondant à l'endommagement. On les utilise alors dans certains cas (par exemple, pour la mise à jour spécifiques des champs dans le cas des grandes rotations ou pour filtrer les composantes dans l'évaluation des résidus). Il n'y a pas de routines d'accès spécifiques.

2.4.2.17 Gestion de la dynamique – SDDYNA

Cette SD contient toutes les informations nécessaires au calcul en dynamique.

SDDYNA – Objets	
Nom	**Description**
SDDYNA(1:15)//'.PARA_SCH'	Paramètres des schémas en temps
SDDYNA(1:15)//'.INFO_SD'	Paramètres de la dynamique
SDDYNA(1:15)//'.NOM_SD'	Nom des SD pour la dynamique (modes statiques, vecteurs, ...)
SDDYNA(1:15)//'.TYPE_FOR'	Type de formulation (déplacement, vitesse ou accélération)
SDDYNA(1:15)//'.COEF_SCH'	Coefficients à utiliser dans le calcul
SDDYNA(1:15)//'.TYPE_CHA'	Informations relatives au chargement ONDE_PLANE
SDDYNA(1:15)//'.NBRE_CHA'	Informations relatives à certains chargements spécifiques à la dynamique
SDDYNA(1:15)//'.VEEL_OLD'	Vecteurs élémentaires du pas précédent pour les schémas multi-pas
SDDYNA(1:15)//'.VEAS_OLD'	Vecteurs assemblés du pas précédent pour les schémas multi-pas
SDDYNA(1:15)//'.VECENT'	Quantités en entrainements (déplacements, vitesses et accélération)
SDDYNA(1:15)//'.VECABS'	Quantités absolues (déplacements, vitesses et accélération)

L'accès à ces informations se fait via quatre routines dédiées chacune à un type, avec une chaine posant la question (sur le modèle de la routine dismoi) :

Opération sur la gestion de la dynamique – SDDYNA	Routine
Lecture information de type <booléen>	ndynlo
Lecture information de type <réel>	ndynre
Lecture information de type <entier>	ndynin
Lecture information de type <chaîne>	ndynkk

À noter que ces questions vont entraîner une erreur fatale si on n'est pas en dynamique, sauf dans le cas de la question NDYNLO(SDDYNA,'DYNAMIQUE') qui répondra .false. si on est en statique (c'est-à-dire qui sait détecter le cas où SDDYNA n'existe pas). En dehors de ces quatre routines, l'accès à la SDDYNA se fait directement dans deux routines :

Opération sur la gestion de la dynamique – SDDYNA	Routine
Lecture informations dans le fichier de commande	ndlect
Enregistrement valeur des différents coefficients	ndnpas

On revient sur les coefficients nécessaires pour le calcul en dynamique car ils sont très nombreux. Ces coefficients sont construits à partir de trois informations :

- Le type de schéma ;
- Les paramètres du schéma (coefficients ALPHA, BETA, KAPPA, etc.) ;
- L'incrément de temps ;

Le fait de dépendre du pas de temps implique que les coefficients sont ré-évalués à chaque pas (dans la routine ndnpas).

Coefficient	Description
COEF_MATR_RIGI	Coefficient devant la matrice de rigidité
COEF_MATR_AMOR	Coefficient devant la matrice d'amortissement
COEF_MATR_MASS	Coefficient devant la matrice de masse
COEF_DEPL_DEPL	Prédicteur en déplacement : coefficient devant le déplacement du pas précédent

COEF_DEPL_VITE	Prédicteur en déplacement : coefficient devant la vitesse du pas précédent
COEF_DEPL_ACCE	Prédicteur en déplacement : coefficient devant l'accélération du pas précédent
COEF_VITE_DEPL	Prédicteur en vitesse : coefficient devant le déplacement du pas précédent
COEF_VITE_VITE	Prédicteur en vitesse : coefficient devant la vitesse du pas précédent
COEF_VITE_ACCE	Prédicteur en vitesse : coefficient devant l'accélération du pas précédent
COEF_VITE_DEPL	Prédicteur en accélération : coefficient devant le déplacement du pas précédent
COEF_VITE_VITE	Prédicteur en accélération : coefficient devant la vitesse du pas précédent
COEF_VITE_ACCE	Prédicteur en accélération : coefficient devant l'accélération du pas précédent
COEF_DEPL	Coefficient devant l'incrément de déplacement
COEF_VITE	Coefficient devant l'incrément de vitesse
COEF_ACCE	Coefficient devant l'incrément d'accélération
COEF_MPAS_FEXT_PREC	Coefficient devant les forces extérieures du pas précédent (schéma multipas)
COEF_MPAS_FINT_PREC	Coefficient devant les forces intérieures du pas précédent (schéma multipas)
COEF_MPAS_FEXT_COUR	Coefficient devant les forces extérieures du pas courant (schéma multipas)
COEF_MPAS_EQUI_COUR	Coefficient devant les autres termes du second membre (inertie, amortissement)
COEF_FDYN_MASSE	Coefficient devant les forces de rappel dynamique (inertie)
COEF_FDYN_AMORT	Coefficient devant les forces de rappel dynamique (amortissement)
COEF_FDYN_RIGID	Coefficient devant les forces de rappel ?????? (sert à Krenk ???)
COEF_FORC_INER	Coefficient devant les forces d'inertie à mettre au dénominateur dans le résidu d'équilibre
INST_PREC	Pas de temps précédent (uniquement utile pour la poursuite avec, à la fois, le schéma HHT complet et les lois de comportement qui ont besoin de ce paramètre)

2.4.2.18 Gestion de la discrétisation temporelle – SDDISC

La structure de données SDDISC contient toutes les informations provenant de l'opérateur DEFI_LIST_INST (création du concept SDLIST) et des informations relatives à la gestion de la discrétisation temporelle dans op0070. Dans un premier temps, les informations provenant de l'opérateur DEFI_LIST_INST (voir [D4.06.17]) sont recopiées localement dans la SDDISC.

SDDISC – Objets	
Nom	**Description**
SDDISC(1:19)//'.LINF'	Informations sur la liste d'instants (voir contenu dans [D4.06.17]) Recopie de l'objet SDLIST(1:8)//'.LIST.INFOR'
SDDISC(1:19)//'.DITR'	Liste des instants – Cette liste est dynamique (en cas de découpe ou d'accélération du pas de temps)
SDDISC(1:19)//'.DINI'	Indicateur du niveau de sous-découpage pour chaque pas de temps. Initialement, le niveau de découpe vaut 1. Il ne peut y avoir plus d'un niveau de découpe entre deux pas successifs (vérification dans la routine nmdcin) – Cette liste est dynamique (en cas de découpe ou d'accélération du pas de temps)
SDDISC(1:19)//'.ITER'	Nombre d'itérations de Newton qu'il a fallu pour chaque pas de temps – Cette liste est dynamique (en cas de découpe ou d'accélération du pas de temps)
SDDISC(1:19)//'.EPIL'	Indicateur du choix de la solution de pilotage (action AUTRE_PILOTAGE)

SDDISC(1:19)//'.LIPO'	Liste des instants de calcul obligatoires. Cet objet sert dans le cas de l'adaptation automatique du pas de temps, il indique les instants qui seront calculés quoiqu'il arrive (même en cas d'agrandissement du pas de temps). On parle aussi d'instants « jalons ». Sa longueur est celle de la liste d'instants initiale donnée par l'utilisateur dans DEFI_LIST_INST. Il s'agit donc d'une liste ***non-dynamique***.
SDDISC(1:19)//'.REPC'	Indicateur d e gestion de la réactualisation du préconditionnement (action REAC_PRECOND)
SDDISC(1:19)//'.EEVR'	Recopie de l'objet SDLIST(1:8)// '.ECHE.EVENR' Voir contenu dans [D4.06.17]
SDDISC(1:19)//'.EEVK'	Recopie de l'objet SDLIST(1:8)// '.ECHE.EVENK' Voir contenu dans [D4.06.17]
SDDISC(1:19)//'.ESUR'	Recopie de l'objet SDLIST(1:8)// '.ECHE.SUBDR' Voir contenu dans [D4.06.17]
SDDISC(1:19)//'.AEVR'	Recopie de l'objet SDLIST(1:8)// '.ADAP.EVENR' Voir contenu dans [D4.06.17]
SDDISC(1:19)//'.AEVK'	Recopie de l'objet SDLIST(1:8)// '.ADAP.EVENK' Voir contenu dans [D4.06.17]
SDDISC(1:19)//'.ATPR'	Recopie de l'objet SDLIST(1:8)// '.ADAP.TPLUR' Voir contenu dans [D4.06.17]
SDDISC(1:19)//'.ATPK'	Recopie de l'objet SDLIST(1:8)// '.ADAP.TPLUK' Voir contenu dans [D4.06.17]
SDDISC(1:19)//'.AEXT'	Objet pour le prolongement de la découpe (traitement du cas de la COLLISION)
SDDISC(1:19)//'.IFCV'	Objet stockant l'état de convergence pour l'adaptation V(1) – Valeur du MAX(ITER_GLOB_MAXI,ITER_GLOB_ELAS) V(2) – Valeur du MIN(ITER_GLOB_MAXI,ITER_GLOB_ELAS) V(3) – Nombre maximum d'itérations possibles (y compris les itérations supplémentaires possibles par ITER_SUPPL) V(4) – Valeur de 'PAS_MINI_ELAS' V(5) – Valeur de 'RESI_GLOB_RELA' V(6) – Valeur de 'RESI_GLOB_MAXI' V(7) – Type de résidu demandé par l'utilisateur : =1 pour RESI_GLOB_RELA =2 pour RESI_GLOB_MAXI =3 pour RESI_GLOB_RELA et RESI_GLOB_MAXI V(8) – Valeur de 'INIT_NEWTON_KRYLOV' V(9) – Valeur de 'ITER_NEWTON_KRYLOV' V(1 0) – Indique qu'on autorise des itérations en plus
SDDISC(1:19)//'.IFRE'	Objet stockant la valeur des résidus à chaque itération de Newton V(3*(ITER-1)+0) – Valeur de RESI_GLOB_RELA V(3*(ITER-1)+1) – Valeur de RESI_GLOB_MAXI V(3*(ITER-1)+2) – Valeur du chargement

Pour manipuler la SDDISC, on utilise une série d'utilitaires.

Opération sur la gestion de la discrétisation temporelle – SDDISC	**Routine**
Routine utilitaire générale qui permet d'accéder au contenu des objets .LINF, .EEVR, .EEVK, .ESUR, .AEVR, .AEVK, .ATPR, .ATPK, .REPC et .EPIL. L'accès peut se faire en lecture ou en écriture .	utdidt

Retourne .RUE. si on sort de la liste d'instants (fin du transitoire)	didern
Valeur de l'instant selon numéro de l'instant	diinst

2.4.2.19 Gestion de la sélection d'un instant – SDSELI

La structure de données SDSELI permet de sélectionner un instant suivant une fréquence, une liste de valeurs, en prenant en compte une tolérance et un type de critère (absolu ou relatif).

SDSELI – Objets	
Nom	**Description**
SDSELI(1:19)//'.INFL'	Informations générales sur la sélection (paramètres de sélection)
SDSELI(1:19)//'.LIST'	Liste des instants donnés par l'utilisateur

Pour opérer sur cette SD, on utilise principalement deux routines :
- une qui lit les paramètres de l'utilisateur (nmcrpx) ;
- recherche si l'instant donné est sélectionné (nmcrpo) ;

Opération sur la gestion de la sélection d'un instant – SDSELI	Routine
Lecture de la liste des instants à sélectionner	nmcrpa
Lecture de la précision et du type de critère (relatif ou absolu)	nmcrpp
Lecture de toutes les informations (appel à nmcrpa et nmcrpp)	nmcrpx
Recherche d'un réel dans une liste (avec précision et critère)	utacli
Routine principale de recherche de l'instant	nmcrpo
Recherche de l'indice dans la SD résultat juste avant un instant donné	nmttch

Cette SD est utilisée pour la gestion de la discrétisation temporelle, de l'état initial, de l'archivage et de l'observation.

2.4.2.20 Gestion des critères de convergence – SDCRIT

Cette SD gère des informations relatives aux critères de convergence, elle sert en particulier à stocker les résidus dans la SD RESULTAT.

SD CRIT – Objets	
Nom	**Description**
SDCRIT(1:19)//'.CRTR'	Valeurs des informations (résidus, nombre d'itérations, etc.)
SDCRIT(1:19)//'.CRDE'	Nom des informations pour le stockage dans la SD RESULTAT (résidus, nombre d'itérations, etc.)

Cette SD est aussi utilisée pour la commande THER_NON_LINE (attention, les objets ne sont pas de la même dimension !). La sauvegarde des informations pour cette SD se fait dans la routine nmcore qui évalue la convergence à chaque itération de Newton.

Opération sur la gestion des critères de convergence – SDCRIT	Routine
Création de la SD	nmcrcv ntcrcv
Sauvegarde des informations dans la SDCRIT	nmcore op0186

Sauvegarde des informations dans la SD RESULTAT	nmarc0 ntarc0

2.4.2.21 Gestion de la liste de sélection – NL_DS_SelectList

Cette structure de données permet de sélectionner un instant à partir d'une liste ou d'une fréquence.

Type	Nom	Description
integer	nb_value	Nombre de valeurs dans la liste des instants donnés
real(kind=8)	incr_mini	Incrément de temps minimum entre deux instants de la liste
real(kind=8), pointer	list_value	Liste des instants
real(kind=8)	precision	Valeur de la précision pour sélectionner un instant
aster_logical	l_abso	Vaut .true. Si la valeur est sélectionné en absolue
real(kind=8)	tolerance	Tolérance calculée si sélection en relatif
integer	freq_step	Fréquence de sélection des instants
aster_logical	l_by_freq	Vaut .true. Si on sélection par la fréquence

La routine selectListRead permet de lire les mots-clefs de l'utilisateur et remplit la structure de données.
La routine selectListGet retourne .true. Si l'instant courant est bien à sélectionner par rapport aux paramètres (fréquence, liste d'instants avec précision, etc.). Enfin, la routine selectListClean désalloue la mémoire (nécessaire car on utilise un pointeur pour la liste des instants).

2.4.2.22 Gestion du calcul modale en cours de calcul non-linéaire – NL_DS_PostTimeStep, NL_DS_Spectral, NL_DS_Stability et NL_DS_SpectralResults

Ces structures de données servent à gérer les mots-clefs MODE_VIBR et CRIT_STAB.

Structure de données pour sauvegarder les paramètres de l'analyse modale : NL_DS_Spectral.

Type	Nom	Description
character(len=16)	option	Option de calcul
character(len=16)	type_matr_rigi	Type de matrice de rigidité
aster_logical	l_small	Vaut .true. si on calcule les plus petites valeurs propres
aster_logical	l_strip	Vaut .true. si on calcule les valeurs propres dans une bande
real(kind=8)	strip_bounds(2)	Bande de fréquence si l_strip
integer	nb_eigen	Nombre de valeurs propres si l_small
integer	coef_dim_espace	Paramètre pour le solveur modal
type(NL_DS_SelectList)	selector	Sélecteur des instants de calcul
character(len=16)	level	Niveau de calcul : par bande, plus petites valeurs ou mode calibration (uniquement nombre de valeurs propres)

Structure de données pour sauvegarder les paramètres de l'analyse de stabilité (CRIT_STAB): NL_DS_Stability.

Type	Nom	Description

aster_logical	l_geom_matr	Vaut .true. pour prendre en compte la rigidité géométrique
aster_logical	l_modi_rigi	Vaut .true. pour prendre en compte la modification de la rigidité
integer	nb_dof_excl	Nombre de DDL exclus
character(len=8), pointer	list_dof_excl	Liste des DDL exclus
integer	nb_dof_stab	Nombre de DDL de stabilisation
character(len=8), pointer	list_dof_stab	Liste des DDL de stabilisation
character(len=16)	instab_sign	Valeur du signe de l'instabilité
real(kind=8)	instab_prec	Valeur absolue de l'instabilité

Structure de données pour sauvegarder les paramètres de l'analyse modale (MODE_VIBR et CRIT_STAB): NL_DS_PostTimeStep.

Type	Nom	Description
aster_logical	l_crit_stab	Si CRIT_STAB est renseigné
aster_logical	l_mode_vibr	Si MODE_VIBR est renseigné
type(NL_DS_Spectral)	crit_stab	Paramètres généraux de CRIT_STAB
type(NL_DS_Spectral)	mode_vibr	Paramètres généraux de MODE_VIBR
type(NL_DS_Stability)	stab_para	Paramètres spécifiques de CRIT_STAB
type(NL_DS_TableIO)	table_io	Structure de données table_container ANALYSE_MODALE
aster_logical	l_hpp	Vaut .true. si HPP

2.4.2.23 Gestion des tables container – NL_DS_TableIO

Cette structure de données permet de gérer les table_container attachées à STAT_NON_LINE et DYNA_NON_LINE.

Type	Nom	Description
character(len=8)	result	Nom de la structure de données résultats
character(len=19)	table_name	Nom JEVEUX de la table_container
character(len=24)	table_type	Type de la table_container
integer	nb_para	Nombre total de paramètres dans la table_container (nombre de colonnes)
integer	nb_para_inte	Nombre de paramètres de type entier
integer	nb_para_real	Nombre de paramètres de type réel
integer	nb_para_cplx	Nombre de paramètres de type complexe
integer	nb_para_strg	Nombre de paramètres de type chaîne
character(len=24), pointer	list_para	Liste des noms des paramètres
character(len=8), pointer	type_para	Liste des types des paramètres

La routine nonlinDSTableIOCreate permet de créer la structure de données et la table_container dans la structure de données evol_noli.
La routine nonlinDSTableIOSetPara permet de donner la liste des paramètres dans la table. On peut les donner de deux manières :

- soit à partir de la structure de données NL_DS_Table (voir § 2.4.2.2) ;
- soit à partir d'une liste exhaustive de ces paramètres.

La routine nonlinDSTableIOClean désalloue les objets (nécessaire à cause de l'usage de pointeurs).

2.4.2.24 Gestion du comportement CARCRI et COMPOR

La structure de données CARCRI est une sd_carte qui contient des réels. Elle stocke actuellement 22 paramètres dont voici la liste.

Indice	Contenu
1	Mot-clef COMPORTEMENT/ITER_INTE_MAXI
2	Mot-clef COMPORTEMENT/TYPE_MATR_TANG
3	Mot-clef COMPORTEMENT/RESI_INTE_RELA
4	Mot-clef COMPORTEMENT/PARM_THETA
5	Mot-clef COMPORTEMENT/ITER_INTE_PAS
6	Mot-clef COMPORTEMENT/ALGO_INTE
7	Mot-clef COMPORTEMENT/VALE_PERT_RELA
8	Mot-clef COMPORTEMENT/RESI_CPLAN_MAXI
9	Mot-clef COMPORTEMENT/ITER_CPLAN_MAXI
10	Mot-clef COMPORTEMENT/RESI_RADI_RELA
11	Entier codé pour la présence de variables de commandes de type ExternalStateVariable
12	Mot-clef SCHEMA_THM/PARM_THETA
13	Mot-clef COMPORTEMENT/POST_ITER
14	Pointeur vers le nombre de variables internes StateVariable pour MFront
15	Pointeur vers les noms des variables internes StateVariable pour MFront
16	Pointeur vers la loi de comportement MFront
17	Indicateur de symétrie de la matrice tangente de comportement
18	Mot-clef SCHEMA_THM / PARM_ALPHA
19	Pointeur vers les noms des paramètres matériaux pour MFront
20	Pointeur vers les noms des paramètres matériaux pour MFront
21	Mot-clef COMPORTEMENT/POST_INCR
22	Type de modèle cinématique pour MFront

La structure de données COMPOR est une sd_carte qui contient des chaînes de caractère. Elle stocke actuellement 20 paramètres dont voici la liste.
L'indice est donné dans le fichier Behaviour_type.h.

Nom de l'indice	Contenu
RELA_NAME	Mot-clef COMPORTEMENT/ RELATION
NVAR	Nombre de variables internes
DEFO	Mot-clef COMPORTEMENT/DEFORMATION

INCRELAS	Indicateur de modèle incrémental ou total
PLANESTRESS	Modèle en contraintes planes (Deborst ou analytique)
NUME	Indice d'appel de la loi de comportement (lcxxxx)
MULTCOMP	Nom de la structure de données issue de DEFI_COMPOR
POSTITER	Contenu du m ot-clef COMPORTEMENT/ POST_ITER
KIT1_NAME THMC_NAME CREEP_NAME CABLE_NAME	Nom de la première relation pour un kit
KIT2_NAME THER_NAME PLAS_NAME SHEATH_NAME	Nom de la deuxième relation pour un kit
KIT3_NAME COUPL_NAME HYDR_NAME	Nom de la troisième relation pour un kit
KIT4_NAME CPLA_NAME MECA_NAME	Nom de la quatrième relation pour un kit
KIT1_NUME THMC_NUME PLAS_NUME	Indice d'appel de la première relation pour un kit
KIT2_NUME THER_NUME CREEP_NUME	Indice d'appel de la deuxième relation pour un kit
KIT3_NUME HYDR_NUME	Indice d'appel de la troisième relation pour un kit
KIT4_NUME MECA_NUME	Indice d'appel de la quatrième relation pour un kit
KIT1_NVAR THMC_NVAR CREEP_NVAR	Nombre de variables internes de la première relation pour un kit
KIT2_NVAR THER_NVAR PLAS_NVAR	Nombre de variables internes de la deuxième relation pour un kit
KIT3_NVAR HYDR_NVAR	Nombre de variables internes de la troisième relation pour un kit
KIT4_NVAR MECA_NVAR	Nombre de variables internes de la quatrième relation pour un kit

3 Gestion de l'algorithme

Nous allons nous intéresser à la gestion de l'algorithme. Pour gérer la convergence, les différents niveaux de boucle et les erreurs survenues lors d'un calcul, on utilise un formalisme commun, basé sur la notion **d'événement**.

3.1 Les différentes boucles

Dans l'algorithme, il y a cinq niveaux de boucles <BOUC> :

- <RESI> : la boucle sur les différents résidus d'équilibre demandés par l'utilisateur (RESI_GLOB_RELA , RESI_GLOB_MAXI , etc) ;
- <NEWT> : la boucle sur les itérations de Newton ;
- <FIXE> : la boucle sur les points fixes (contact) ;
- <INST> : la boucle sur les instants de calcul ;
- <CALC> : la boucle sur le calcul.

La « boucle » <CALC> n'en est pas vraiment une. Elle sert à gérer deux situations :

1. Le calcul se termine normalement (pas d'erreur) mais autrement que lorsqu'on a atteint la fin de la boucle sur les pas de temps. Pour l'instant, le seul cas est celui où le pilotage a atteint ses bornes ;
2. Un événement est déclenché à l'extérieur de la boucle sur les pas de temps, c'est-à-dire pendant le post-traitement (détection d'instabilité par modes vibratoires) ;
   Les différentes boucles sont réparties entre op0070, nmnewt (statique et dynamique implicite) et ndexpl (dynamique explicite). Pour la boucle sur les résidus d'équilibre, il faut regarder dans la routine nmcore, appelée par nmconv. Pour la boucle sur les points fixes, on utilise les routines nmible/nmtble, contenues dans nmnewt.

3.2 État des boucles

L'état d'une boucle est stocké dans l'objet SDERRO (§2.4.2.6), on y accède par nmleeb et nmeceb. Il existe six états :

1. CONT : la boucle continue ;
2. CONV : la boucle a convergé (événements de type convergence et divergence) ;
3. ERRE : une erreur (événement de type erreur) est survenue pendant la boucle, on va la traiter ;
4. EVEN : un événement (événement de type informatif) est survenu pendant la boucle ;
5. STOP : une erreur (événement de type erreur) est survenue pendant la boucle, on s'arrête ;
6. CTCD : état particulier de la boucle de Newton pour le contact discret ;

3.3 Les événements

On peut classer les événements selon leur origine :

- Les événements définis par l'utilisateur dans DEFI_LIST_INST (par exemple, DELTA_GRANDEUR) ;
- Les événements intrinsèques à l'algorithme : les erreurs et la convergence des différentes boucles ;

3.3.1 Types des événements

Un événement est défini par son type qui est une chaîne de caractère en deux parties <XXXX_YYYY>. La première chaîne <XXXX> décrit le type de l'événement :

- Les événements donnant lieu à une erreur sont préfixés par <ERRC_*> ou <ERRI_*> ;
- Les événements donnant lieu à une convergence sont préfixés par <CONV_*> ;
- Les événements donnant lieu à une divergence sont préfixés par <DIVE_*> ;
- Les événements informatifs sont préfixés par <EVEN_*> ;

La seconde chaîne <YYYY> décrit le niveau de boucle de l'événement :

- Boucle sur les résidus d'équilibre <*_RESI> ;
- Boucle sur les itérations de Newton <*_NEWT> ;
- Boucle de point fixe pour le contact <*_FIXE> ;
- Boucle sur les pas de temps <*_INST> ;

Cette information décrit dans quelle boucle les événements peuvent potentiellement se déclencher.

3.3.1.1 Événements de type erreur <ERR*_*>

Des erreurs sont à traiter immédiatement (c'est-à-dire dès déclenchement de l'événement) car elles sont bloquantes pour le processus, elles sont notées <ERRI>. Les erreurs à traiter uniquement à convergence de la boucle sont notées <ERRC>.
Par exemple :

- <ERRI_NEWT> est une erreur à traiter immédiatement dans une itération de Newton comme un défaut d'intégration de la loi de comportement empêche ou une matrice non factorisable ;
- <ERRC_NEWT> est une erreur à traiter à convergence de Newton, par exemple une sortie d'un critère physique hors des bornes de son domaine de définition ;
- <ERRI_FIXE> est une erreur à traiter immédiatement dans une boucle de point fixe ;

Remarque :

Les erreurs typées <ERRI_CALC> entraînent l'arrêt immédiat du calcul car aucune action ne peut les traiter. Ce sont les arrêts en manque de temps CPU et l'arrêt extérieur par utilisateur.

3.3.1.2 Événements de type convergence <CONV_*>

Les événements de type convergence se traitent à chaque niveau de boucle. Cette convergence peut-être liée à une fonctionnalité activée ou pas (cf §2.4.1.1).

3.3.1.3 Événements de type divergence <DIVE_*>

Les événements de type divergence se traitent à chaque niveau de boucle. Cette divergence peut-être liée à une fonctionnalité activée ou pas (cf §2.4.1.1).
En pratique, l'état de chaque boucle est plutôt traitée en divergence. En effet, pour éviter d'avoir à vérifier qu'une fonctionnalité spécifique est active (pilotage, Deborst, contact, etc) et donc tester la convergence d'une boucle en vérifiant cette fonctionnalité, on préfère émettre un événement de divergence qu'un événement de convergence : on ne peut avoir divergence que si la fonctionnalité est activée, alors qu'on pourrait avoir convergence même si la fonctionnalité n'est pas activée.

3.3.1.4 Événements de type informatif <EVEN>

Les événements de type informatifs sont les autres événements (ni erreur, ni convergence). Par défaut, ils ne servent qu'à donner des indications à l'utilisateur (souvent sous la forme de messages d'alarmes, le passage de RESI_GLOB_RELA à RESI_GLOB_MAXI par exemple). Certains ne sont activables que sur demande de l'utilisateur (commande DEFI_LIST_INST).
Ces événements pourront aussi être traité explicitement (autrement que par l'émission d'une alarme ou d'une information) via DEFI_LIST_INST.
Ces événements ne sont pas liés à un niveau de boucle.

3.3.1.5 Le cas des code-retour

Un code-retour est un entier unique donnant le statut d'un opération. Il est possible de lier un événement à la valeur d'un code-retour. Chaque valeur du code-retour peut générer un événement différent, de n'importe quel type. Les codes retour -1 et 0 ont toujours le même sens :

- -1 : on n'a rien fait (pas de factorisation, pas d'intégration de loi de comportement, etc.) ;
- 0 : on a fait quelque chose et tout s'est bien passé ;

Les autres valeurs de codes retour ont un sens dépendant de leur type :

Type	Description	Signification du code	
FAC	Retour de la factorisation	-1	Pas de factorisation
		0	Tout s'est bien passé
		1	La matrice est singulière

		2	La factorisation a échoué
		3	On ne sait pas dire si la matrice est singulière
PIL	Retour du pilotage	-1	Pas de pilotage
		0	Tout s'est bien passé
		1	Pas de solution de l'équation de pilotage
		2	On a atteint une borne (fin du calcul)
LDC	Retour de l'intégration de la loi de comportement	-1	Pas d'intégration du comportement
		0	Tout s'est bien passé
		1	Échec lors de l'intégration de la loi de comportement
		2	Un paramètre physique n'est pas dans son domaine de définition
		3	La contrainte SIZZ n'est pas nulle (algorithme de De Borst)
CTC	Retour du traitement du contact discret	-1	Pas de contact discret
		0	Tout s'est bien passé
		1	Le nombre maximum d'itérations de contact a été atteint
		2	La matrice du contact est singulière

3.4 Gestion des événements

3.4.1 Modification, ajout ou suppression d'un événement

La plupart des événements standardisés sont prévus dans la structure de données et les utilitaires qui y sont liés. Dans la plupart des cas, il ne s'agit de modifier que la routine nmcrga (voir §2.4.2.6 en y ajoutant la caractéristique du l'événement dans les DATA en début de routine.

Description	**DATA**
Nom de l'événement	NEVEN
Nom du code-retour lié à l'événement (XXX si pas code-retour)	NCRET
Valeur du code-retour lié à l'événement (99 si pas code-retour)	VCRET
Type et niveau de déclenchement de l'événement	TEVEN
Fonctionnalité activant un événement de type convergence ou divergence	FEVEN
Code du message à afficher quand l'événement se déclenche	MEVEN

Le système de vérification des messages fait que le code du message à afficher (MEVEN) n'est pas suffisant, il faut aussi impacter nmevim.

3.4.2 Émissions des événements

Pour déclencher un événement le développeur doit le prévoir dans l'algorithme en appelant la routine nmcrel. Par exemple :

Commande	**Effet**
CALL NMCREL(SDERRO,'ERRE_TIMN',.TRUE.)	Manque de temps CPU pendant une itération de Newton

CALL NMCREL(SDERRO,'RESI_MAXR',.TRUE.)	Passage de RESI_GLOB_RELA à RESI_GLOB_MAXI
CALL NMCREL(SDERRO,'DIVE_FIXG',. FALSE .)	Pas de divergence de la boucle de point fixe sur la géométrie

Si l'événement est de type <ERRI_BOUC>, le statut de la boucle <BOUC> est automatiquement modifié. C'est une précaution pour éviter que le développeur oublie de traiter l'événement (et donc risque de provoquer des résultats faux). Une erreur immédiate de niveau <ERRI> est répercutée à tous les niveaux de boucle pour les mêmes raisons.
Pour activer un événement lié à un code-retour, il faut utiliser la routine nmcret au lieu de nmcrel. Par exemple CALL NMCRET(SDERRO,'LDC',LDCCVG) s'occupe de transformer en événement la valeur du code retour de la loi de comportement.

3.4.3 Traitement des événements

L'idée principale est de traiter les événements à tous les niveaux de boucle, de changer l'état des boucles selon le résultat de ce traitement.

3.4.3.1 Événements de type convergence et divergence

Les événements de type convergence ou divergence sont examinés à chaque niveau de boucle, dans une routine dédiée appelée nmevcv . Cette routine modifie l'état de la boucle :

1. On suppose initialement que la boucle continue (état CONT, cf §3.2)
2. On boucle sur les événements de type <CONV_*> et <DIVE_*> en prenant en compte des éventuelles fonctionnalités activées ;
3. On calcule l'état résultat pour ce niveau de boucle : si tous les événements de divergence sont faux et tous les événements de convergence sont vrais, alors ce niveau de boucle est dans l'état convergé ;
4. On vérifie les états de convergence des boucles intérieures à ce niveau. Pour que l'état final de cette boucle soit convergé, toutes les boucles intérieures doivent être convergées. Si c'est le cas, la boucle passe dans l'état convergé (état CONV, cf §3.2)
5. On modifie l'état de la boucle courante en appelant la routine nmeceb ;
6. La routine nmeceb transmet l'état de la boucle aux boucles supérieures (évite les erreurs non-traitées) ;

Boucle	Code	Routine	Routine appelante
Résidus	RESI	nmcvgr	nmconv
Newton	NEWT	nmcvgn	nmnewt
Point fixe	FIXE	nmcvgf	nmtble
Pas de temps	INST	nmcvgp	op0070
Calcul	CALC	nmcvgc	op0070

3.4.3.2 Événements de type erreurs

Les événements de type erreur sont particulièrement sensibles et doivent être traités de manière très rigoureuse pour éviter les résultats faux. En premier lieu, l'événement de type erreur a été émis par le biais des routines standards : nmcrel et nmcret. On rappelle que dans le cas des événements erreur, on modifie systématiquement l'état de la boucle en mode ERRE dans l'optique d'éviter de continuer ces boucles si jamais l'erreur n'a pas été traitée convenablement par l'algorithme (oubli du développeur).
De manière systématique, les événements de type erreur sont testés dans l'algorithme via la routine nmltev. Si un événement de type erreur est activé, un GOTO est immédiatement fait dans le programme.

4 Algorithme général

Le processus est standardisé au maximum, pour chaque niveau de boucle :
- Évaluation de la convergence par appel aux routines nmcvg* ;
- Action suite à la situation de la boucle par appel aux routines nmact* ;

4.1 Lectures et initialisations globales

La routine op0070 gère l'algorithme global et se découpe en trois parties :
- Lecture et initialisations des données ;
- Boucle sur les pas de temps ;
- Post-traitements ;
- Gestion des erreurs globales ;
- Archivage ;

Pour la lecture et l'initialisation des données, on se reportera en particulier aux informations relatives à la gestion des SD (§2). L'ensemble des initialisations faites à ce niveau sera valable pour tout le calcul. La routine principale d'initialisation est nminit. Voici les opérations prévues dans cette routine :

Opérations d'initialisations dans nminit	**Routine**
Création du profil de la matrice et de la SDNUME	nmnume
Création des variables-chapeaux	nmchap
Création du vecteur des fonctionnalité activées list_func_acti	nmfonc
Création des vecteurs dans les variables-chapeaux	nmcrch
Création de la structure de données pilotage	nmdopi
Duplication NUME_DDL pour SDNUME	nmpro2
Préparation de la carte COMPOR (transformation en CHAM_ELEM_S)	nmdoco
Création de SDDISC et SDOBSE	diinit
Initialisation des variables de commande	nmcrvv
Pré-calcul des MATR_ELEM constants au cours du calcul	nminmc
Pré-calcul des VECT_ELEM et VECT_ASSE constants au cours du calcul	nminvc
Création de la SDCRIT	nmcrcv
Initialisation du calcul par sous-structuration	nmlssv
Création de la SD pour le chargement FORCE _SOL	nmexso
Calcul de l'accélération initiale	accel0
Création de la SDCONV	nmcrcg
Initialisation de NL_DS_Print	nminim
Pré-calcul des MATR_ASSE constants au cours du calcul	nminma
Réalisation d'une observation initiale	nmobsv
Création de la SD RESULTAT (EVOL_NOLI)	nmnoli
Création de la table des grandeurs (pour ERRE_THM)	cetule
Calcul du second membres initial pour les schémas multi-pas en dynamique	nmihht

L'algorithme général de op0070 est résumé sur l'organigramme (Figure 1).

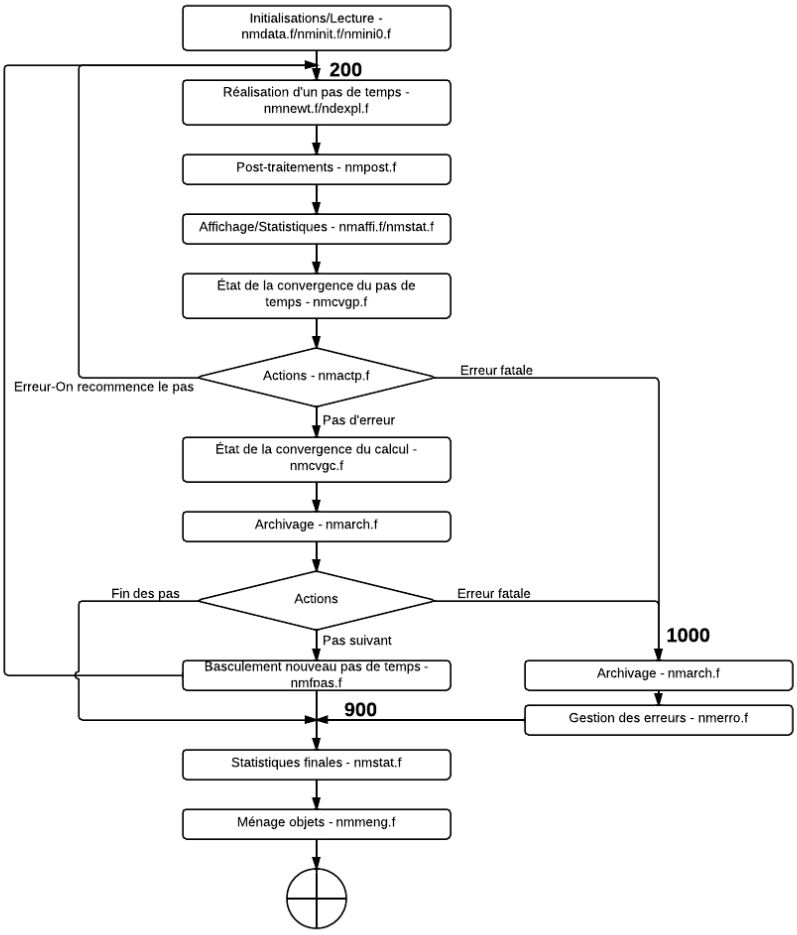

Figure 1: Organigramme général de `op0070`

4.2 Réalisation d'un pas de temps

Un pas de temps utilise plusieurs niveaux de boucles imbriqués :

- Des boucles de point fixe, utilisées exclusivement pour le contact (jusqu'à trois niveaux de boucle) ;
- Une boucle sur les itérations de Newton. Un processus de Newton contient une prédiction d'Euler et des corrections de Newton ;
  L'algorithme général de `nmnewt` est résumé sur l'organigramme (Figure 2).

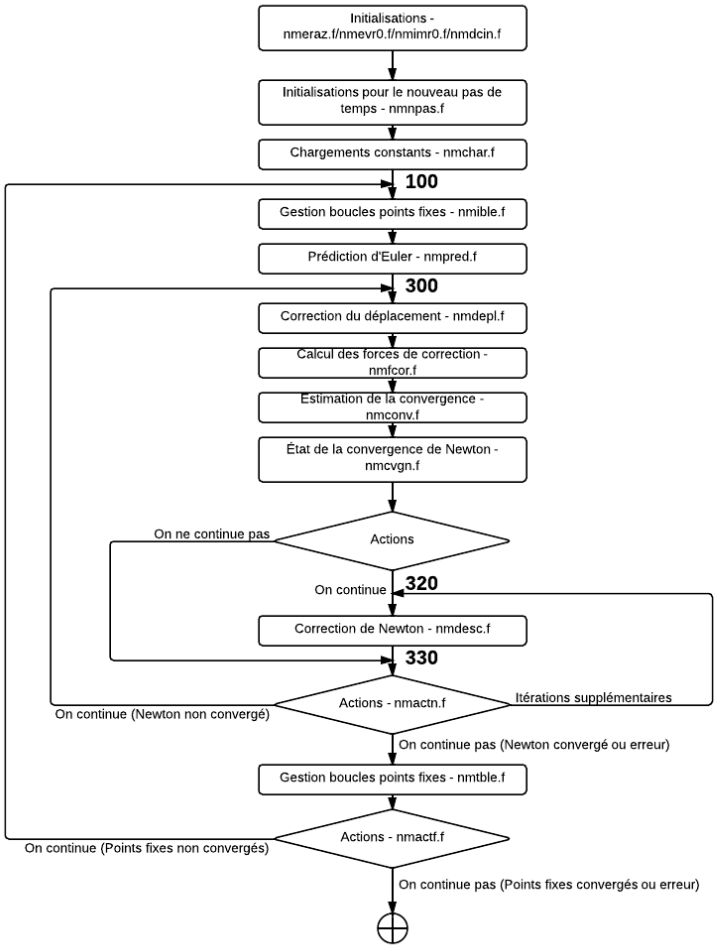

Figure 2: Organigramme général de `nmnewt`

4.2.1 Initialisations du pas

Chaque pas de temps est initialisé par une série de routines :

- Les routines `nmeraz`, `nmevr0` s'occupent d'initialiser la gestion des événements ;
- La routine `nmdcin` vérifie que la découpe du pas de temps est correcte (contrôle du nombre de niveau de découpe d'un pas de temps à l'autre) ;

La routine la plus importante est la routine `nmnpas` . Elle réalise les opérations suivantes :

- Les routines `nmimr0` et `nminin` initialisent l'affichage, en particulier le tableau de convergence (ce qui permet de faire varier la nature de celui-ci d'un pas de temps à l'autre) ;
- La routine `nmvcre` prépare les variables de commande pour le pas de temps courant ;
- On initialise à zéro l'incrément de déplacement cumulé depuis le début du pas de temps (vecteur `DEPDEL` , voir § 2.4.1.6). Attention l'initialisation est spéciale à cause de la prise en compte des grandes rotations ;
- On initialise toutes les données pour la dynamique, en particulier les différents coefficients (voir § 2.4.2.17) dans `ndnpas` ;
- On initialise différentes informations pour Newton-Krylov et le contact (routines `cfinit` , `mmapin` et `nmnkft`) ;

4.2.2 Prédiction d'Euler

La prédiction d'Euler est une estimation de l'incrément de déplacement en linéarisant le problème par rapport au temps. Il existe plusieurs méthodes et plusieurs types de matrice utilisables. Le calcul proprement dit est réalisé dans la routine nmpred (et ses -jolies- filles), au moyen des principes établis dans le §5.

4.2.3 Mise à jour des champs

La mise à jour des champs est réalisée dans la routine nmdepl. Cette mise à jour consiste à modifier les vecteurs solutions (déplacements, vitesses et accélérations), en prenant en compte éventuellement :

- La recherche linéaire sans pilotage ;
- La recherche de η pour le pilotage avec ou sans recherche linéaire ;
- Le contact (formulation discrète) ou les liaisons unilatérales ;

Voici les phases :

1. Recalcul des efforts extérieurs, routine nmfext ;
2. Conversion des incréments de solution DEPSO1 et DEPSO2 (voir §2.4.1.6), issus de la résolution du système linéaire, vers les incréments en déplacement/vitesse/accélération (prise en compte des coefficients de changement de schéma), via la routine nmincr ;
3. Calcul recherche linéaire et pilotage, routines nmreli, nmpich et nmrepl ;
4. Mise à jour de la direction de descente (prise en compte des coefficients issus de la recherche linéaire et du pilotage), routine nmpild ;
5. Modification des déplacements par le contact et les liaisons unilatérales , par la routine nmcoun ;
6. Actualisation des champs de solutions dans nmmajc ;

La phase 4.2.3 consiste à mettre à jour les champs « plus » (DEPPLU, VITPLU, ACCPLU) et les champs cumulés (DEPDEL, VITDEL, ACCDEL), en prenant en compte les éventuelles grandes rotations (mise à jour des quaternions) et de l'endommagement aux nœuds.

4.2.4 Forces de correction

Puisque les déplacements ont été modifiés (voir § 4.2.3), il est nécessaire de ré-évaluer les forces variables : soit les forces extérieures de type « suiveuse », soit les forces intérieures et les réactions de contact/frottement, soit les forces liées à la dynamique (inertie, amortissement). L'ensemble est réalisé dans la routine nmfcor.

4.2.5 Estimation de la convergence

L'estimation de la convergence de Newton est un moment critique, qui va conditionner la justesse du calcul. L'ensemble de l'estimation est réalisé dans la routine nmconv . Cette routine s'occupe du calcul des résidus (nmresi), de l'estimation de leur convergence (nmcore). C'est donc dans cette routine qu'on va traiter la boucle <RESI> sur les résidus d'équilibre. Mais on doit aussi prendre en compte le nombre d'itérations de Newton, le cas de la convergence du contact (formulations discrète ou continue), de la méthode de De Borst ou de la méthode IMPLEX.

Remarque :

Il convient d'être très prudent dans la modification de cette routine.

4.2.6 Correction de Newton

Si le processus n'a pas convergé, on procède au calcul d'une correction de Newton. On réalise donc le calcul d'un système linéaire (voir §5) dans la routine nmdesc (comme direction de ***desc***ente).

4.2.7 Boucle sur les points fixes

Il existe trois boucles dite « de point fixe » entre la boucle sur les pas de temps et la boucle sur les itérations de Newton. Ces boucles sont utilisés pour le contact :

1. Boucle de point fixe sur la géométrie ;
2. Boucle de point fixe sur les seuils de frottement ;
3. Boucle de point fixe sur les statuts de contact ;

Ces trois boucles sont gérées par le duo de routine `nmible`/`nmtble`, via le passage de la variable `NIVEAU`, qui indique dans quel type de boucle de point fixe on se trouve actuellement.

5 Construction et résolution des systèmes

Une partie importante de l'algorithme non-linéaire consiste à résoudre des systèmes linéaires, que l'on construit en quatre temps :

- Calcul et assemblage des chargements ;
- Calcul et assemblage des seconds membres ;
- Calcul et assemblage de la matrice ;
- Résolution du système linéaire ;

Dans cette partie, nous allons décrire les différentes phases et les principales routines à considérer.

5.1 Systèmes à résoudre

Actuellement, il existe plusieurs systèmes différents qui sont résolus dans l'opérateur `op0070`. Ils auront toujours la forme suivante (sauf pour le post-traitement, qui utilise les opérateurs de recherche de valeurs propres) :

$$\begin{bmatrix} K & B^T \\ B & 0 \end{bmatrix} \cdot \begin{Bmatrix} \delta r \\ \delta s \end{Bmatrix} = \begin{Bmatrix} L_r \\ L_s \end{Bmatrix} \quad (1)$$

$\left[K\right]$ est la matrice de rigidité ou une combinaison linéaire de matrices et $\left[B\right]$ est la matrice des Lagrange de Dirichlet.

$\left\{\delta r\right\}$ est l'incrément de solution des valeurs nodales inconnues du système et $\left\{\delta s\right\}$ est l'incrément des paramètres de Lagrange associés à la dualisation des conditions limites. Concrètement, les « valeurs nodales inconnues » du systèmes peuvent être des déplacements, des vitesses, des accélérations, des pressions, des températures, etc. Les deux seconds membres sont également des valeurs nodales. Le détail des systèmes est donné dans les documentations de référence [R5.03.01] et [R5.05.05].

5.2 La routine `merimo`

Pour le calcul des matrices tangentes (options `FULL_MECA`, `FULL_MECA_ELAS`, `RIGI_MECA_TANG`, `RIGI_MECA_ELAS`, `RIGI_MECA`, `RIGI_MECA_IMPLEX`) et les efforts intérieurs (option `RAPH_MECA`), on passe systématiquement par la routine `merimo`. Cette routine prépare les champs d'entrées (nombreux dans ce cas), elle est appelée à la fois pour le calcul des vecteurs élémentaires pour les efforts intérieurs (voir §5.5.2) mais aussi pour le calcul des matrices élémentaires tangentes de rigidité (voir §5.3).

5.3 Calcul des matrices

Nous considérons ici des matrices élémentaires (`MEELEM`) et des matrices assemblées (`MEASSE`). Tout comme pour le chargement (§5.5.1) , ce calcul utilise un système en deux temps :

- Création d'une **liste locale** des matrices à calculer ou assembler ;
- Calcul ou assemblage des matrices ;

Pour la phase de création de la liste des matrices à calculer ou assembler, il y a plusieurs routines (contrairement aux chargements qui n'utilisent que la routine `nmchar`). Ces routines sont les suivantes :

Opérations	Routine
Construction de la liste locale pour les matrices utilisées en dynamique explicite	`ndxprm`
Construction de la liste locale pour les matrices utilisées en correction	`nmcoma`
Construction de la liste locale pour les matrices utilisées en prédiction	`nmprma`
Construction de la liste locale pour les matrices utilisées en post-traitement (modes vibratoires ou modes de flambement)	`nmflma`
Construction de la liste locale pour les matrices constantes durant tout le calcul	`nminmc`

Pour la phase de calcul ou assemblage des matrices, on retrouve des routines similaires au cas des chargements.

Opérations	Routine
Ajout d'une matrice à assembler et/ou calculer dans la liste locale Initialisation de la liste locale	nmcmat nmcmat
Aiguillage calcul et/ou assemblage des matrices	nmxmat
Calcul des matrices	nmcalm
Assemblage des matrices	nmassm
Calcul et assemblage des matrices de rigidité	nmrigi

Il y a néanmoins une exception notable : le calcul des matrices tangentes non-linéaires utilise un appel direct par nmrigi car cela nécessite des paramètres supplémentaires (appel à merimo, voir 5.2) dont n'ont pas besoin les autres matrices. Le type de matrice MERIGI utilise donc nmrigi à la place de nmcalm et nmassm.
Si on désire modifier une matrice, il faut donc :

- Ajouter son calcul ou son assemblage au bon endroit (ndxprm, nmcoma, nmprma, nmflma ou nminmc) selon le cas, voire ajouter une nouvelle routine ;
- Ajouter le calcul élémentaire ou l'assemblage dans nmcalm et nmassm ;
  Les routines nmxmat et nmcmat de manipulation de la liste locale **ne doivent pas être modifiées**.

5.4 Calcul de la matrice résultante MATASS

Selon le type de matrice résultante que l'on désire obtenir (en prédiction, en correction ou pour l'accélération initiale), on utilise trois routines globales qui vont s'occuper de cette construction. Ce sont les routines nmprac , nmcoma et nmprma. Ces trois routines gèrent également l'affichage (option de calcul de la matrice de rigidité) et les statistiques (temps passé dans les opérations). Elles ont le même schéma de principe :

- Calcul ou assemblage de matrices élémentaires (MEELEM) dans des matrices assemblées (MEASSE), (voir § 5.3) ;
- Gestion des paramètres de réactualisation des matrices (rigidité, masse, amortissement), routine nmchrm ;
- Gestion du type de la matrice de rigidité (nom de l'option), routine nmchoi ;
- Construction de la matrice résultante par combinaison linéaire des autres matrices. En dynamique, les différentes contributions dans la matrice résultante (matrices de masse, d'amortissement et de rigidité) se combinent en utilisant un coefficient multiplicateur dépendant du schéma en temps utilisé et du pas de temps, on récupère ces coefficients via la routine d'accès NDYNRE (voir § 2.4.2.17). C'est la routine nmmatr ;
- Factorisation (ou pas) de la matrice résultante par appel à la routine preres ;

Opérations	Routine
Calcul de la matrice assemblée résultante pour le calcul de l'accélération initiale	nmprac
Calcul de la matrice assemblée résultante pour la phase de prédiction	nmprma
Calcul de la matrice assemblée résultante pour la phase de correction	nmcoma
Gestion des paramètres de réactualisation des matrices (rigidité, masse , amortissement)	nmchrm
Gestion du type de la matrice de rigidité (nom de l'option)	nmchoi
Choix de re-création de la numérotation	nmrenu
Calcul de la matrice résultante MATASS	nmmatr
Prise en compte des chargements suiveurs avec leur fonction multiplicatrice	ascoma
Prise en compte de la modification de la matrice résultante par le contact/frottement (méthode DISCRETE)	nmasfr

5.5 Calcul du second membre

Le calcul du second membre est la partie la plus différente d'un système linéaire à l'autre, il va contenir :

- Les chargements donnés (voir le § 5.5.1)
- Les forces internes provenant de l'intégration du comportement (§5.5.2) ;
- Les quantités liées aux conditions limites de Dirichlet comme les réactions d'appui (§5.5.3) ;
- Les quantités liées aux variables de commande (§5.5.4) ;
- Les différentes quantités liées au contact/frottement (pas de détails dans ce document) ;
- Les contributions d'inertie et d'amortissement pour la dynamique (§5.5.6) ;

Dans le cas de la dynamique, pour les schémas multi-pas (Newmark complet avec MODI_EQUI='OUI') on va de plus ajouter les contributions des pas de temps précédents.

5.5.1 Calcul des chargements

Il y a trois modes d'évaluation des chargements et deux phases. Les modes sont les suivants :

- Mode <ACCI> des chargements pour l'évaluation de l'accélération initiale en dynamique ;
- Mode <FIXE> des chargements pour l'évaluation des chargements fixes ne dépendant pas de la solution ;
- Mode <VARI> des chargements pour l'évaluation des chargements variables dépendant de la solution (chargements suiveurs) ;

Il y a également deux phases. Soit on est en correction, soit on est en prédiction. Ces phases ne sont utiles que pour des cas très particuliers : amortissement modal, méthode IMPLEX et impédance modale.
La routine principale qui calcule les chargements est nmchar. Dans cette routine, suivant les fonctionnalités activées (list_func_acti) et suivant le mode/phase de calcul, on va provoquer le calcul des vecteurs élémentaires (variable-chapeau VEELEM) et leur assemblage en vecteurs assemblées (variable-chapeau VEASSE). Pour cela, on utilise un certain nombre de routines utilitaires qui gèrent des listes locales à nmchar.

Opérations	Routine
Ajout d'un chargement à assembler ou calculer, avec une option éventuelle Initialisation de la liste des chargements (initialisations listes locales de nmchar)	nmcvec nmcvec
Aiguillage calcul ou assemblage des chargements	nmxvec
Calcul des chargements	nmcalv
Assemblage des chargements	nmassv

Certains chargements n'ont pas de phase de calcul élémentaire mais seulement la création d'un vecteur assemblé directement. Si on désire ajouter un chargement, il faut donc :

- Définir quelle phase et quel mode va l'activer ;
- Ajouter son calcul ou son assemblage dans nmchar selon la fonctionnalité activée ;
- Ajouter le calcul élémentaire et/ou l'assemblage dans nmcalv et nmassv ;

Les routines nmxvec, nmcvec **ne doivent pas être modifiées**.

5.5.2 Calcul des quantités liées aux efforts intérieurs

Le calcul des efforts intérieurs peut être réalisé de deux manières :

- Par intégration de la loi de comportement, c'est l'option de calcul RAPH_MECA ;
- Par ré-utilisation des contraintes calculées par ailleurs, c'est l'option de calcul FORC_NODA ;

Cette distinction est importante car elle n'engendre pas les mêmes coûts (le second cas est beaucoup plus rapide). L'intégration du comportement est une opération coûteuse qui implique de modifier variables internes et contraintes, et, souvent, de résoudre localement (à chaque point de Gauss), un système non-linéaire. Comme le calcul des contraintes est également nécessaire lors de l'évaluation des matrices tangentes, le comportement est aussi intégré lors du calcul des matrices tangentes (option FULL_MECA). En statique, la phase de prédiction calcule les efforts intérieurs par l'option FORC_NODA. En dynamique, on intègre systématiquement le comportement.

Opérations	Routine
Calcul des vecteurs élémentaires pour les efforts intérieurs (intégration du comportement)	nmfint
Assemblage des vecteurs élémentaires pour les efforts intérieurs (intégration du comportement)	nmaint

Le calcul de l'option FORC_NODA est fait par le mécanisme d'évaluation des chargements (§5.5.1).

5.5.3 Calcul des quantités liées aux conditions limites dualisées

La dualisation des conditions limites de Dirichlet (voir [R3.01.01]) entraîne le calcul de deux quantités de type vecteur assemblée :

- Les réactions d'appui, par le produit de la matrice des conditions limites dualisées avec les Lagrange correspondant aux degrés de liberté $[B].\{\lambda\}$, il s'agit du vecteur identifié par CNDIRI ;
- La valeur des degrés de liberté imposée par dualisation $[B].\{u\}$. A convergence, cette quantité sera égale aux conditions limites données $\{u^d\}$, le vecteur est identifié par CNBUDI ;

Opérations	Routine
Calcul des vecteurs élémentaires pour les réactions d'appui	nmdiri
Assemblage des vecteurs élémentaires pour les réactions d'appui	nmadir
Calcul et assemblage des quantités imposées par dualisation	nmbudi

5.5.4 Calcul des quantités liées aux variables de commande

Le calcul de ces quantités se fait par le mécanisme d'évaluation des chargements (§5.5.1), type CNVCF0, CNVCF1 et CNVCPR.

5.5.5 Calcul des quantités constantes pendant le calcul

Un certain nombre de vecteurs sont constants durant le calcul, ils utilisent alors un mécanisme similaire aux chargements (§5.5.1), à l'aide des routines nmxvec, nmcvec, nmcalv et nmassv, sauf qu'on passe par la routine nminvc au lieu de nmchar.

Opérations	Routine
Calcul et assemblage des vecteurs constants durant le calcul	nminvc

5.5.6 Calcul des quantités liées à la dynamique – Forces d'inertie et d'amortissement

Le calcul de ces quantités se fait par le mécanisme d'évaluation des chargements (§ 5.5.1), type CNDYNA. Ces quantités n'existent pas à l'état élémentaire puisqu'ils sont le résultat d'un produit matrice/vecteur.

Opérations	Routine
Calcul des quantités dynamiques pour le second membre	ndfdyn
Calcul des forces d'inertie	nminer
Calcul des forces d'amortissement	nmhyst

Selon les schémas en temps, il est nécessaire d'utiliser des coefficients multiplicateurs spécifiques devant chacune de ces quantités, on récupère ces coefficients via la routine d'accès NDYNRE (voir § 2.4.2.17).

5.5.7 Seconds membres résultants

Il en reste plus qu'à construire les différents seconds membres par combinaison linéaire de toutes les quantités détaillées précédemment. Les routines principales de construction de ces seconds membres sont au nombre de sept :

Opérations	Routine
Calcul du second membre pour la correction	nmassc
Calcul du second membre pour la prédiction – Option DEPL_CALCULE ou EXTRAPOLE	nmassd
Calcul du second membre pour l'accélération initiale	nmassi
Calcul du second membre pour la prédiction	nmassp
Calcul du second membre pour la prédiction – Cas statique	nsassp
Calcul du second membre pour la prédiction – Cas dynamique implicite	ndassp
Calcul du second membre pour la prédiction – Cas dynamique explicite	nmassx

Toutes ces routines reposent sur les mêmes principes :

- Récupération des quantités nécessaires, déjà assemblées ou calculées localement dans ces routines, via la variable-chapeau VEASSE (voir les §5.5.1 à §5.5.6) ;
- Récupération des différents coefficients multiplicateurs. En dynamique, selon les schémas en temps, il est nécessaire d'utiliser des coefficients multiplicateurs spécifiques devant chacune de ces quantités, on récupère ces coefficients via la routine d'accès NDYNRE (voir § 2.4.2.17) ;
- Construction d'un ou de deux seconds membres par combinaison linéaire (utilisation des routines standards vtaxpy). Il y a deux seconds membres dans le cas du pilotage ;

Pour améliorer la lisibilité, on utilise des routines utilitaires standardisées pour regrouper les cas les plus fréquents.

Opérations	Routine
Calcul des composantes du vecteur second membre pour les chargements de type Dirichlet, fixes au cours du pas de temps, pour le cas normal et le cas piloté	nmasdi
Calcul des composantes du vecteur second membre pour les chargements de type Neumann, fixes au cours du pas de temps, pour le cas normal et le cas piloté	nmasfi
Calcul des composantes du vecteur second membre pour les chargements de type Neumann, variables (suiveurs) au cours du pas de temps, pour le cas normal (pas de cas piloté)	nmasva
Calcul des composantes du vecteur second membre pour les chargements spécifiques de la dynamique, variables (suiveurs) au cours du pas de temps, pour le cas normal (pas de cas piloté)	ndasva
Calcul des composantes du vecteur second membre pour les chargements spécifiques de la dynamique, calcul de l'accélération initiale, variables (suiveurs) au cours du pas de temps, pour le cas normal (pas de cas piloté)	nmacva

5.6 Résolution du système

La résolution se fait via l'intermédiaire de la routine nmresd, qui prend en entrée la matrice résultante assemblée (voir § 5.4) et les seconds membres idoines (deux seconds membres si on a du pilotage).

Opérations	Routine
Résolution du système	nmresd

Résolution du système sur les degrés de liberté physiques (appel à `resoud`)	`nmreso`
Résolution du système sur les degrés de liberté généralisés	`nmresg`

La routine de résolution va donc retourner un ou deux champs solutions `DEPSO1` et `DEPSO2`, stockés dans la variable-chapeau `SOLALG`.