Mot-clé SOLVEUR

1 But

Le mot clé facteur SOLVEUR est commun à toutes les commandes qui résolvent des **systèmes d'équations linéaires** (STAT_NON_LINE, CALC_MODES, ...). Pour résoudre ces systèmes d'équations, on utilise des algorithmes particuliers appelés «solveurs linéaires». Le mot-clé SOLVEUR permet de choisir le solveur linéaire à utiliser parmi deux catégories: les solveurs directs et les solveurs itératifs.

Parmi les solveurs directs, on dispose de l'algorithme classique de «Gauss» (METHODE='LDLT'), d'une factorisation multifrontale ('MULT_FRONT') et de la bibliothèque MUMPS ('MUMPS').
Pour les solveurs itératifs, il est possible de faire appel à un gradient conjugué ('GCPC') ou à certains algorithmes de la bibliothèque PETSc ('PETSC').

Seuls MULT_FRONT, MUMPS et PETSC sont **parallélisés**. Le premier en mémoire partagée (OpenMP), les deux autres principalement en mémoire distribuée (MPI). Leurs schémas parallèles peuvent même se chaîner (PETSc + MUMPS en tant que préconditionneur) ou tirer partie d'un deuxième niveau de parallélisme (OpenMP dans MUMPS et bibliothèque BLAS).
Par contre, quelque soit leur niveau de parallélisme, tous les solveurs restent compatibles avec un traitement parallèle des calculs élémentaires et des assemblages.

D'autre part, seuls les trois solveurs directs sont compatibles avec le calcul modal, les études de flambement et certaines procédures de raffinement de pas de temps en non linéaire.

Pour plus de détails et de conseils sur l'emploi des solveurs linéaires on pourra consulter les notices d'utilisation spécifiques [U2.08.03]/[U2.08.06].

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Table des Matières

1 But....1
2 Syntaxe....3
3 Opérandes....6
3.1 Opérande METHODE....6
3.2 Paramètres communs à plusieurs solveurs....7
3.3 METHODE='MULT_FRONT'....9
3.4 METHODE='LDLT'....10
3.5 METHODE='MUMPS'....11
3.5.1 Paramètres fonctionnels....11
3.5.2 Paramètres de relaxation....12
3.5.3 Paramètres numériques....13
3.5.4 Paramètres pour la gestion mémoire....15
3.5.5 Paramètres pour réduire le temps calcul....18
3.6 METHODE='GCPC'....21
3.7 METHODE='PETSC'....23
3.7.1 Mots-clefs spécifiques au préconditionneur FIELDSPLIT....26
3.7.2 Mots-clefs spécifiques au préconditionneur UTILISATEUR....26
3.7.3 Mot-clef OPTION_PETSC....26

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2 Syntaxe

```
◊  SOLVEUR = _F (

# Paramètre commun à tous les solveurs (directs, itératifs)
    ◊ ELIM_LAGR=          /  'NON',                          [DEFAUT]
                          /  'OUI',
                          /  'LAGR2',            (pour MUMPS seulement)
# Paramètres communs aux solveurs directs ('MULT_FRONT', 'LDLT', 'MUMPS')
    ◊  NPREC=                        /  8,                   [DEFAUT]
                                     /  nprec                [I]
    ◊  STOP_SINGULIER=               /  'OUI',               [DEFAUT]
                                     /  'NON'

#Solveur direct «interne» de type multifrontal : cf. §3.3.
    /  METHODE='MULT_FRONT'
            ◊  RENUM=                /  'METIS',             [DEFAUT]
                                     /  'MD',
                                     /  'MDA'


#Solveur direct de type multifrontal basé sur la bibliothèque MUMPS : cf §3.5.
    /  METHODE='MUMPS',                                      [DEFAUT]
            ◊  TYPE_RESOL=           /  'AUTO',              [DEFAUT]
                                     /  'NONSYM',
                                     /  'SYMGEN',
                                     /  'SYMDEF'
            ◊  PCENT_PIVOT=          /  20,                  [DEFAUT]
                                     /  pcent                [R]
            ◊  RESI_RELA=            /  -1.0,  (non lin./modal) [DEFAUT]
                                     /  +1.e-6,(linéaire)    [DEFAUT]
                                     /  resi                 [R]
            ◊  FILTRAGE_MATRICE=     /  -1.d0,               [DEFAUT]
                                     /  filtma       (périmètre limité cf. §3.5)
            ◊  MIXER_PRECISION=      /  'OUI',       (périmètre limité cf. §3.5)
                                     /  'NON'                [DEFAUT]
            ◊  PRETRAITEMENTS=       /  'SANS',
                                     /  'AUTO'               [DEFAUT]
            ◊  RENUM=                /  'AUTO',              [DEFAUT]
                                     /  'AMD',
                                     /  'AMF',
                                     /  'QAMD',
                                     /  'PORD',
                                     /  'METIS',
                                     /  'PARMETIS' (uniquement avec la version MPI)
                                     /  'SCOTCH'
                                     /  'PTSCOTCH' (uniquement avec la version MPI)
            ◊  POSTTRAITEMENTS=      /  'SANS',
                                     /  'FORCE',
                                     /  'MINI',
                                     /  'AUTO'               [DEFAUT]
            ◊  MATR_DISTRIBUEE=      /  'OUI',       (périmètre limité cf. §3.5)
                                     /  'NON'                [DEFAUT]
            ◊  GESTION_MEMOIRE=      /  'AUTO',              [DEFAUT]
```


Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
                                    /   'IN_CORE',
                                    /   'OUT_OF_CORE',
                                    /   'EVAL'
```

#Paramètres limités à certaines versions de MUMPS (cf. §3.5)

```
            ◊   ACCELERATION     =      /'AUTO',                [DEFAUT]
                                        /   /'FR',
                                        /   /'FR+',
                                        /   /'FR++',
                                        /   /'LR',
                                        /   /'LR+'
                                        /   /'LR++',

            ◊   LOW_RANK_SEUIL=         /   0.0                 [DEFAUT]
                                        /   lr_seuil            [R]

            ◊   REDUCTION_MPI=          /   1,                  [DEFAUT]
                                        /   redmpi              [I]
```

 # Solveur itératif «interne» de type GCPC préconditionné par un Cholesky Incomplet $ILU(k)$ ou par une factorisée simple précision (via MUMPS). Cf. § 3.6.

```
    /   METHODE='GCPC',
            ◊   /   PRE_COND=           /   'LDLT_INC',         [DEFAUT]
                    ◊   NIVE_REMPLISSAGE=       /   0,          [DEFAUT]
                                                /   niv         [I]

                /   PRE_COND=           /   'LDLT_SP',
                                        /   'LDLT_DP',
                    ◊   PCENT_PIVOT =           /   20,         [DEFAUT]
                                                /   pcent       [I]
                    ◊   REAC_PRECOND=           /   30,         [DEFAUT]
                                                /   reac        [I]
                    ◊   GESTION_MEMOIRE=        /   'AUTO',        [DEFAUT]
                                                /   'IN_CORE',
                    ◊   RENUM=                  /   'SANS',        [DEFAUT]
                                                /   'PARMETIS',
                                                /   'METIS',
                    ◊   LOW_RANK_SEUIL=         /   1.E-08,        [DEFAUT]
                                                /   lr_seuil,      [R]

            ◊   NMAX_ITER=              /   0,                  [DEFAUT]
                                        /   niter               [I]
            ◊   RESI_RELA=              /   10^-6,              [DEFAUT]
                                        /   resi                [R]
```

#Solveurs itératifs issus de la bibliothèque externe PETSc : cf. § 3.7.

```
    /   METHODE='PETSC',
            ◊   ALGORITHME=             /   'FGMRES',           [DEFAUT]
                                        /   'GMRES',
                                        /   'GMRES_LMP',
                                        /   'CG',
                                        /   'CR',
                                        /   'GCR'
            ◊   /   PRE_COND=           /   'LDLT_SP',          [DEFAUT]
                                        /   'LDLT_DP',
                    ◊   PCENT_PIVOT =           /   20,         [DEFAUT]
```


Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
                                        /  pcent          [I]
                  ◊  REAC_PRECOND=      /  30,            [DEFAUT]
                                        /  reac           [I]
                  ◊  GESTION_MEMOIRE=   /  'AUTO',        [DEFAUT]
                                        /  'IN_CORE',
                  ◊  RENUM=             /  'SANS',        [DEFAUT]
                                        /  'PARMETIS',
                                        /  'METIS',
                  ◊  LOW_RANK_SEUIL=    /  0.0,           [DEFAUT]
                                        /  lr_seuil,      [R]


               /  PRE_COND=          /  'LDLT_INC',
                  ◊  NIVE_REMPLISSAGE=  /  0,             [DEFAUT]
                                        /  niv            [I]
                  ◊  REMPLISSAGE=       /  1.0,           [DEFAUT]
                                        /  rem            [R]


               /  PRE_COND=          /  'ML',
                                     /  'BOOMER',
                                     /  'GAMG',
                                     /  'BLOC_LAGR'


               /  PRE_COND=          /  'JACOBI',
                                     /  'SOR',
                                     /  'SANS'


               /  PRE_COND=          /  'FIELDSPLIT',
                  ♦  NOM_CMP=        /  (),                 [listK]
                  ♦  PARTITION_CMP=     (),                 [listI]

               /  PRE_COND=          /  'UTILISATEUR',
                  ♦  KSP_UTIL=       /  fonc,               [fonction]
                   ◊  RENUM =        /  ' SANS ',              [DEFAUT]


            ◊  NMAX_ITER=            /  0,                  [DEFAUT]
                                     /  niter               [I]
            ◊  RESI_RELA=            /  10-6,               [DEFAUT]
                                     /  resi                [R]
            ◊  MATR_DISTRIBUEE=      /  'OUI',     (périmètre limité cf. §3.7)
                                     /  'NON'               [DEFAUT]


            ◊  OPTION_PETSC=         /  '',                 [K]
),
```


Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

3 Opérandes

3.1 Opérande METHODE

Ce mot clé permet de choisir la méthode de résolution des systèmes linéaires :

#Solveurs directs

/'MULT_FRONT' **Solveur direct de type multifrontale** (sans pivotage pendant la factorisation). Cette méthode est parallélisée en mémoire partagée (OpenMP) et peut être exécutée sur plusieurs processeurs (*via* l'interface Astk menu Options / Options de lancement / ncpus).

/'LDLT' **Solveur direct avec factorisation de Crout** par blocs (sans pivotage). Ce solveur est paginé, il peut donc s'exécuter avec peu de mémoire.

/'MUMPS' [**DEFAUT**] **Solveur direct de type multifrontale** avec pivotage. Ce solveur appelle la bibliothèque MUMPS développée par CERFACS/CNRS/ENS Lyon/INPT/INRIA/Université de Bordeaux.
Permet de traiter les modèles conduisant à des matrices non définies positives (hors conditions aux limites). Par exemple, les éléments "mixtes" ayant des degrés de liberté de type "Lagrange" (éléments incompressibles, etc).
Cette méthode est parallélisée principalement en mémoire distribuée (MPI) mais certaines de ses étapes peuvent aussi bénéficier d'un parallélisme en mémoire partagée (OpenMP). Elle peut être exécutée sur plusieurs cœurs eux-mêmes éventuellement distribués sur plusieurs nœuds.
MUMPS est fourni avec des rénuméroteurs externes en 64 bits. Cela permet d'étendre le périmètre d'utilisation de code_aster aux très gros modèles éléments finis (> 10^7 degrés de liberté). Attention toutefois, la résolution des systèmes linéaires associés à ces modèles nécessite de disposer de grandes ressources de calcul et d'activer les différents niveaux de parallélisme voire une des options d'accélération.

#Solveurs itératifs

/'GCPC' **Solveur itératif de type gradient conjugué avec préconditionnement** par une factorisation incomplète à *k* niveaux ou bien complète en simple précision.

/'PETSC' **Solveurs itératifs issus de la bibliothèque PETSc** (*Argonne National Laboratory*) avec différents préconditionneurs.
Cette méthode est parallélisée en mémoire distribuée (MPI) et peut être exécutée sur plusieurs processeurs.
Attention : les solveurs PETSC et MUMPS étant incompatibles en séquentiel, PETSC n'est pas disponible dans les versions séquentielles de code_aster. Pour utiliser PETSC, il faut donc lancer une version parallèle de code_aster (quitte à ne solliciter qu'un seul processeur).

Conseil :

La méthode par défaut est MUMPS. Elle permet, à la fois, de pleinement bénéficier des gains en temps et mémoire que procure le **parallélisme,** et, de résoudre des **problèmes numériquement difficiles** (X-FEM, incompressibilité, THM...).
Pour résoudre plus efficacement un problème de **grande taille** (> 10^6 degrés de liberté), on peut avoir recours aux compressions 'low-rank' de MUMPS (cf. mots-clés ACCELERATION/LOW_RANK_SEUIL) ou, si le périmètre fonctionnel le permet, aux solveurs itératifs PETSC ou GCPC.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Pour plus de détails et de conseils sur l'emploi des solveurs linéaires on pourra consulter les notices d'utilisation spécifique [U2.08.03] et [U2.08.06].

3.2 Paramètres communs à plusieurs solveurs

```
◊   NPREC =            /  nprec
                       /  8      [DEFAUT]

◊  STOP_SINGULIER  =   /  'OUI'  [DEFAUT]
                       /  'NON'
```

Ces deux paramètres sont communs à tous les solveurs linéaires directs (LDLT, MULT_FRONT, MUMPS).

Ils servent à contrôler le déroulement de la factorisation numérique et la qualité de la solution du système linéaire. La factorisation numérique d'une matrice peut échouer dans deux cas de figures : problème de construction de la factorisée (matrice structurellement ou numériquement singulière) et détection numérique d'une singularité (solution du système linéaire instable).
Les mot-clés NPREC et STOP_SINGULIER permettent de fixer le seuil de détection des singularités et le comportement à adopter en cas d'échec lors de la factorisation.

nprec sert à calibrer le processus de détection de singularité de la matrice du système à résoudre. Avec LDLT et MULT_FRONT, on prend la valeur absolue de nprec, avec MUMPS, on prend nprec car son signe a une importance : si $nprec<0$, on désactive la détection de singularité, sinon on l'active.
Dans tous les cas, si la valeur nprec est laissée à zéro on l'initialise à la valeur par défaut (8).
En initialisant ce paramètre à une valeur assez faible (1 ou 2) (respectivement forte, par exemple, 20), la détection de singularité se déclenchera très souvent (respectivement rarement).

Pour LDLT , MULT_FRONT :
Lorsqu'au terme de la factorisation, on constate qu'un terme diagonal d' est devenu très petit (par rapport à ce qu'il était avant la factorisation d), c'est que la matrice est (probablement) presque singulière. Soit $n=\log\left|\frac{d}{d'}\right|$, ce rapport de magnitude indique que sur une équation (au moins) on a perdu n chiffres significatifs.

Si $n>nprec$, on considère que la matrice est singulière. Si l'utilisateur a indiqué :

STOP_SINGULIER='OUI'	Le code s'arrête alors en erreur.
STOP_SINGULIER='NON'	L'exécution se poursuit après émission d'une ALARME. La qualité de la solution n'est alors pas garantie. Ce paramétrage n'est pas conseillé. En non linéaire, ce n'est pas forcément trop préjudiciable à la qualité des résultats car ceux-ci sont « corrigés » par le processus de Newton.

Pour MUMPS :

Si pour au moins un pivot, la norme infinie de la ligne (ou de la colonne) est inférieure au seuil 10^{-nprec} alors la matrice est considérée comme singulière.

On compare quelques aspects des deux types de critères de détection de singularité dans la documentation [U2.08.03].

Remarques:

- *Toute perte importante de chiffres significatifs lors d'une factorisation est un indicateur d'un problème mal posé. Plusieurs causes sont possibles (liste non exhaustive) : des conditions aux*

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

limites de blocage de la structure insuffisantes, des relations linéaires redondantes, des données numériques très hétérogènes (termes de pénalisation trop grands)...
• Pour LDLT et MULT_FRONT, la détection de singularité est faite tout le temps car elle est très peu coûteuse.
• Concernant MUMPS, un mécanisme permet de vérifier la qualité de la solution par ailleurs (RESI_RELA). On a donc laissé la liberté de désactiver ce critère (en choisissant un nprec négatif).
• Par défaut, avec le solveur direct MUMPS, on a donc un double contrôle de la qualité de la solution : en linéaire, RESI_RELA et NPREC, en non linéaire, le critère de Newton et NPREC. Il est possible de les débrancher, mais ce n'est pas conseillé sans raison valable.

◊ ELIM_LAGR = 'NON'/'OUI'/'LAGR2'

Ce mot clé permet d'éliminer les équations de Lagrange correspondant aux conditions cinématiques dualisées.

Par défaut (sauf pour MUMPS), ces équations ne sont pas éliminées ('NON').

La technique d'élimination utilisée pour ELIM_LAGR='OUI' est décrite dans [R3.03.05].
Pour utiliser ELIM_LAGR='OUI' sur plusieurs processeurs, il est impératif d'avoir au préalable choisi DISTRIBUTION='CENTRALISE' dans AFFE_MODELE. La phase d'élimination est alors répliquée sur tous les processeurs (sans entraîner de gain en temps). La résolution du système linéaire résultant s'effectue ensuite en parallèle sur tous les processeurs (cette fois avec un gain en temps, qui dépend du solveur linéaire choisi).
On ne peut pas utiliser ELIM_LAGR='OUI' avec le solveur linéaire direct 'MULT_FRONT'.

Dans le cas du solveur MUMPS, une troisième valeur est possible : 'LAGR2'. L'objectif est alors de supprimer la deuxième équation de Lagrange, mais de conserver la première.
La valeur 'LAGR2' est le défaut pour le solveur MUMPS.

Ce paramètre peut être temporairement désactivé par le code pour ne pas fausser le calcul du déterminant de la matrice. Cette fonctionnalité est principalement requise par les opérateurs CALC_MODES avec OPTION parmi ['PROCHE','AJUSTE','SEPARE'] et INFO_MODE. L'utilisateur est alors averti de ce changement automatique de paramétrage *via* un message dédié (uniquement en INFO=2).

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

3.3 METHODE='MULT_FRONT'

Périmètre d'utilisation:
Solveur robuste, à déconseiller toutefois pour les modélisations nécessitant du pivotage (éléments finis mixtes, incompressibles…), pour les matrices généralisées avec liaisons (opérateurs ASSE_ELEM/MATR_SSD …) ainsi que sur les gros modèles éléments finis (> 10^6 degrés de liberté).
Dans ces cas de figures, utilisez plutôt la méthode MUMPS (cf. § 3.5) ou, en non linéaire, PETSC + PRE_COND='LDLT_SP' (cf. § 3.7).

◊ RENUM =

Cet argument permet de numéroter les nœuds du modèle pour diminuer la taille de la factorisée (et donc les consommations CPU et mémoire de la résolution) :

/'METIS' [**DEFAUT**] Méthode de numérotation basée sur une dissection emboîtée. On utilise le produit externe du même nom qui est un standard mondial dans le domaine. C'est, en général, la méthode la plus efficace (en temps CPU et en mémoire).

/'MD' ('Minimum Degré') cette numérotation des nœuds minimise le remplissage de la matrice lors de sa factorisation.

/'MDA' ('Minimum Degré Approché') cette numérotation est en principe moins optimale que 'MD' en ce qui concerne le remplissage mais elle est plus économique à calculer. Elle est toutefois préférable à 'MD' pour les gros modèles ($\geq 50\,000$ degrés de liberté).

Remarque:

- *Dans le cas de matrices généralisées[1] comportant des contraintes de liaison, MULT_FRONT n'applique pas de renumérotation. Cette stratégie n'est pas préjudiciable car ces matrices sont souvent quasi-pleines et de petites tailles. Le choix du renuméroteur opéré par l'utilisateur est donc ignoré. Un message informatif signale ce cas de figure dans le fichier message.*

1 Générées par l'opérateur NUME_DDL_GENE.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

3.4 METHODE='LDLT'

Périmètre d'utilisation:
Solveur universel mais uniquement sur des petits modèles éléments finis (< 10^5 ddls) . Au delà, la méthode est très lente.
À déconseiller pour les modélisations nécessitant du pivotage (éléments finis mixtes, incompressible...).

Remarque:

- *La matrice est systématiquement renumérotée à l'aide de l'algorithme Reverse-Cuthill-Mackee. L'utilisateur ne peut pas modifier ce choix.*

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

3.5 METHODE='MUMPS'

Périmètre d'utilisation:
Solveur universel performant et robuste. À utiliser sur tous types de problèmes et notamment les gros modèles éléments finis (>10^6 degrés de liberté), voire les très gros modèles (>10^7 degrés de liberté). Surtout s'ils mélangent les modélisations, les types d'éléments finis ou si les systèmes linéaires associés nécessitent du pivotage (éléments finis mixtes, incompressibles, modélisation X-FEM, condition aux limites dualisées, liaisons entre groupe de mailles, etc).
En non linéaire, il est souvent préférable d'utiliser MUMPS en tant que préconditionneur, *via* l'option PETSC + PRE_COND='LDLT_SP'/ 'LDLT_DP'(cf. § 3.7).

Le solveur MUMPS, actuellement développé par CERFACS/CNRS/ENS Lyon/INPT/INRIA/Université de Bordeaux, est un solveur direct de type multifrontal parallélisé (en MPI et en OpenMP). Il est robuste car il permet de pivoter les lignes et colonnes de la matrice lors de la factorisation numérique.
Seules les versions publiques de MUMPS v5.2.1 et v5.2.1 sont acceptées dans le couplage avec *code_aster* .
D'autre part, leurs versions en accès restreint MUMPS v5. 2 . 1 consortium/v5. 4 . 1 consortium sont elles aussi acceptées mais uniquement pour des usages EDF. Elles procurent un accès, en avance de phase par rapport à la version publique, à des fonctionnalités exploratoires.
Toutes sont désormais compilées avec des renuméroteurs externes 64 bits ((PAR)METIS, (PT)SCOTCH et PORD), ce qui permet à MUMPS, en les activant (cf. mot-clé RENUM), de traiter de très gros modèles éléments finis (millions de mailles).
Attention toutefois, la résolution des systèmes linéaires associés à ces modèles nécessite de disposer de grandes ressources de calcul et d'activer les différents niveaux de parallélisme voire une des options d'accélération (cf. mot-clé ACCELERATION).
Pour plus d'informations on pourra consulter le §4 de [U2.08.03] et [U2.08.06].

3.5.1 Paramètres fonctionnels

◊ TYPE_RESOL =

Ce mot clé permet de choisir le type de résolution MUMPS :

/'NONSYM'	Doit être choisi pour les matrices non symétriques.
/'SYMGEN'	Doit être choisi pour les matrices symétriques non définies positives. C'est le cas le plus général dans *code_aster* du fait de la dualisation des conditions aux limites par des coefficients de Lagrange.
/'SYMDEF'	Peut être choisi pour les matrices symétriques définies positives. Il n'y a pas de pivotage. L'algorithme est plus rapide et moins coûteux en mémoire.
/'AUTO' [**DEFAUT**]	Le code choisira ' NONSYM ' pour les matrices non symétriques et ' SYMGEN ' pour les matrices symétriques.

Il n'est pas interdit de choisir 'NONSYM' pour une matrice symétrique. Cela doublera probablement le coût de calcul mais cette option donne à MUMPS plus de possibilités algorithmiques (pivotage, *scaling*…). *A contrario*, il peut être intéressant, en non linéaire, de symétriser son problème non symétrique (cf. [U4.51.03], mot-clé MATR_RIGI_SYME). C'est le même type d'astuce que pour les paramètres de relaxation FILTRAGE_MATRICE et MIXER_PRECISION.

◊ RESI_RELA = / resi
/ 1.d-6 [**DEFAUT**] en linéaire
/ -1.d0 [**DEFAUT**] en non linéaire et en calcul modal.

Ce paramètre est désactivé par une valeur négative. Il est appelable dans les opérateurs qui peuvent avoir besoin de contrôler la qualité d'une résolution complète de système linéaire (donc pas les opérateurs éclatés effectuant juste des factorisations ; Par exemple FACTORISER et INFO_MODE).

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

En précisant une valeur strictement positive à ce mot-clé (par exemple 10^{-6}), l'utilisateur indique qu'il souhaite tester la validité de la solution de chaque système linéaire résolu par MUMPS (en relatif par rapport à la solution exacte).
Cette démarche prudente est conseillée lorsque la solution n'est pas elle même corrigée par un autre processus algorithmique (algorithme de Newton, détection de singularité…) bref dans les opérateurs linéaires THER_LINEAIRE et MECA_STATIQUE. En non linéaire ou en calcul modal, le critère de détection de singularité et la correction de l'algorithme englobant (Newton ou solveur modal) sont des garde-fous suffisants. On peut donc débrancher ce processus de contrôle (c'est ce qui est fait par défaut *via* la valeur -1) et ce, d'autant plus qu'il a un coût en temps non négligeable et que celui-ci est plus important (en relatif) en parallèle et/ou en gestion mémoire avec déchargement des gros objets sur disque (cf. mot-clé GESTION_MEMOIRE). C'est une fonctionnalité supplémentaire que n'offrent pas les autres solveurs directs de *code_aster*.

Si l'erreur relative (basée sur les conditionnements et les erreurs inverses du système linéaire traité) sur la solution estimée par MUMPS est supérieure à resi le code s'arrête en ERREUR_FATALE, en précisant la nature du problème et les valeurs incriminées.
L'activation de ce mot-clé initie aussi un processus de raffinement itératif (sauf si POSTTRAITEMENTS='SANS') dont l'objectif est d'améliorer la solution obtenue. Ce posttraitement bénéficie d'un paramétrage particulier (mot-clé POSTTRAITEMENTS). C'est la solution résultante de ce processus d'amélioration itérative qui est testée par RESI_RELA.

Remarque:

• Dans le cas particulier où POSTTRAITEMENTS='MINI' et RESI_RELA>0 , l'estimation de la qualité effectuée par MUMPS n'est que partielle (uniquement basée sur les erreurs inverses) et donc code_aster n'arrête pas le calcul si cette valeur est supérieure à resi . Cette combinaison de mots-clés n'a d'intérêt qu'avec INFO=2 pour expertiser les qualités des solutions.

3.5.2 Paramètres de relaxation

```
◊   FILTRAGE_MATRICE=   /   filtma
                        /   -1.d0   [ DEFAUT ]
◊   MIXER_PRECISION =   /   'OUI'
                        /   'NON'   [DEFAUT]
```

Ces paramètres sont réservés au non linéaire quasi-statique. **Une valeur négative de filtma désactive la fonctionnalité.**

Ces fonctionnalités permettent de «relaxer» les résolutions effectuées avec MUMPS afin de gagner en performance. L'idée est simple. En non linéaire, le calcul de la matrice tangente peut être entaché d'erreur. Cela va probablement ralentir le processus de Newton (en nombre d'itérations), mais si la manipulation de cette matrice approximée est moins coûteuse, on peut globalement gagner en temps (moins d'opérations flottantes), en consommation mémoire (RAM voire disque si l'OOC est activé) et en bande passante (effet cache, volume d'I/O).
Ainsi, l'activation de la fonctionnalité FILTRAGE_MATRICE, avec une valeur de filtma>0, conduit *code_aster* a ne fournir à MUMPS que les termes matriciels vérifiant

$$|\mathbf{K}_{ij}| > \texttt{filtma} \cdot \left(|\mathbf{K}_{ii}| + |\mathbf{K}_{jj}| \right)$$

Le filtre est donc basé sur un seuil relatif par rapport aux valeurs absolues des termes diagonaux correspondant.
En initialisant MIXER_PRECISION à 'OUI', on utilise la version simple précision de MUMPS en lui fournissant une matrice *Aster* double précision (éventuellement filtrée *via* FILTRAGE_MATRICE). D'où potentiellement des gains en mémoire (souvent 50%) et en temps au niveau de la résolution. Cependant, cette astuce n'est vraiment payante que si la matrice tangente est bien conditionnée ($\eta(\mathbf{K}) < 10^{+6}$). Sinon la résolution du système linéaire est trop imprécise et l'algorithme non linéaire risque de ne plus converger.

Remarques:

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

• Ces paramètres de relaxation des résolutions de systèmes linéaires via MUMPS sont dans la lignée de ceux qui existent déjà pour les solveurs non linéaires (mot-clés NEWTON/REAC_ITER, MATRICE ...). Ces familles de paramètres sont clairement complémentaires et elles peuvent permettre de gagner des dizaines de pour-cents en consommation CPU et RAM. Passer un peu de temps à les calibrer, sur un premier jeu de données, peut être payant lorsqu'on doit par la suite effectuer de nombreux calculs similaires.
• Cette idée a été reprise avec le préconditionneur LDLT_SP de GCPC/PETSC.
• Mais pour gagner de la place mémoire sans risquer de perdre en précision de calcul, on peut aussi s'intéresser aux éléments suivants: calcul parallèle[U2.08.03], paramètres GESTION_MEMOIRE, MATR_DISTRIBUEE voire *RENUM.*

3.5.3 Paramètres numériques

◊ PRETRAITEMENTS =

Ce mot clé permet de contrôler le type de pré-traitement à opérer au système pour améliorer sa résolution (diverses stratégies d'équilibrage des termes de la matrice et de permutation de ses lignes et de ses colonnes) :

/'SANS'	Pas de prétraitement.
/'AUTO' [**DEFAUT**]	MUMPS choisit la meilleure combinaison de paramètres en fonction de la situation.

◊ RENUM =

Ce mot clé permet de contrôler la renumérotation et l'ordre d'élimination. Les différents outils proposés ne sont pas forcément tous disponibles. Cela dépend de l'installation de MUMPS/ code_aster. Ces outils se décomposent en deux catégories: les outils « basiques » dédiés à un usage et fournis avec MUMPS ('AMD', 'AMF', 'QAMD', 'PORD'), et, les bibliothèques plus « riches » et plus « sophistiquées » qu'il faut installer séparément ('METIS/PARMETIS', 'SCOTCH/PTSCOTCH').

/'AMD'	'Approximate Minimum Degree' (Minimum Degré Approché)
/'AMF'	'Approximate Minimum Fill' (Remplissage Minimum Approché)
/'QAMD'	Variante de 'AMD' (détection automatique de ligne quasi-dense)
/'PORD'	Outil externe de renumérotation distribué avec MUMPS.
/'METIS'	Outil externe de renumérotation (disponible aussi avec METHODE='MULT_FRONT'). C'est le renuméroteur de référence depuis la fin des années 90. Il est mondialement reconnu et utilisé.
/'PARMETIS'	Version parallèle MPI de l'outil précédent. Uniquement disponible avec la version MPI de code_aster. Cf. [U2.08.03]. À privilégier sur les très gros modèles éléments finis (millions de mailles).
/'SCOTCH'	Outil externe de renumérotation qui tend à supplanter l'outil de référence dans le domaine (METIS).
/'PTSCOTCH'	Version parallèle MPI de l'outil précédent. Uniquement disponible avec la version MPI de code_aster. Cf. [U2.08.03]. À privilégier sur les très gros modèles éléments finis (millions de mailles).
/'AUTO' [**DEFAUT**]	MUMPS choisit la meilleure combinaison de paramètres en fonction du problème et des packages disponibles. Si l'utilisateur spécifie un renuméroteur particulier et que ce dernier n'est pas disponible, le solveur choisit le plus adéquat dans la liste des disponibles et une ALARME est émise.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Remarque:

• Le choix du renuméroteur a une grande importance sur les consommations mémoire et temps du solveur linéaire. Si on cherche a optimiser/régler les paramètres numériques liés au solveur linéaire, ce paramètre doit être un des premiers à essayer.

• Les renuméroteurs (PAR)METIS, (PT)SCOTCH et PORD sont compilés en 64 bits afin de permettre à MUMPS de résoudre des systèmes linéaires de plusieurs dizaines de millions d'inconnues.

◊ POSTTRAITEMENTS =

Ce paramêtre est appelable dans les opérateurs qui peuvent avoir besoin de contrôler la qualité d'une résolution complète de système linéaire (donc pas les opérateurs éclatés effectuant juste des factorisations, par ex. FACTORISER et INFO_MODE). Son intérêt est moindre avec les opérateurs non-linéaires (STAT_NON_LINE ...) sauf en cas de résolutions difficiles.

Ce mot clé permet de contrôler deux choses :
– la procédure de raffinement itératif dont l'objectif est d'améliorer la qualité de la solution (cf. mot-clé RESI_RELA),
– l'estimation partielle (valeur 'MINI') ou complète (les autres valeurs) par MUMPS de la qualité de la solution. L'arrêt du calcul par *code_aster* ne s'effectuera que si cette estimation est complète (donc pas avec 'MINI') et que si sa valeur est supérieure au critère renseigné dans RESI_RELA.

/'SANS'	Désactivation.
/'FORCE'	**Ce paramètre n'est utilisé que si RESI_RELA est activé.** MUMPS effectue au moins une itération de raffinement itératif car son critère d'arrêt est initialisé à une valeur très faible. Le nombre d'itérations est borné à 10. Il effectue un diagnostic complet de la qualité de la solution *via* le calcul assez coûteux d'estimation des conditionnements et des erreurs inverses du système linéaire creux traité.
/'AUTO' [**DEFAUT**]	**Ce paramètre n'est utilisé que si RESI_RELA est activé.** MUMPS effectue souvent une itération de raffinement itératif. Son critère d'arrêt est proche de la précision machine et le nombre d'itérations est borné à 4. Il effectue un diagnostic complet de la qualité de la solution *via* le calcul assez coûteux d'estimation des conditionnements et des erreurs inverses du système linéaire creux traité.
/'MINI'	MUMPS effectue exactement deux itérations de raffinement itératif. A l'usage on note que c'est souvent suffisant pour résoudre efficacement beaucoup des systèmes linéaires. Cette option est à privilégier en cas de recherche de performance, sans perte de précision, lors de l'activation des compressions low-rank (cf. mot-clé ACCELERATION). Par défaut ici MUMPS n'effectue pas de diagnostic de la qualité de la solution. On effectue seulement un diagnostic partiel, *via* le seul calcul (peu coûteux) des erreurs inverses, si RESI_RELA>0 (utile seulement pour expertise).

Remarques:

• Ce processus consomme principalement des descentes-remontées dont le surcoût en temps est encore raisonnable en In-Core séquentiel. Par contre, il peut être important en Out-Of-Core et en parallèle, voire peu utile en non linéaire (l'algorithme de Newton corrige).

• Pour limiter toute dérive contre-productive, avec les valeurs 'AUTO' et 'FORCE' la procédure de raffinement itératif est bridée en interne MUMPS : dès qu'une itération ne procure pas un gain d'au moins un facteur 5, le processus s'arrête. Le nombre d'itérations généralement constaté (en posant INFO=2) est 1 ou 2.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

• Sur certains cas-tests mal conditionnés (par exemple perf001e), le forçage de ce processus a permis d'atteindre la précision souhaitée (via les valeurs 'FORCE' ou 'MINI').

3.5.4 Paramètres pour la gestion mémoire

Pour gagner en mémoire RAM sans changer de solveur linéaire (et de modélisation ou de plate-forme informatique), plusieurs stratégies sont disponibles (et souvent combinables). On les liste ici par ordre d'importance.

- **À précision numérique constante et avec des gains en temps de calcul:**
  Le parallélisme (menu Options/mpi_** d'Astk) couplé, ou non, avec l'activation du mot-clé MATR_DISTRIBUEE en mode parallélisme distribué (mode par défaut) ;
  L'activation des prétraitements (fait par défaut, cf. PRETRAITEMENTS) ;
  La redistribution du parallélisme entre le niveau MPI et celui OpenMP (cf. REDUCTION_MPI).

- **À précision numérique constante mais avec, potentiellement, des pertes en temps de calcul:**
  L'activation explicite des facultés de déchargement sur disque de MUMPS (cf. GESTION_MEMOIRE),
  Le changement de renuméroteur (cf. mot-clé RENUM vu précédemment).

  Il faut aussi veiller à provisionner un espace supplémentaire réservé au pivotage de taille raisonnable : mot-clé PCENT_PIVOT. **Souvent les valeurs par défaut de ces paramètres (GESTION_MEMOIRE='AUTO', RENUM='AUTO' et PCENT_PIVOT=20) procurent les meilleurs compromis** pour adapter cette partie du paramétrage au cas de figure.

- **En acceptant une perte de précision** au sein d'un processus non linéaire (par ex. STAT ou DYNA_NON_LINE, CALC_MODES, ...) : tous les paramètres de relaxation liés au solveur (FILTRAGE_MATRICE, MIXER_PRECISION vus précédemment) voire ceux liés au processus non linéaire proprement dit (matrice tangente élastique, espace de projection en calcul modal...).

- En n'utilisant plus MUMPS comme solveur direct mais comme préconditionneur (cf. GCPC ou PETSC avec REAC_PRECOND='LDLT_SP/DP' cf. §3.6/3.7).

Pour plus de plus amples informations on pourra consulter les documentations [U2.08.03] (Notice d'utilisation des solveurs linéaires) et [U2.08.06] (Notice d'utilisation du parallélisme).

```
◊   PCENT_PIVOT =   /   pcent  [I]
                    /   20%    [ DEFAUT ]
```

Ce mot-clé permet de choisir un pourcentage de mémoire que MUMPS réservera en début de calcul pour le pivotage. En effet, pour factoriser une matrice *Aster*, il est souvent préférable de permuter deux de ses lignes et/ou de ses colonnes (cf. [R6.02.03] §2.3.1). Or les objets informatiques gérant ce pivotage sont difficiles à dimensionner *a priori*. C'est pour cela que l'outil demande à l'utilisateur une estimation préalable et arbitraire de cet espace supplémentaire.
La valeur par défaut est de 20%. Elle correspond à un nombre de pivotages raisonnable qui est suffisant pour la plupart des calculs *Aster*. Si par exemple MUMPS estime à 100 la place nécessaire à une factorisation sans pivotage, il allouera *in fine* 120 pour gérer le calcul avec pivotage. Une valeur dépassant les 50% doit rester exceptionnelle.

Par la suite, si l'espace mémoire requis par les pivotages s'avère plus important, la place allouée sera insuffisante et le code demandera d'augmenter ce critère. Deux cas de figures se présenteront alors suivant le type de gestion mémoire choisi (*via* le mot-clé GESTION_MEMOIRE) :

- S'il s'agit d'un mode précis, 'IN-CORE' ou 'OUT_OF_CORE', le calcul s'arrête en ERREUR_FATALE et propose différentes solutions palliatives.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

- S'il s'agit du mode automatique, 'AUTO', le calcul va se poursuivre et retenter de factoriser avec un valeur de PCENT_PIVOT doublée. Jusqu'à trois tentatives de ce type seront effectuées avant, en cas d'échecs répétés, un arrêt en ERREUR_FATALE + proposition de différentes solutions palliatives.

Remarques:

• Pour les petits problèmes (<1000 ddls), MUMPS peut sous-estimer ses besoins en pré-allocations d'espace mémoire. Une grande valeur de PCENT_PIVOT (>100) n'est alors pas surprenante.

• Processus d'auto-apprentissage: si, dans le processus décrit précédemment, on est amené à modifier automatiquement la valeur de PCENT_PIVOT , c'est cette nouvelle valeur qui est utilisée jusqu'à la fin de l'opérateur. On suppose que la difficulté numérique ne va pas diminuer et on conserve donc cette valeur de pivotage afin de ne plus perdre de temps en tentatives avortées de factorisation.

• En mode ' AUTO ', conjointement au doublement de l'espace supplémentaire de pivotage, on peut aussi être amené à passer automatiquement en gestion mémoire MUMPS Out-Of-Core (comme si on avait paramétré explicitement GESTION_MEMOIRE='OUT_OF_CORE'). Cela se produit suivant certains code retour MUMPS ou à la troisième (et dernière) tentative.

◊ GESTION_MEMOIRE=

Ce mot-clé permet de choisir le mode de gestion mémoire du produit externe MUMPS, voire en dernier ressort, de certains objets gérés directement par *code_aster*.
Les deux premiers modes sont « sans filet »: aucune correction de paramétrage ne sera opérée « à la volée » en cas de problème. Au contraire du 3 ieme mode, le mode automatique, qui va tout faire (dans certaines limites !) pour que le calcul n'achoppe pas pour des raisons de place mémoire. En particulier, suivant la mémoire qu'il arrivera à dégager par ailleurs, il va jouer sur les modes In-Core et Out-Of-Core de MUMPS voire sur l'espace requit pour son pivotage (cf. mot-clé PCENT_PIVOT).

/'IN_CORE'	On privilégie au maximum la vitesse du calcul. C'est l'option qui requiert le plus de mémoire, car ici on permet à MUMPS de conserver en RAM tous les objets dont il a besoin.
/'OUT_OF_CORE'	On privilégie au maximum les économies en consommation mémoire. C'est l'option qui requiert le moins de mémoire, car ici on impose à MUMPS de décharger sur disque ses objets les plus encombrants[2].
/'AUTO' [**DEFAUT**]	On décide automatiquement de la gestion mémoire à imposer à MUMPS (cf. In-Core ou Out-Of-Core précédent) en fonction des capacités mémoire disponibles à ce moment précis du calcul. On active aussi un mécanisme de pré-allocation mémoire afin que MUMPS puisse profiter du maximum de la mémoire disponible (cf. paragraphe ci-dessous). Cela permet de limiter les problèmes d'allocations tardives pour assurer le pivotage. Deux mécanismes de correction automatiques peuvent aussi se mettre en place si nécessaire (augmentation du PCENT_PIVOT, débranchement pré-allocations mémoire).
/'EVAL'	Aide à la calibration mémoire du calcul. On fournit un affichage synthétique (cf. figure 3.1) des ressources mémoires requises par le calcul *code_aster* + MUMPS[3] suivant le type de gestion choisi: In-Core ou Out-Of-Core. Ensuite le calcul s'arrête en ERREUR_FATALE afin de permettre à l'utilisateur de relancer son calcul en choisissant un paramétrage mémoire s'appuyant sur ces éléments.

2 Les blocs de factorisées (réelles ou complexes) gérés par le processeur courant. Les vecteurs d'indices (entiers) décrivant ces blocs restent eux en RAM.

3 Les estimées MUMPS peuvent être légèrement surévaluées. Elles récapitulent les chiffres requis pour une utilisation en « stand-alone » du produit MUMPS sur le problème issu de *code_aster*. Ces estimations intègrent donc, non seulement les objets dont va avoir besoin MUMPS pour construire sa factorisée, mais aussi les objets préalables de stockage des données (matrice, RHS) et un 30 Mo forfaitaire pour l'exécutable MUMPS.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
*******************************************************************************
 - Taille du système linéaire: 500000

 - Mémoire RAM minimale consommée par code_aster                          : 200 Mo
 - Estimation de la mémoire Mumps avec GESTION_MEMOIRE='IN_CORE'          :3500 Mo
 - Estimation de la mémoire Mumps avec GESTION_MEMOIRE='OUT_OF_CORE'      : 500 Mo
 - Estimation de l'espace disque pour Mumps avec GESTION_MEMOIRE='OUT_OF_CORE':2000 Mo

  ===> Pour ce calcul, il faut donc une quantité de mémoire RAM au minimum de
        - 3500 Mo si GESTION_MEMOIRE='IN_CORE',
        -  500 Mo si GESTION_MEMOIRE='OUT_OF_CORE'.
 En cas de doute, utilisez GESTION_MEMOIRE='AUTO'.
*******************************************************************************
```

Figure 3.1._ Affichage dans le fichier message en mode 'AUTO'.

L'activation de l'Out-Of-Core contribue à réduire la mémoire RAM requise par processeur, mais cela peut ralentir le calcul (coût des I/O RAM/disque). Ce surcoût peut être notable lorsqu'on effectue de nombreuses descente-remontées (par ex. calcul non linéaire avec beaucoup de pas de temps ou d'itérations de Newton, recherche de nombreux modes propres en calcul modal...). Car dans ces étapes algorithmiques, on passe autant de temps à manipuler les données (en RAM) qu'à aller les chercher (sur disque). Ceci est d'autant plus vrai que le disque est commun à plusieurs coeurs de calcul. Pour cette raison, on privilégie au maximum le mode In-Core (surtout en parallèle).

En mode 'EVAL', les pré-estimations des consommées mémoire sont beaucoup plus rapides et moins coûteuses en mémoire que le calcul complet. Elles peuvent permettre de calibrer son étude sur machine locale ou sur un noeud interactif d'une machine centralisée avant de lancer en mode batch l'étude proprement dite.
A titre anecdotique, ce mode peut aussi servir à tester grossièrement la mise en données et/ou l'exécutable utilisé. Si tout fonctionne jusqu'à cette évaluation c'est plutôt bon signe pour le calcul ultérieur !

En mode 'AUTO', on permet à MUMPS de «s'étaler en RAM» pour gagner en temps et pour limiter les besoins tardifs en mémoire liés au pivotage. MUMPS va ainsi pouvoir prendre toute la mémoire qu'il juge nécessaire, même éventuellement au delà de ses estimées initiales. Cela lui permet de pallier d'éventuels futurs besoins. Pour ce faire, *code_aster* lui fournit une estimation de la RAM disponible.
Ces pré-allocations permettent souvent de ne plus avoir à ajuster, souvent «au doigt mouillé», le paramètre 'PCENT_PIVOT'. D'où un gain de temps certain pour la mise au point des études.
D'autre part, en mode 'AUTO', un calcul *code_aster*+MUMPS tire ainsi vraiment parti de toute la mémoire disponible: le Vmpeak est proche du chiffre paramétré dans Astk.
Par contre, dans les deux autres modes ('IN_CORE' et 'OUT_OF_CORE'), MUMPS n'a pas le droit de «s'étaler» en RAM. Il ne pré-alloue aucun espace supplémentaire au delà de ses estimées mémoire initiales. Cela permet de conserver un mode de fonctionnement sûr en cas de mauvaise évaluation de la mémoire disponible[4].
Un autre mécanisme permet aussi de pallier ce genre de désagrément: si MUMPS cherche à allouer un objet de taille supérieure à l'espace mémoire réellement disponible, on retente une nouvelle factorisation en ne lui permettant plus de pré-allouer d'espace supplémentaire. Cette stratégie corrective, similaire à celle utilisée pour le paramètre PCENT_PIVOT, n'est activée qu'avec le mode 'AUTO'.

Remarques:

• Dans les modes standards ('IN_CORE' et 'OUT_OF_CORE') code_aster décharge sur disque les plus gros objets [5] liés au système linéaire. Et ce, afin de laisser à MUMPS un maximum de place en mémoire RAM. Si par la suite MUMPS ne dispose pas de suffisamment de place pour allouer ses données, une alarme est émise et le calcul continue son déroulement. Suivant les cas, le calcul peut s'achever sans encombre mais au prix d'un gros surcoût en temps (swap système) ou s'arrêter en ERREUR_FATALE . L'utilisateur se voit alors proposer différentes alternatives dont le paramétrage en mode ' AUTO '.

4 Cela peut arriver sur certaines plate-formes (par ex. clpaster-Rocks).
5 Matrice (MATR_ASSE), description des inconnues (NUME_DDL)...

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

• En mode ' AUTO ', si cet espace libéré est insuffisant pour permettre à MUMPS de fonctionner pleinement en In-Core, on décharge sur disque tout le reliquat d'objets JEVEUX déchargeables. Puis, suivant l'espace mémoire ainsi dégagée, on active le mode In-Core ou Out-Of-Core de MUMPS ou on s'arrête en ERREUR_FATALE (+ conseils).
• Les déchargements massifs d'objets JEVEUX évoqués précédemment peuvent, dans des cas exceptionnels, ralentir grandement l'exécution. Cela peut arriver par exemple en cas d'engorgement des accès disque en mode parallèle ou lorsqu'on décharge beaucoup de données (champs aux différents pas de temps, champs projetés...). La solution peut alors être d'occuper moins de processeurs par noeud, de consommer moins de mémoire (augmenter le nombre de processeurs, mode Out-Of-Core..) ou de découper son calcul en plusieurs étapes.
• En mode 'EVAL', l'évaluation puis l'arrêt sont effectués à la première factorisation matricielle via MUMPS. Soit, par exemple, dans la phase de prédiction pour STAT_NON_LINE , ou dans le test de Sturm pour CALC_MODES. C'est souvent suffisant pour avoir un bon ordre de grandeur des besoins en mémoire. Pour éventuellement repousser cette évaluation, il faut découper son calcul et utiliser un autre solveur linéaire (par exemple ' MULT_FRONT ') pour les opérateurs que l'on souhaite préserver.

◊ MATR_DISTRIBUEE = / 'OUI'
/ 'NON' [**DEFAUT**]

Ce paramètre est pour l'instant limité aux opérateurs MECA_STATIQUE, STAT_NON_LINE et DYNA_NON_LINE et il n'est actif qu'en parallèle distribué (AFFE_MODELE/PARTITION/PARALLELISME!='CENTRALISE').
En activant ce mot-clé, le stockage de la matrice assemblée se fait de manière distribuée sur tous les processeurs (on ne stocke plus de valeurs inutiles appartenant aux autres processeurs). Cela permet d'économiser de la mémoire en parallèle sans surcoût en temps, ni perte de précision (ce mot-clé n'a aucune influence en séquentiel ou en parallèle centralisé).

3.5.5 Paramètres pour réduire le temps calcul

Pour réduire le temps calcul sans changer de solveur linéaire (et de modélisation ou de plate-forme informatique), plusieurs stratégies sont disponibles (et combinables). On les liste ici par ordre d'importance.

- **À précision numérique constante:**
  - le parallélisme MPI et/ou OpenMP ou, éventuellement, une combinaison optimisée des deux (cf. mot-clé REDUCTION_MPI);
  - l'activation de différentes accélérations et/ou compressions (cf. mot-clé ACCELERATION) ;
  - le changement de renuméroteur (mot-clé RENUM vu précédemment, souvent le choix fait par défaut est optimal);
  - l'activation des prétraitements (fait par défaut, mot-clé PRETRAITEMENTS) ;
  - l'usage de MUMPS, on pas en tant que solveur direct mais en tant que préconditionneur. Peut-être très performant, surtout en non linéaire, avec la mutualisation sur plusieurs pas de Newton du même préconditionneur (cf. PETSC ou GCPC avec PRE_COND='LDLT_SP/DP' §3.6/3.7)

- **En acceptant une perte de précision,** ce qui n'est souvent pas préjudiciable (la précision est suffisante) ou compensée par un processus non linéaire englobant (par ex. STAT ou DYNA_NON_LINE, CALC_MODES...):
  - pour problèmes de grande taille (N au moins > 2.10^6 ddls), l'activation des compressions low-rank (cf. mots-clés ACCELERATION/LOW_RANK_SEUIL ci-dessous);
  - les paramètres de relaxation liés au solveur (FILTRAGE_MATRICE, MIXER_PRECISION vus précédemment) voire ceux liés au processus non linéaire proprement dit (matrice tangente élastique, espace de projection en calcul modal...);
  - la réduction des post-traitements (mot-clé POSTTRAITEMENTS).

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Pour plus de plus amples informations on pourra consulter les documentations [U2.08.03] (Notice d'utilisation des solveurs linéaires) et [U2.08.06] (Notice d'utilisation du parallélisme).

◊ ACCELERATION= / 'AUTO' [DEFAUT]
/ 'FR' (disponible pour toutes les versions de MUMPS)
/ 'FR+' ou 'FR++' (uniquement avec les versions « consortium »)

/ 'LR' (disponible pour toutes les versions de MUMPS)
/ 'LR+'ou 'LR++' (uniquement avec les versions « consortium »)

type d'accélération: valeurs souvent efficaces, 'FR++' ou 'LR++'.

Si ACCELERATION='LR'/'LR+' ou 'LR++' :
◊ LOW_RANK_SEUIL= / 0.0 [DEFAUT]
/ lr_seuil [R]

seuil de compression: valeurs souvent efficaces entre 10^{-12} et 10^{-9} (cf. explications ci-dessous).

Ces mots-clés définissent le type d'accélération mis en œuvre pour réduire le temps calcul. Ces accélérations peuvent réduire significativement le temps de calcul de grosses études, et ce, sans restriction de périmètre d'utilisation et avec peu ou pas d'impact sur la précision, la robustesse et le comportement global de la simulation. Leur disponibilité dépend des versions de MUMPS couplées avec code_aster.
Elles sont surtout intéressantes sur des problèmes de grandes tailles (N au moins > 2.10 ^{6} ddls). Les gains constatés sur quelques études code_aster varient de 20% à 80%. Ils augmentent avec la taille du problème, son caractère massif et ils sont complémentaires de ceux procurés par le parallélisme et le renuméroteur.
Les différentes valeurs du paramètre ACCELERATION sont :

- La valeur ' AUTO ' (prise par défaut) choisit le meilleur paramétrage en fonction de la version de MUMPS disponible, du cas traité et de la configuration de calcul. A partir de MUMPS 5.4.1, pour la version publique 'AUTO'='FR' et, pour celle consortium, 'AUTO'='FR+'.
- La valeur ' FR ' permet la mise en œuvre d'une résolution MUMPS standard (dite 'Full-Rank'), c'est-à-dire sans compression 'low-rank' et sans « optimisations agressives » des options internes à MUMPS. C'est celle utilisée par défaut à partir de MUMPS 5.4.1publique.
- La valeur ' FR+ ' permet la mise en œuvre d'une résolution sans compression 'low-rank' mais en activant des « optimisations agressives » d'options internes à MUMPS. C'est celle utilisée par défaut à partir de MUMPS 5.4.1consortium.
- La valeur ' FR++ ' rajoute à 'FR+ ' une option particulière de MUMPS qui réduit le temps de son étape d'analyse.

- La valeur ' LR ' active une résolution MUMPS avec compression 'Low-Rank'. Le taux de compression est fixé par le paramètre fourni par le mot-clé LOW_RANK_SEUIL. *Grosso modo* , plus ce chiffre est grand, par exemple 10 $^{-12}$ ou 10 $^{-9}$, plus la compression va être importante et donc plus les gains en temps peuvent être intéressants. A partir d'un certain seuil de compression (donc «d'approximation»), il est conseillé d'activer, en complément, la procédure de raffinement itératif (par exemple *via* POSTTRAITEMENTS='MINI'). Celle-ci permet de retrouver, avec souvent un faible surcoût, une erreur sur la solution proche de celle que l'on aurait obtenue avec le calcul standard, ' FR '.
- Les valeurs ' LR+'/ 'LR++' activent la même option que précédemment (' LR ') mais en rajoutant les « optimisations agressives » de ' FR+ ' (resp. ' FR++') ainsi qu'une méthodologie de compression plus sophistiquée.

Pour plus de détails sur ces deux mots-clés on pourra consulter [U2.08.03] § 7.2.7.

◊ **REDUCTION_MPI** = / redmpi [I]
/ 0 [**DEFAUT**]

Ce mot-clé n'est utilisable que sur une version parallèle MPI du code. Il n'est actif que pour une valeur entière strictement positive et compatible avec le nombre de MPI activé (cf. exemple ci-dessous). Il permet de réduire, juste au niveau de MUMPS, le parallélisme de premier niveau (MPI) et de le

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

redistribuer sur celui de second niveau (OpenMP). En sortie de MUMPS, le calcul reprend la configuration parallèle fixée ai lancement de code_aster.
Ainsi, avec un calcul code_aster distribué sur 4 processus MPI[6] chacun utilisant 9 OpenMP[7] (occupant par exemple 1 nœud du cluster de calcul gaia, soit 4x9=36 cœurs) :

1. les matrices et vecteurs sont construits dans les procédures de calculs élémentaires et d'assemblages de code_aster et uniquement en parallèle MPI ; celles-ci utilisent donc seulement 4 c œ urs sur les 36 alloués ; l'accélération de ces étapes de calcul est donc au maximum de **X4** ;
2. les résolutions de systèmes linéaires MUMPS utilisent elles entièrement les 48 c œ urs grâce à aux 2 niveaux de parallélisme MPI et OpenMP. Leurs capacités d'accélération de ceux-ci sont `grosso-modo` similaires et avec des efficacités parallèles de l'ordre de 30 à 50 %. D'où une accélération potentielle de cette étape de calcul entre **X15 et X24** ;

Le fait d'activer le mot-clé `REDUCTION_MPI`, va permettre :

- d'utiliser plus de processus MPI afin de mieux accélérer la partie 1 ;
- t out en conservant le niveau d'accélération de la partie 2 et sans impacter sa consommation mémoire globale ;

Ainsi, avec **`REDUCTION_MPI=3`** sur l'exemple précédent et en distribuant cette fois le calcul code_aster , non pas sur 4 mais sur 4x3=12MPI, chacun utilisant 3 OpenMP (pour continuer à rester sur 1 nœud à 36 cœurs):

- Les partie s construction de matrices et de vecteurs peu vent être accélérées cette fois jusqu'à **X12** en occupant 12 c œ urs ;
- L a partie MUMPS reste tout aussi efficace en occupant toujours 36 c œ urs ; en distribuant cette fois le calcul sur 12/3=4 MPI, chacun utilisant 3x3=9 OpenMP. **On retrouve le point de fonctionnement initial ement cité** .
  La partie MUMPS étant la plus consommatrice en mémoire, la distribution initiale 12MPI x 3 OpenMP n'est sans doute pas possible ou difficile sur 1 seul nœud. D'où ce besoin de ré-organisation du parallélisme juste au niveau de MUMPS.

Cette redistribution du parallélisme au sein de MUMPS permet ainsi :

- **D'accélérer globalement les calculs dont les lois de comportement sont coûteuses** , et ce, sans impacter l'autre partie importante du calcul, celle de résolution de système linéaire ; n otamment en termes de consommation mémoire.
- **Eventuellement de réduire le pic mémoire global d'un calcul** (somme des pic mémoire des processus MPI) en réduisant le nombre de processus MPI dans l'étape de calcul MUMPS ; c elle-ci étant souvent celle nécessitant le plus de mémoire.

Remarque:

• Ce paramètre n'est disponible qu'avec la version consortium de MUMPS et il n'est pas activable avec l'opérateur de calcul modal `CALC_MODES` . Celui-ci ayant sa propre redistribution du parallélisme (2 niveaux MPI et 1 niveau OpenMP).

6 Paramètre `mpi_nbcpu` de l'outil de lancement.
7 Resp. `ncpus`. Ce paramétrage est normalement automatiquement déterminé par l'outil de lancement.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

3.6 METHODE='GCPC'

Périmètre d'utilisation:
Problèmes symétriques réels sauf ceux requérant obligatoirement une détection de singularité (calcul modal). En non linéaire, si le problème est réel non symétrique, on peut utiliser ce solveur pourvu que la matrice ait été rendue symétrique .

◊ PRE_COND =

Ce mot-clé permet de choisir la méthode de préconditionnement :

/'LDLT_INC' [**DEFAUT**]	Décomposition **LDL**^{T} incomplète (par niveau) de la matrice assemblée
/'LDLT_SP'	Factorisation simple précision calculée par MUMPS
/'LDLT_DP'	Factorisation double précision calculée par MUMPS

L'utilisation de 'LDLT_SP' ou 'LDLT_DP' est plus coûteuse en CPU/RAM mais plus robuste. Ces préconditionneurs sont obtenus en calculant avec la bibliothèque MUMPS une factorisation approchée de la matrice du système linéaire. Cette factorisation peut être calculée en simple (LDLT_SP) ou en double (LDLT_DP) précision. Son intérêt réside surtout dans sa mutualisation (cf. mot-clé REAC_PRECOND) pendant plusieurs résolutions si on cherche à résoudre des problèmes de type multiples seconds membres (par ex. STAT_NON_LINE ou chaînage thermo-mécanique avec MECA_STATIQUE).

◊ NIVE_REMPLISSAGE = / niv
/ 0 [**DEFAUT**]

Ce paramètre ne concerne que le préconditionneur LDLT_INC. La matrice de préconditionnement ($\mathbf{P}$) utilisée pour accélérer la convergence du gradient conjugué est obtenue en factorisant de façon plus ou moins complète la matrice initiale ($\mathbf{K}$).

Plus niv est grand, plus la matrice $\mathbf{P}$ est proche de $\mathbf{K}^{-1}$ et donc plus le gradient conjugué converge vite (en nombre d'itérations). En revanche, plus niv est grand plus le stockage de $\mathbf{P}$ devient volumineux (en mémoire et sur disque) et plus les itérations sont coûteuses en CPU.

Il est conseillé d'utiliser la valeur par défaut ($niv=0$). Si $niv=0$ ne permet pas au gradient conjugué de converger, on essaiera successivement les valeurs $niv=1,2,3\ldots$.
De même si le nombre d'itérations du gradient conjugué est jugé trop important, il est souvent bénéfique d'augmenter le niveau de remplissage.

◊ REAC_PRECOND = / reac
/ 30 [**DEFAUT**]

Ce paramètre ne concerne que les préconditionneurs LDLT_SP et LDLT_DP
Ces préconditionneurs sont beaucoup plus coûteux à construire que le préconditionneur incomplet mais ils sont plus robustes. Pour le rendre vraiment compétitif par rapport au solveurs directs classiques (MULT_FRONT ou MUMPS double précision), il faut le conserver pendant plusieurs résolutions successives. On joue ainsi sur la «relative proximité» de ces itérés successifs. Pour ce faire, le paramètre REAC_PRECOND conditionne le nombre de fois où l'on garde le même préconditionneur alors que la matrice du problème a changé. Tant que la méthode itérative GCPC prend moins de reac itérations pour converger, on conserve le préconditionneur inchangé ; si elle dépasse ce nombre, on réactualise le préconditionneur en recalculant une factorisation.

◊ PCENT_PIVOT = / pcent

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
                    /   20 [DEFAUT]
◊  GESTION_MEMOIRE  =   /   'AUTO',             [DEFAUT]
                    /   'IN_CORE'
◊   LOW_RANK_SEUIL   =  /   0.0 ,              [DEFAUT]
                    /    lr_seuil            [R]
◊  RENUM  =         /   'SANS',             [DEFAUT]
                    /   ' PARMETIS '
                    /   ' METIS ',
```

Ces paramètres ne concernent que les préconditionneurs LDLT_SP et LDLT_DP. Le mot clé ' LOW_RANK_SEUIL ' permet de régler la précision de la compression block-low-rank utilisée par MUMPS. Plus ce paramètre est grand, plus la compression est sévère et moins la factorisation est fidèle.
Il s'agit des mêmes mots-clé que pour le solveur MUMPS, *cf.* §3.5.4.

```
◊   NMAX_ITER   =   /   niter
                    /   0   [DEFAUT]
```

Nombre d'itérations maximum de l'algorithme de résolution itératif. Si $niter=0$ alors le nombre maximum d'itérations est calculé comme suit :

$niter=nequ/2$ où $nequ$ est le nombre d'équations du système.

```
◊   RESI_RELA   =   /   resi
                    /   10-6        [DEFAUT]
```

Critère de convergence de l'algorithme. C'est un critère relatif sur le résidu non-préconditionné :

$$\frac{\|\mathbf{r}_m\|}{\|\mathbf{f}\|}\leq resi$$

$\mathbf{r}_m$ est le résidu non préconditionné à l'itération m
$\mathbf{f}$ est le second membre et la norme $\|\ \|$ est la norme euclidienne usuelle.

Remarque:

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

• *Lorsqu'on utilise le préconditionneur LDLT_INC , la matrice est systématiquement renumérotée à l'aide de l'algorithme Reverse-Cuthill-Mackee. L'utilisateur ne peut pas modifier ce choix.*

3.7 METHODE='PETSC'

Périmètre d'utilisation:
Tous types de problèmes réels sauf ceux requérant obligatoirement une détection de singularité (calcul modal). À utiliser en priorité sur les problèmes non linéaires (avec PRE_COND='LDLT_SP') ou sur les problèmes « frontières » (> 5.10^7 degrés de liberté).

Attention : les solveurs PETSC et MUMPS étant incompatibles en séquentiel, seul MUMPS est disponible dans les versions séquentielles de code_aster. Pour utiliser PETSC, il faut donc toujours lancer une version parallèle de code_aster (quitte à ne solliciter qu'un seul processeur).

◊ ALGORITHME =

Nom des solveurs itératifs (de type Krylov) de PETSc accessibles depuis *code_aster* :

/'FGMRES' [**DEFAUT**]	'Flexible Generalised Minimal RESidual'
/'GMRES'	'Generalised Minimal RESidual'
/'GMRES_LMP'	'Generalised Minimal RESidual', avec préconditionneur de second niveau à mémoire limitée (Limited Memory Preconditioner)
/'CG'	Gradient conjugué
/'CR'	Résidu conjugué
/'GCR'	'Generalised Conjugate Residual'

La méthode par défaut assure le meilleur rapport entre robustesse et coût du calcul.
Les méthodes 'CG' et 'CR' sont à réserver aux modélisations conduisant à des matrices symétriques. En non symétrique, outre 'GMRES', on peut faire appel à 'GCR' qui traite des matrices quelconques.
L'algorithme 'GMRES_LMP' s'appuie sur le solveur itératif 'GMRES'. Il doit être obligatoirement utilisé avec le préconditionneur de premier niveau 'LDLT_SP'. Son utilisation est intéressante dans un calcul non-linéaire : en effet, le préconditionneur de second niveau améliore le préconditionnement d'un système à partir d'informations spectrales issues des résolutions linéaires précédentes (voir [R6.01.02])

◊ PRE_COND =

Nom des préconditionneurs de PETSc accessibles depuis *code_aster :*

/'LDLT_INC'	Factorisation incomplète par niveau
/'LDLT_SP' [**DEFAUT**]	Factorisation simple précision calculée avec MUMPS
/'LDLT_ D P'	Factorisation double précision calculée avec MUMPS
/'ML'	Multigrille algébrique « *multilevel* » (bibliothèque ML)
/'BOOMER'	Multigrille algébrique « BoomerAMG » (bibliothèque HYPRE)
/'GAMG'	Multigrille algébrique (bibliothèque PETSc)
/'BLOC_LAGR'	Préconditionneur par blocs de type Lagrangien Augmenté
/'JACOBI'	Pré-conditionneur diagonal standard
/'SOR'	'Successive Over Relaxation'
/'FIELDSPLIT'	Préconditionneur par blocs (fonctionnalité avancée)

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

/'UTILISATEUR'	Préconditionneur en Python (fonctionnalité avancée)
/'SANS'	Pas de préconditionneur

Seuls LDLT_SP, LDLT_DP, ML, BOOMER et JACOBI ont exactement le même fonctionnement en séquentiel et en parallèle. Les deux autres, 'LDLT_INC' et 'SOR', modifient un peu le calcul en utilisant des blocs diagonaux locaux aux processeurs. Ils sont plus simples à mettre en œuvre mais moins efficaces. 'SANS' permet de ne pas appliquer de préconditionneur ce qui ne présente un intérêt que lors de la mise au point d'un calcul.
Les préconditionneurs multigrilles algébriques ML, BOOMER et GAMG ont un périmètre d'application très restreint :

- calcul sans multiplicateurs de Lagrange (utiliser AFFE_CHAR_CINE pour imposer les chargements),
- avec un nombre de degrés de liberté constant par nœud.

Ils sont néanmoins très efficaces en parallèle. On notera que le pré-conditionneur ML s'appuie au cours de son algorithme sur un tirage aléatoire, ce qui peut entraîner une convergence légèrement différente entre deux résolutions identiques. Ces préconditionneurs multi-grilles sont à utiliser avec les solveurs CG ou GCR (qui peut fonctionner alors que CG échoue).

Le préconditionneur 'BLOC_LAGR' est un préconditionneur par blocs conçu pour les calculs avec multiplicateurs de Lagrange. Il doit être utilisé avec METHODE='PETSC'.

Les préconditionneurs 'LDLT_SP' et 'LDLT_DP' sont *a priori* les plus robustes, mais aussi les plus coûteux à construire. Cependant, et à l'inverse des autres préconditionneurs, ils ne sont pas reconstruits à chaque résolution linéaire, ce qui les rendent finalement compétitifs (*cf.* mot-clé REAC_PRECOND). Ces préconditionneurs sont à utiliser avec le solveur FGMRES par défaut (ou bien CG ou GCR si la matrice est symétrique). Il est préférable d'éviter GMRES (ou bien son équivalent symétrique CR) combinés à un préconditionneur en simple précision (risque d'obtenir une solution imprécise, car le critère d'arrêt du solveur est perturbé par le mélange d'arithmétiques).
Dans un calcul non-linéaire, on peut enfin utiliser 'LDLT_SP' avec l'algorithme 'GMRES_LMP' : un préconditionneur de second-niveau (LMP) améliore alors le préconditionnement d'une résolution linéaire à partir d'informations spectrales issues des résolutions linéaires précédentes (voir [R6.01.02]).

Le préconditionneur 'FIELDSPLIT' offre un cadre très général pour définir des préconditionneurs par blocs. Il s'agit d'une fonctionnalité très puissante qui nécessite une très bonne connaissance des méthodes itératives et de la librairie PETSc. Elle est essentiellement utilisée dans des actions de recherche.

Le préconditionneur 'UTILISATEUR' permet l'utilisation d'un préconditionneur défini par l'utilisateur dans une fonction, reposant sur l'interface Python de PETSc. Il s'agit d'une fonctionnalité avancée qui nécessite une très bonne connaissance des méthodes itératives et de la librairie PETSc. Elle est essentiellement utilisée dans des actions de recherche.

◊ NIVE_REMPLISSAGE = / niv
/ 0 [**DEFAUT**]

Ce paramètre ne concerne que le préconditionneur LDLT_INC.
Niveau de remplissage du préconditionneur de Cholesky Incomplet.

◊ REMPLISSAGE = / α
/ 1.0 [**DEFAUT**]

Ce paramètre ne concerne que le préconditionneur LDLT_INC.
Facteur d'accroissement de la taille du préconditionneur en fonction du niveau de remplissage (cf §3.6). La référence est fixée à $niv=0$ pour lequel $\alpha=1$. Ce paramètre n'est pris en compte que si PRE_COND='LDLT_INC'. Ce chiffre permet à PETSc de prévoir grossièrement la taille nécessaire

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

pour stocker le préconditionneur. Si cette estimation est trop faible, PETSc agrandit les objets à la volée, mais cette opération est plus coûteuse.

```
◊  REAC_PRECOND =  /  reac
                   /  30 [DEFAUT]
```

Ce paramètre ne concerne que les préconditionneurs LDLT_SP et LDLT_DP.

Ces préconditionneurs sont beaucoup plus coûteux que le préconditionneur incomplet mais ils sont plus robustes. Pour les rendre vraiment compétitifs par rapport au solveurs directs classiques (MULT_FRONT ou MUMPS double précision), il faut les conserver pendant plusieurs résolutions successives.
Le paramètre REAC_PRECOND conditionne le nombre de fois où l'on garde le même préconditionneur alors que la matrice du problème a changé. Tant que le solveur itératif (ALGORITHME) appelé par PETSC prend moins de reac itérations pour converger, on conserve le préconditionneur inchangé ; s'il dépasse ce nombre, on réactualise le préconditionneur en refaisant une factorisation simple précision.

```
◊  PCENT_PIVOT   =  /  pcent
                   /  20 [DEFAUT]
◊  GESTION_MEMOIRE =  /  'AUTO',              [DEFAUT]
                   /  'IN_CORE'
◊   LOW_RANK_SEUIL  =  /  1.E-08 ,            [DEFAUT]
                   /   lr_seuil          [R]
◊  RENUM  =        /  'SANS',             [DEFAUT]
                   /  ' PARMETIS '
                   /  ' METIS ',
```

Ces paramètres ne concernent que le s préconditionneur s LDLT_SP et LDLT_DP. Le mot clé ' LOW_RANK_SEUIL ' permet de régler la précision de la compression block-low-rank utilisée par MUMPS. Plus ce paramètre est grand, plus la compression est sévère et moins la factorisation est fidèle.
Il s'agit des mêmes mots-clés que pour le solveur MUMPS, *cf*. §3.5.4.

```
◊   MATR_DISTRIBUEE =  /  'OUI'
                       /  'NON' [DEFAUT]
```

Ce paramètre est pour l'instant limité aux opérateurs MECA_STATIQUE, STAT_NON_LINE et DYNA_NON_LINE et il n'est actif qu'en parallèle distribué (AFFE_MODELE/PARTITION/PARALLELISME!='CENTRALISE').
En activant ce mot-clé, le stockage de la matrice assemblée se fait de manière distribuée sur tous les processeurs (on ne stocke plus de valeurs inutiles appartenant aux autres processeurs). Cela permet d'économiser de la mémoire en parallèle sans surcoût en temps, ni perte de précision (ce mot-clé n'a aucune influence en séquentiel ou en parallèle centralisé). Il est à noter qu'il est recommandé d'utiliser le partitionnement SOUS_DOMAINE dans AFFE_MODELE afin d'éviter de potentiels problèmes de conditionnement liés à la renumérotation de la matrice assemblée.

```
◊   NMAX_ITER   =  /  niter
                   /  0   [DEFAUT]
```

Nombre d'itérations maximum de l'algorithme de résolution itératif. Si $niter \leq 0$, le nombre d'itérations maximal est fixé par défaut à 10000. Cette valeur est réduite à 100 si l'on utilise le préconditionneur LDLT_SP.

```
◊   RESI_RELA   =  /  resi
                   /  10-6    [DEFAUT]
```

Critère de convergence de l'algorithme. C'est un critère relatif sur le résidu préconditionné :

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

$$\frac{\|M^{-1}.\mathbf{r}_m\|}{\|M^{-1}.\mathbf{f}\|} \leq resi$$

M^{-1} est le préconditionneur
$\mathbf{r}_m$ est le résidu à l'itération m
$\mathbf{f}$ est le second membre et la norme $\|\ \|$ est la norme euclidienne usuelle.

Remarques :

1 *Le critère de convergence pour* `PETSC` *est évalué différemment de* `GCPC` *;*
2 *Lorsque le préconditionneur est de mauvaise qualité (par exemple à cause d'un mauvais conditionnement du problème), le critère de convergence utilisé par* `PETSC` *peut donner lieu à des mauvaises solutions ; c'est pourquoi dans les opérateurs de calcul linéaire, une vérification supplémentaire sur le résidu non-préconditionné est effectuée. La tolérance choisie pour ce critère supplémentaire est* $\sqrt{resi}$ *;*
3 *L'algorithme* `'GCR'` *s'appuie sur un pré-conditionnement à droite et vérifie donc le critère de convergence en norme non pré-conditionnée.*
4 *Lorsqu'on utilise le préconditionneur* `LDLT_INC` *, la matrice est systématiquement renumérotée à l'aide de l'algorithme Reverse-Cuthill-Mackee. L'utilisateur ne peut pas modifier ce choix.*
5 Le préconditionneur `LDLT_INC` *est incompatible avec* `MATR_DISTRIBUEE='OUI'.`

3.7.1 Mots-clefs spécifiques au préconditionneur FIELDSPLIT

◊ NOM_CMP

Liste des composantes des champs qui apparaissent dans la modélisation, par exemple NOM_CMP=('DX','DY','DZ','PRES') pour une modélisation mixte en déplacement et pression pour un problème d'élasticité incompressible.

◊ PARTITION_CMP

Manière dont on va grouper les composantes des champs. Dans l'exemple NOM_CMP=('DX','DY','DZ','PRES'), on utilise PARTITION_CMP=(3,1), ce qui signifie que l'on va traiter séparément les trois composantes de déplacement et la composante de pression.

◊ OPTION_PETSC

Chaîne de caractère qui définit le préconditionneur par blocs. Cette chaîne doit être sur une ligne unique (sans retour à la ligne).

3.7.2 Mots-clefs spécifiques au préconditionneur UTILISATEUR

◊ KSP_UTIL

Fonction écrite par l'utilisateur qui décrit l'algorithme de résolution. Elle est écrite en Python et exploite l'interface PETSc pour ce langage, appelée petsc4py. **Il s'agit d'une utilisation avancée de PETSc, à réserver exclusivement au fin connaisseur à fin de recherche.**

3.7.3 Mot-clef OPTION_PETSC

Outre définir précisément le préconditionneur FIELDSPLIT (comme évoqué ci-dessus), le mot-clé OPTION_PETSC permet également de passer à PETSc des options supplémentaires, non-prises en charge par l'interface décrite ici. Cette librairie est en effet conçue pour être hautement configurable à l'exécution ce que permet de réaliser ce mot-clé. **Il s'agit d'une utilisation avancée de PETSc, à réserver exclusivement au fin connaisseur à fin de recherche.**

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

◊ OPTION_PETSC
Chaîne de caractère qui permet de passer un jeu d'options supplémentaires à PETSc

Il est important de noter que les options supplémentaires passées à PETSc sont conservées de son initialisation à sa fermeture, généralement à l'exécution de la commande FIN(). Donc si un utilisateur passe des options supplémentaires dans STAT_NON_LINE, elles seront utilisées pour toutes les résolutions ultérieures même avec d'autres opérateurs de résolution comme DYNA_VIBRA.
Pour contourner cela, il possible de stopper PETSc ce qui a pour effet de réinitialiser toutes les options. Cela se fait à l'aide des commandes :

```
from code_aster import LinearAlgebra      # import du module LinearAlgebra
LinearAlgebra.petscInitialize()           # appel à la méthode qui initialise PETSc. Elle
                                          # peut prendre comme arguments de nouvelles
                                          # options
LinearAlgebra.petscFinalize()             # appel à la méthode qui ferme PETSc et réinitialise
                                          # les options
```


Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)