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, MODE_ITER_SIMULT, ...). 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 3 catégories : les solveurs directs, les solveurs itératifs et les solveurs hybrides.

Concernant les solveurs directs, on dispose de l'algorithme classique de « Gauss » (METHODE='LDLT'), d'une factorisation multifrontale ('MULT_FRONT') et d'une résolution externe ('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').
Lorsque l'on mixe les deux premières approches, on fait de la résolution hybride. C'est le cas du solveur multi-domaines FETI ('FETI').

Seuls MULT_FRONT, MUMPS, PETSC et FETI sont **parallélisés**. Le premier en mémoire partagée (OpenMP), les autres en mémoire distribuée (MPI). Mais tous les solveurs sont compatibles avec un traitement parallèle des calculs élémentaires et des assemblages [U4.41.01].
D'autre part, seuls les trois solveurs directs sont compatibles avec le calcul modal et les études de flambement.

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

Table des Matières

2 Syntaxe

```
◊  SOLVEUR = _F (

# Paramètre commun à tous les solveurs (directs, itératifs et hybride)
   ◊  SYME=                          /  'NON',                  [DEFAUT]
                                     /  'OUI'.
# Paramètres communs aux solveurs directs et hybride ('MULT_FRONT', 'LDLT', 'MUMPS' et 'FETI')
   ◊  NPREC=                         /  8,                      [DEFAUT]
                                     /  nprec.                  [I]
   ◊  STOP_SINGULIER=                /  'OUI',                  [DEFAUT]
                                     /  'NON',

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

#Solveur direct « classique » de type Gauss : cf. §3.4.
   /  METHODE='LDLT',
            ◊  RENUM=                /  'RCMK',                 [DEFAUT]
                                     /  'SANS'.

#Solveur direct de type multifrontal basé sur le produit externe MUMPS : cf §3.5.
   /  METHODE='MUMPS',
            ◊  TYPE_RESOL=           /  'AUTO',                 [DEFAUT]
                                     /  'NONSYM',
                                     /  'SYMGEN',
                                     /  'SYMDEF'.
            ◊  PCENT_PIVOT=          /  20,                     [DEFAUT]
                                     /  pcent.                  [R]
            ◊  ELIM_LAGR2=           /  'OUI',                  [DEFAUT]
                                     /  'NON'.
            ◊  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',
                                     /  'SCOTCH'.
            ◊  POSTTRAITEMENTS=      /  'SANS',
                                     /  'FORCE',
                                     /  'AUTO'.                 [DEFAUT]
            ◊  MATR_DISTRIBUEE=      /  'OUI',       (périmètre limité cf. §3.5)
```

```
                                    /   'NON'.                  [DEFAUT]
          ◊  GESTION_MEMOIRE=       /   'AUTO',                 [DEFAUT]
                                    /   'IN_CORE',
                                    /   'OUT_OF_CORE',
                                    /   'EVAL'.
```

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

```
  /  METHODE='GCPC',
          ◊  /  PRE_COND=           /   'LDLT_INC',           [DEFAUT]
                ◊  NIVE_REMPLISSAGE=      /   0,              [DEFAUT]
                                          /   niv.
                ◊  RENUM=                 /   'SANS',
                                          /   'RCMK'.         [DEFAUT]
             /  PRE_COND=           /   'LDLT_SP',
                ◊  PCENT_PIVOT =          /   20,             [DEFAUT]
                                          /   pcent.
                ◊  REAC_PRECOND=          /   30,             [DEFAUT]
                                          /   reac.
                ◊  RENUM=                 /   'SANS',         [DEFAUT]
          ◊  NMAX_ITER=             /   0,                    [DEFAUT]
                                    /   niter.                [I]
          ◊  RESI_RELA=             /   10-6,                 [DEFAUT]
                                    /   resi.                 [R]
```

#Solveurs itératifs basés sur la bibliothèque externe PETSc : c *f. § 3.7 .*

```
  /  METHODE='PETSC',
          ◊  ALGORITHME=            /   'GMRES',              [DEFAUT]
                                    /   'CG',
                                    /   'CR',
                                    /   'GCR'.
          ◊  /  PRE_COND=           /   'LDLT_SP',            [DEFAUT]
                ◊  PCENT_PIVOT =          /   20,             [DEFAUT]
                                          /   pcent.
                ◊  REAC_PRECOND=          /   30,             [DEFAUT]
                                          /   reac.
                ◊  RENUM=                 /   'SANS',         [DEFAUT]
             /  PRE_COND=           /   'LDLT_INC',
                ◊  NIVE_REMPLISSAGE=      /   0,              [DEFAUT]
                                          /   niv.
                ◊  REMPLISSAGE=           /   1.0,            [DEFAUT]
                                          /   rem.
                ◊  RENUM=                 /   'SANS',
                                          /   'RCMK'.         [DEFAUT]
              /  PRE_COND=          /   'ML',
                                    /   'BOOMER',
                ◊  RENUM=                 /   'SANS',         [DEFAUT]
              /  PRE_COND=          /   'JACOBI',
                                    /   'SOR',
                                    /   'SANS'.
                ◊  RENUM=                 /   'SANS',         [DEFAUT]
                                          /   'RCMK'.
```

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

#Solveur hybride multidomaines de type FETI : *cf. §3.8.*

```
    /  METHODE='FETI',
                ♦  PARTITION=            sdfeti
                ◊  NMAX_ITER=            /  0,                  [DEFAUT]
                                         /  niter.              [I]
                ◊  REAC_RESI=            /  0,                  [DEFAUT]
                                         /  nreac.              [I]
                ◊  RESI_RELA=            /  10-6,               [DEFAUT]
                                         /  resi.               [R]
                ◊  PRE_COND=             /  'SANS',
                                         /  'LUMPE'.            [DEFAUT]
                ◊  SCALING=              /  'SANS',
                                         /  'MULT'.             [DEFAUT]
                ◊  TYPE_REORTHO_DD=      /  'SANS',
                                         /  'GSM',              [DEFAUT]
                                         /  'GS',
                                         /  'IGSM'.
                ◊  NB_REORTHO_DD=        /  0,                  [DEFAUT]
                                         /  nb_reortho.         [I]
                ◊  RENUM=                /  'METIS',            [DEFAUT]
                                         /  'MD',
                                          / 'MDA'.
                ◊  VERIF_SDFETI=         /  'OUI',              [DEFAUT]
                                         /  'NON'.
                ◊  TEST_CONTINU=         /  10-8,               [DEFAUT]
                                         /  test_continu.       [R]
                ◊  STOCKAGE_GI=          /  'CAL',              [DEFAUT]
                                         /  'OUI',
                                         /  'NON'.
                ◊  INFO_FETI=            /  'FFFFFFFFFFFFFFF',  [DEFAUT]
                                         /  info_feti.          [K15]
                ◊  NB_SD_PROC0=          /  0,                  [DEFAUT]
                                         /  nb_sdproc0.         [I]
                ◊  ACCELERATION_SM=      /  'OUI',              [DEFAUT]
                                         /  'NON'.
                ◊  NB_REORTHO_INST=      /  0,                  [DEFAUT]
                                         /  nb_reortho_inst.    [I]

                    ),
```

3 Opérandes

3.1 Opérande METHODE

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

#Solveurs directs

/'MULT_FRONT' [**DEFAUT**] **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' **Solveur direct de type multifrontale** avec pivotage. Ce solveur appelle la bibliothèque MUMPS développée par CERFACS/IRIT/INRIA/CNRS.
Permet de traiter les modèles conduisant à des matrices non définies positives (hors conditions aux limites). Par exemple, les éléments "mixtes" ayant des ddls de type "Lagrange" (éléments incompressibles...).
Cette méthode est parallélisée en mémoire distribuée (MPI) et peut être exécutée sur plusieurs processeurs (*via* l'interface Astk menu Options / Options de lancement / mpi_nbcpu & mpi_nbnoeud).

#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 (*via* l'interface Astk menu Options / Options de lancement / mpi_nbcpu & mpi_nbnoeud).

Attention : les solveurs PETSC et MUMPS étant incompatibles en séquentiel, on privilégie généralement MUMPS. Pour utiliser PETSC, il faut donc souvent lancer une version parallèle de *Code_Aster* (quitte à ne solliciter qu'un seul processeur).

#Solveur hybride

/'FETI' **Solveur hybride par décomposition de domaines de type FETI.** Gradient conjugué préconditionné projeté (GCPPC) pour le problème d'interface et solveur direct multifrontal pour les résolutions locales.
Cette méthode est parallélisée en mémoire distribuée (MPI) et peut être exécutée sur plusieurs processeurs (*via* l'interface Astk menu Options / Options de lancement / mpi_nbcpu & mpi_nbnoeud).

Conseil :

La méthode par défaut reste la multifrontale interne MULT_FRONT. Mais pour pleinement bénéficier des gains en temps et mémoire que procure le **parallélisme** ou pour résoudre un **problème numériquement difficile** (X-FEM, incompressibilité, THM), on préconise l'utilisation de **MUMPS**.
Pour résoudre un problème de **grande taille** (> 10^6 ddls), le solveur PETSC avec ses réglages par défaut peut permettre de passer un calcul malgré des limitations mémoire de la machine de calcul.

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

3.2 Paramètres communs à plusieurs solveurs

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

Ce paramètre est commun à tous les solveurs linéaires (directs, itératifs et hybride). Il n'est appelable que pour les opérateurs non linéaires quasi-statiques.

Si la matrice du système linéaire $\mathbf{K}$ est non-symétrique, le mot-clé SYME='OUI' permet de symétriser cette matrice avant la résolution du système. La matrice est alors remplacée par $\mathbf{K}'=\frac{1}{2}\left(\mathbf{K}+\mathbf{K}^T\right)$.

Attention :
L'intérêt de ce mot clé est de gagner du temps lors de la résolution des systèmes linéaires car les solveurs directs sont généralement plus coûteux en non symétrique.
La symétrisation de la matrice $\mathbf{K}$ conduit cependant à résoudre un système linéaire différent. Ce faisant, elle peut dégrader la convergence des algorithmes non-linéaires.

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

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

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

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 a 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 et FETI :
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 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 bonne raison.*

3.3 METHODE='MULT_FRONT'

Périmètre d'utilisation :
Solveur universel utilisable partout. À déconseiller pour les modélisations nécessitant du pivotage (éléments finis mixtes, incompressible ...).

◊ RENUM =

Cet argument permet de renuméroter 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.

3.4 METHODE='LDLT'

Périmètre d'utilisation :
Solveur universel (mais très lent sur de gros cas). À déconseiller pour les modélisations nécessitant du pivotage (éléments finis mixtes, incompressible...).

◊ RENUM =

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

/'SANS'	On garde l'ordre initial donné dans le fichier de maillage.
/'RCMK' [**DEFAUT**]	'Reverse Cuthill-MacKee', cet algorithme de renumérotation est souvent efficace pour réduire la place nécessaire (en stockage SKYLINE) de la matrice assemblée et pour réduire le temps nécessaire à la factorisation de la matrice.

3.5 METHODE='MUMPS'

Périmètre d'utilisation :
Solveur universel.

Le solveur MUMPS actuellement développé par CERFACS/CNRS/ENSEEIHT-INPT/INRIA (Copyright au §3 de [U2.08.03]) est un solveur direct de type multifrontal, parallélisé (en MPI) et robuste, car il permet de pivoter les lignes et colonnes de la matrice lors de la factorisation numérique.
À partir de la MUMPS v4.10.0, le périmètre du produit dans *Code_Aster* est complet. Pour les versions antérieures (en fait seule la 4.9.2 est temporairement acceptée), les calculs de déterminant nécessaires aux opérateurs MODE_ITER_INV (option 'SEPARE'/'AJUSTE') et INFO_MODE (méthode APM) ne sont pas possibles. Le calcul s'arrête alors en ERREUR_F en conseillant la montée de version.

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. paramètre SYME). C'est le même type d'astuces que pour les paramètres de relaxation FILTRAGE_MATRICE et MIXER_PRECISION.

◊ ELIM_LAGR2 =

Ce mot-clé permet d'éliminer les doubles Lagrange de la prise en compte des conditions aux limites :

/'OUI' [**DEFAUT**]	On ne tient plus compte que d'un Lagrange, l'autre étant spectateur
/'NON'	On conserve les matrices dualisées usuelles.

Historiquement, les solveurs linéaires directs de *Code_Aster* ('MULT_FRONT' et 'LDLT') ne disposaient pas d'algorithme de pivotage. Pour contourner ce problème, la prise en compte des conditions limites par des Lagranges a été modifiée en introduisant des doubles Lagranges au prix d'un surcoût mémoire et calcul. Comme MUMPS dispose de facultés de pivotage, ce choix de dualisation des conditions limites peut être remis en cause.
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 MODE_ITER_INV et INFO_MODE. L'utilisateur est alors averti de ce changement automatique de paramétrage *via* un message dédié (uniquement en INFO=2).

◊ 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).

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).
Si l'erreur relative 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 dont l'objectif est d'améliorer la solution obtenue. Ce post-traitement 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.

3.5.2 Paramètres numériques

```
◊   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 :

• 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.

• Mais pour gagner de la place mémoire sans risquer de perdre en précision de calcul, on peut aussi s'intéresser aux éléments suivant: calcul parallèle[U2.08.03], paramètres GESTION_MEMOIRE , MATR_DISTRIBUEE voire *RENUM.*

• *Cette idée a été reprise avec le préconditionneur* `LDLT_SP` *de* `GCPC/PETSC`*.*

◊ 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 pré-traitement.
/'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 « frustres » 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'`, `'SCOTCH'`).

/'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.
/'SCOTCH'	Outil externe de renumérotation qui tend à supplanter l'outil de référence dans le domaine (METIS).
/'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.

Remarques :

• Les outils externes de renumérotation ont en fait des fonctionnalités plus larges: partitionnement de graphes et de maillages, organisation de déroulement de tâches... Par exemple, dans Code_Aster, METIS et SCOTCH sont aussi utilisés dans l'opérateur de partitionnement de maillage `DEFI_PART_FETI`*.*

• 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.

• En particulier, lors de calculs modaux parallèles, on a parfois observé des speed-ups décevants du fait d'un choix inapproprié de renuméroteur. Dans ce cas là on a constaté que le choix d'un renuméroteur « sophistiqué » était contre-performant. Il vaut mieux imposer à MUMPS un simple `'AMF'` *ou* `'QAMD'`*, plutôt que* `'METIS'` *(souvent pris automatiquement en mode* `'AUTO'`*).*

◊ POSTTRAITEMENTS =

Ce paramètre n'est utilisé que si `RESI_RELA` est activé. 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 ex. `FACTORISER` et `INFO_MODE`).

Ce mot clé permet de contrôler la procédure de raffinement itératif dont l'objectif est d'améliorer la qualité de la solution (cf. mot-clé `RESI_RELA`).

/'SANS'	Désactivation.
/'FORCE'	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.
/'AUTO' [**DEFAUT**]	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.

Remarques :

•Ce processus consomme principalement des descentes-remontées dont le coût en terme de performances est faible en In-Core. Par contre, elles peuvent être coûteuses en Out-Of-Core voire peu utiles en non linéaire (l'algorithme de Newton corrige).

•Pour limiter toute dérive contre-productive, ce processus est bridé 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.

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

3.5.3 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) :

•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).

•À précision numérique constante mais avec potentiellement des pertes en temps de calcul : l'activation explicite des facultés Out-Of-Core de MUMPS (cf. mot-clé GESTION_MEMOIRE ci dessous) ou 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 à votre cas de figure.

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

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
                   /  2 0%      [ 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.

•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

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

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.

```
*******************************************************************************
 - 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 à 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

aussi les objets préalables de stockage des données (matrice, RHS) et un 30 Mo forfaitaire pour l'exécutable MUMPS.

4 Cela peut arriver sur certaines plate-formes (par ex. clpaster-Rocks).

5 Matrice (MATR_ASSE), description des inconnues (NUME_DDL)...

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'.
• 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 MODE_ITER_SIMULT . 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, on provoque le découpage au plus juste de l'objet matrice dans *Aster* (on ne stocke plus de valeurs inutiles appartenant aux autres processeurs). Cela permet d'économiser de la mémoire en parallèle distribué (ce mot-clé n'a aucune influence en séquentiel ou en parallèle centralisé) sans surcoût en temps, ni perte de précision.

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 qu'on active, au préalable, le mot-clé SYME .

◊ 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 *via* l'outil externe MUMPS

Cette deuxième approche est plus coûteuse en CPU/RAM mais plus robuste. 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.

◊ RENUM =

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

/'SANS'	On garde l'ordre initial donné dans le fichier de maillage.
/'RCMK'	'Reverse Cuthill-MacKee', cet algorithme de renumérotation est souvent efficace pour réduire la place nécessaire (en stockage SKYLINE) de la matrice assemblée et pour réduire le temps nécessaire à la factorisation de la matrice.

La valeur par défaut de ce mot-clé dépend du préconditionneur. Seul LDLT_INC utilise une rénumérotation RCMK par défaut.

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

Ce paramètre ne concerne que le préconditionneur LDLT_SP .
Ce préconditionneur est beaucoup plus coûteux que le préconditionneur incomplet mais il est plus robuste car plus proche de la solution exacte. 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 refaisant une factorisation simple précision.

◊ PCENT_PIVOT = / pcent
/ 20 [**DEFAUT**]

Ce paramètre ne concerne que le préconditionneur LDLT_SP .
Il s'agit du même mot-clé que pour le solveur MUMPS, *cf*. §3.5.3.

◊ 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.

3.7 METHODE='PETSC'

Périmètre d'utilisation :
Tous types de problèmes réels sauf ceux requérant obligatoirement une détection de singularité (calcul modal).

Attention : les solveurs PETSC et MUMPS étant incompatibles en séquentiel, on privilégie généralement MUMPS. Pour utiliser PETSC, il faut donc souvent 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* :

/'GMRES' [**DEFAUT**]	'Generalised Minimal RESidual'
/'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.

◊ PRE_COND =

Nom des préconditionneurs de PETSc accessibles depuis *Code_Aster :*

/'LDLT_INC'	Factorisation incomplète par niveau
/'LDLT_SP' [**DEFAUT**]	Factorisation simple précision *via* l'outil externe MUMPS
/'ML'	Multigrille algébrique « *multilevel* » (bibliothèque ML)
/'BOOMER'	Multigrille algébrique « BoomerAMG » (bibliothèque HYPRE)
/'JACOBI'	Pré-conditionneur diagonal standard
/'SOR'	'Successive Over Relaxation'
/'SANS'	Pas de préconditionneur

En mode parallèle, seuls LDLT_SP, ML, BOOMER et JACOBI offrent un préconditionnement équivalent au mode séquentiel. 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 et BOOMER ont un périmètre d'application très restreint (calcul sans multiplicateurs de Lagrange et avec un nombre de degrés de liberté constant par nœud). Ils sont néanmoins très efficaces en parallèle.
Le préconditionneur 'LDLT_SP' est *a priori* le plus robuste mais c'est aussi le plus coûteux. Cependant, et à l'inverse des autres préconditionneurs, il n'est pas reconstruit à chaque résolution linéaire, ce qui le rend finalement compétitif (*cf.* mot-clé REAC_PRECOND).

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

◊ RENUM = / 'SANS'
/ 'RCMK'

Renuméroteur de type 'Reverse Cuthill-Mackee' (comme 'LDLT') pour limiter le remplissage de la factorisée incomplète.
La valeur par défaut de ce mot-clé dépend du type de préconditionneur utilisé. Seul le préconditionneur LDLT_INC réalise une renumérotation par défaut (RCMK).

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

Ce paramètre ne concerne que le préconditionneur LDLT_SP.
Ce préconditionneur est beaucoup plus coûteux que le préconditionneur incomplet mais il est plus robuste car plus proche de la solution exacte. 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.
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**]

Ce paramètre ne concerne que le préconditionneur LDLT_SP .
Il s'agit du même mot-clé que pour le solveur MUMPS, *cf*. §3.5.3.

◊ 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, on provoque le découpage au plus juste de l'objet matrice dans *Aster* (on ne stocke plus de valeurs inutiles appartenant aux autres processeurs). Cela permet d'économiser de la mémoire en parallèle distribué (ce mot-clé n'a aucune influence en séquentiel ou en parallèle centralisé) sans surcoût en temps, ni perte de précision.

◊ NMAX_ITER = / niter
/ 1 [**DEFAUT**]

Nombre d'itérations maximum de l'algorithme de résolution itératif. Si $niter \leq 0$ alors il est fixé automatiquement par PETSc (10^5).

◊ RESI_RELA = / resi
/ 10^{-6} [**DEFAUT**]

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

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

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 :

- *Le critère de convergence pour PETSC est évalué différemment de GCPC ;*
- *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 $\sqrt{resi}$.*

3.8 METHODE='FETI'

Périmètre d'utilisation :
Cette méthode est limitée aux opérateurs MECA_STATIQUE et STAT_NON_LINE. Les limitations sont :
- Pas d' AFFE_CHAR_CINE,
- Pas de macro-élément,
- Pas de Dirichlet généralisé entre sous-domaines (AFFE_CHAR_MECA/LIAISON_***),
- Pas de force fluide,
- Pas de contact-frottement,
- Des réserves sur les modélisations et les algorithmes susceptibles d'impacter les interfaces (sauf les chargements d'AFFE_CHAR_MECA pour lesquels tous les cas de figures sont prévus) : éléments joints, fissure... Il vaut mieux essayer de contingenter ces zones compliquées à l'intérieur des sous-domaines.
- Solveurs locaux à chaque sous-domaine tous homogènes, basés sur MULT_FRONT.

♦ PARTITION = sdfeti

Nom utilisateur de l'objet SD_FETI décrivant le partitionnement en sous-domaines. Il est généré par un appel préalable aux opérateurs DEFI_PART_FETI/OPS [U4.23.05].

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

Nombre d'itérations maximum du GCPPC résolvant le problème d'interface. Si niter = 0 alors le nombre maximum d'itérations est calculé comme suit :

niter=max(nbi/100,10) où nbi le nombre d'inconnues du problème d'interface.

◊ REAC_RESI = / nreac
/ 0 [**DEFAUT**]

Fréquence de réactualisation du calcul du résidu. Valeurs conseillées : 10 ou 20.

◊ RESI_RELA = / resi
/ 10^{-6} [**DEFAUT**]

Critère de convergence de l'algorithme : c'est un critère relatif sur le résidu projeté du problème d'interface :

$$\frac{\|\mathbf{P}\,\mathbf{r}_m\|}{\|\mathbf{b}\|} \leq resi$$

$\mathbf{r}_m$ est le résidu à l'itération m
$\mathbf{P}$ l'opérateur de projection
$\mathbf{b}$ est le second membre et $\|\ \|$ est la norme euclidienne usuelle.

◊ PRE_COND =

Cet argument permet de choisir le type de préconditionneur pour le GCPPC :

/'SANS'	Pas de préconditionnement.
/'LUMPE' [**DEFAUT**]	Préconditionnement lumpé.

Normalement le préconditionneur lumpé conduit à un gain en itérations et en CPU, sans surcoût mémoire.

◊ SCALING =
Cet argument permet de choisir le type de scaling (mise à l'échelle) adopté pour le préconditionneur. Il n'est donc pris en compte que si PRE_COND est différent de 'SANS'.

/'SANS'	Pas de phase de scaling.
/'MULT' [**DEFAUT**]	Mise à l'échelle par la multiplicité des nœuds d'interface.

Normalement la phase de scaling conduit à un gain en itérations et en CPU, sans surcoût mémoire. Surtout lorsque le partitionnement produit beaucoup de points de jonction (points appartenant à plus de deux sous-domaines).

◊ TYPE_REORTHO_DD =

Cet argument permet de choisir le type de réorthogonalisation des directions de descente (au sein d'une résolution de système linéaire ou entre différentes résolutions cf. ACCELERATION_SM). Il est lié au paramètre NB_REORTHO_DD.

/'SANS'	Pas de réorthogonalisation des méthodes de descente.
/'GS'	Réorthogonalisation de Gram-Schmidt.
/'GSM' [**DEFAUT**]	Réorthogonalisation de Gram-Schmidt Modifiée.
/'IGSM'	Réorthogonalisation de Gram-Schmidt Modifiée Itérative.

Cette phase permet de lutter contre la propension des directions de descente du GCPPC à perdre leur orthogonalité. En théorie, 'IGSM' est meilleure que 'GSM' qui est lui même supérieur à 'GS'. En pratique, le meilleur compromis «surcoût calcul/qualité d'orthogonalité» est souvent réalisé par 'GSM'.

◊ NB_REORTHO_DD = / nb_reortho
/ 0 [**DEFAUT**]
Nombre de directions de descente initiales utilisées dans la phase de réorthogonalisation. En principe, plus il est grand, meilleure est la convergence, mais plus grand est aussi le surcoût calcul et mémoire. Il faut donc trouver un compromis entre ces éléments.
Si nb_reortho = 0 alors ce nombre est calculé comme suit : nb_reortho=max(niter/10,5) où niter est le nombre maximal d'itérations définies ci-dessus.

◊ RENUM Voir [§3.3].

◊ STOP_SINGULIER Voir [§3.2].

◊ NPREC Voir [§3.2].

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

On comptabilise les incohérences en terme de nom de modèle et de noms de chargement, entre le paramétrage de l'opérateur appelant le mot-clé SOLVEUR et celui fournit à l'opérateur de partitionnement qui reste stocké dans la SD_FETI. Il faut que les noms de modèles soient identiques et que la liste des chargements de l'opérateur appelant soit égale à celle de DEFI_PART_OPS. Si ce n'est pas le cas et si VERIF_SDFETI='OUI', on s'arrête en ERREUR_FATALE, sinon on émet une ALARME.

◊ TEST_CONTINU= / test_continu
/ 10^{-8} [**DEFAUT**]

Critère du test de continuité à l'interface : c'est un critère relatif sur les valeurs (non nulles) des inconnues aux interfaces. Si on est en dessus du critère, il y a émission d'une ALARME . Ce critère est pour l'instant désactivé.

◊ STOCKAGE_GI =

Lorsque le nombre de sous-domaines augmente, un objet devient proéminent, c'est $\mathbf{G}_I$ la matrice des traces des modes de corps rigides sur l'interface. Elle est utilisée dans la phase de projection, c'est-à-dire $10+\text{nombre\_itérations\_FETI}\times 4$. Pour permettre à l'utilisateur d'adapter le compromis «taille mémoire/temps CPU», son stockage est paramétrable :

/'OUI' [**DEFAUT**]	Elle est calculée et stockée une fois pour toute. Cela nécessite plus de mémoire mais moins de temps calcul lorsqu'on s'en sert.
/'NON'	C'est l'inverse, elle est recalculée à chaque fois que cela est nécessaire.
/'CAL'	Le choix 'OUI' ou 'NON' va être calculé automatiquement. Si la taille de la matrice est inférieure à la taille moyenne des matrices de rigidité locales, on stocke ('OUI'), sinon, on recalcule ('NON').

◊ INFO_FETI = / info_feti
/ 'FFFFFFFFFFFFFFF' [**DEFAUT**]

Mot-clé de *monitoring* et de *debugging*. Son paramétrage est détaillé dans la documentation [U2.08.03].

◊ NB_SD_PROC0 = / nb_sdproc0
/ 0 [**DEFAUT**]

Paramètre utilisé en mode parallèle MPI, permettant d'attribuer un nombre de sous-domaines arbitraire au processeur 0 (le «maître»). Ce nombre peut ainsi être inférieur à celui qui lui serait attribué par la procédure de répartition automatique «sous-domaines/processeurs». Cela permet de le soulager, en CPU et en espace mémoire, par rapport aux autres processeurs car il doit gérer des étapes supplémentaires et des objets JEVEUX potentiellement volumineux (phase de réorthogonalisation, projections du problème grossier…).
Il n'est actif que si il est licite : $\text{nb\_sdproc0}>0$ et $\text{nb\_sdproc0}<\text{nbsd}-\text{nbproc}+1$ (*nbproc* , nombre de processeurs et *nbsd* , nombre de sous-domaines).

◊ ACCELERATION_SM =

Cet argument permet d'activer la phase d'accélération d'un problème avec multiples seconds membres (par exemple, un calcul d'élasticité avec des chargements thermiques dépendant du temps).

/'OUI' [**DEFAUT**]	Accélération activée si les conditions sont réunies (voir plus bas).
/'NON'	Accélération désactivée.

Lorsqu'il est activé, le GCPPC du solveur FETI ne va pas démarrer son processus en partant de zéro, mais va au contraire s'appuyer sur une information *a priori*, celle des directions de descente stockées au nb_reortho_inst pas de temps précédents. On va donc gagner beaucoup en nombre d'itérations et donc en CPU, en concédant assez peu en mémoire (si l'interface est faible devant la taille du problème).

Cet argument est lié au paramètre NB_REORTHO_INST et n'est activé que si le problème est une succession de systèmes linéaires à seconds membres différents et si la réorthogonalisation des directions de descente, au sein de chaque pas de temps, est activée (TYPE_REORTHO_DD différent de 'SANS').

◊ NB_REORTHO_INST = / nb_reortho_inst
/ 0 [**DEFAUT**]

À un pas de temps donné, c'est le nombre de pas de temps précédents dont on va utiliser les directions de descente pour la procédure d'accélération. En principe, plus il est grand, meilleure est la convergence, mais plus grand est aussi le surcoût calcul et mémoire. Il faut donc trouver un compromis entre ces éléments.

Si $\text{nb\_reortho\_inst} = 0$ alors ce nombre est calculé comme suit :

$$\text{nb\_reortho\_inst} = max(\text{nb\_pas\_temps}/5,5)$$

où nb_pas_temps est le nombre de pas de temps du problème.

Et si, au pas de temps num_pas_temps, nb_reortho_inst est supérieur au nombre de pas de temps précédents disponibles (par ex. au 5ième pas de temps on ne peut utiliser que les 4 pas de temps antérieurs) on le fixe dynamiquement à cette valeur :

$$\text{nb\_reortho\_inst} = \text{num\_pas\_temps} - 1$$