Mot-clé SOLVEUR

1 But

Le mot clé facteur SOLVEUR se retrouve dans les nombreuses commandes (STAT_NON_LINE, MODE_ITER_SIMULT..) qui requièrent la construction et la **résolution de systèmes linéaires**. Pour résoudre ces systèmes d'équations on utilise des algorithmes particuliers appelés «solveurs linéaires». Ces solveurs linéaires sont en fait omniprésents dans le déroulement des opérateurs de *Code_Aster* car ils sont souvent enfouis au plus profond d'autres algorithmes numériques : schéma non linéaire, intégration en temps, analyse modale etc. Ils en consomment souvent la majeure partie du temps CPU et de la mémoire.
Ce mot-clé permet de choisir entre les trois classes de solveurs : 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 outils de la librairie publique 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 OpenMP, les autres en MPI. Mais tous les solveurs sont 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 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] et les documentations de référence associées [R6...]. Les problématiques connexes d'amélioration des performances (RAM/CPU) d'un calcul et, de l'utilisation du parallélisme, font aussi l'objet de notices détaillées: [U1.03.03] et [U2.08.06].

Table des Matières

2 Syntaxe

On liste ici de manière exhaustive, l'ensemble de tous les paramètres du mot-clé SOLVEUR. Suivant les opérateurs, ils ne sont pas tous licites, activables ou nécessaires. Parfois, suivant les opérateurs, leur valeur par défaut (par ex. RESI_RELA), la liste de leurs valeurs possibles (par ex. STOP_SINGULIER) ou leur périmètre d'utilisation peut varier (par ex. NPREC). Toutes ces cas de figure sont décrits dans la suite de ce document.

```
◊  SOLVEUR = _F (

# Paramètres communs aux solveurs directs et hybride ('MULT_FRONT', 'LDLT', 'MUMPS' et
'FETI')
   ◊  STOP_SINGULIER=              /  'OUI',                  [DEFAUT]
                                   /  'NON',
                                   /  'DECOUPE' (périmètre limité cf. §3.2).
   ◊  NPREC=                       /  8,                      [DEFAUT]
                                   /  nprec.                  [I]
# Paramètre commun à tous les solveurs (directs, itératifs et hybride)
   ◊  SYME=                        /  'NON',                  [DEFAUT]
                                   /  'OUI'.

# Paramètres différenciés suivant les solveurs

#Solveur direct «maison» de type multifrontal:
#Périmètre d'utilisation: solveur universel sauf 1 exception, cf. §3.3.
   /  METHODE='MULT_FRONT',                                   [DEFAUT]
          ◊  RENUM=                /  'METIS',                [DEFAUT]
                                   /  'MD',
                                   /  'MDA'.

#Solveur direct "classique" de type Gauss:
#Périmètre d'utilisation: solveur universel mais très lent sur de gros cas, cf. § 3.4.
   /  METHODE='LDLT',
          ◊  RENUM=                /  'RCMK',                 [DEFAUT]
                                   /  'SANS'.

#Solveur direct de type multifrontal basé sur le produit externe MUMPS:
#Périmètre d'utilisation: solveur universel sauf 2 exceptions, cf. § 3.5.
   /  METHODE='MUMPS',
          ◊  TYPE_RESOL=           /  'AUTO',                 [DEFAUT]
                                   /  'NONSYM',
                                   /  'SYMGEN',
                                   /  'SYMDEF'.
          ◊  PCENT_PIVOT=          /  10,                     [DEFAUT]
                                   /  pcent.                  [R]
          ◊  ELIM_LAGR2=           /  'OUI',                  [DEFAUT]
                                   /  'NON'.
          ◊  RESI_RELA=            /  -1.0,(non linéaire/modal)[DEFAUT]
                                   /  +1.e-6,   (en 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]
        ◊  OUT_OF_CORE=             /   'OUI',
                                    /   'NON'.                  [DEFAUT]
        ◊  LIBERE_MEMOIRE=          /   'OUI',
                                    /   'NON'.                  [DEFAUT]
```

Solveur itératif «maison» de type GCPC préconditionné par un Cholesky Incomplet ILU(*k*) ou par une factorisée simple précision (*via* MUMPS).

#Périmètre d'utilisation: problèmes symétriques réels sauf ceux requérant obligatoirement une détection de singularité (calcul modal, critère `CRIT_FLAMB` *).* C*f. §3.6*

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

#Solveurs itératifs basés sur la librairie externe PETSc:

#Périmètre d'utilisation: tous types de problèmes sauf ceux requérant obligatoirement une détection de singularité (calcul modal, critère `CRIT_FLAMB` *).* C*f. §3.7.*

```
/   METHODE='PETSC',
        ◊  ALGORITHME=              /   'CG',                   [DEFAUT]
                                    /   'BCGS',
                                    /   'BICG',
                                    /   'CR',
                                    /   'GMRES',
                                    /   'TFQMR'.
        ◊  /  PRE_COND=             /   'LDLT_INC',             [DEFAUT]
              ◊  NIVE_REMPLISSAGE=      /   0,                  [DEFAUT]
                                        /   niv.
              ◊  REMPLISSAGE=           /   1.0,                [DEFAUT]
                                        /   rem.
           /  PRE_COND=             /   'LDLT_SP',
              ◊  REAC_PRECOND=          /   30,                 [DEFAUT]
                                        /   reac.
```

```
                    /   PRE_COND=          /   'JACOBI',
                    /   PRE_COND=          /   'SOR',
                ◊   RENUM=                 /   'SANS',
                                           /   'RCMK'.              [DEFAUT]
                ◊   NMAX_ITER=             /   0,                   [DEFAUT]
                                           /   niter.               [I]
                ◊   RESI_RELA=             /   10-6,                [DEFAUT]
                                           /   resi.                [R]
```

#Solveur hybride multidomaines de type FETI:

#Périmètre d'utilisation: problèmes symétriques réels avec des restrictions . Les plus importantes concernent les éléments finis mixtes et les blocages via `AFFE_CHAR_CINE` *. 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]
```

#Fin du catalogue de commande

```
                    ),
```

3 Opérandes

3.1 Opérande METHODE

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

#Solveurs directs

/'MULT_FRONT' [**DEFAUT**] **Solveur direct de type multifrontale**. Le stockage matriciel est MORSE (ou 'CSC' pour 'Compressed Sparse Column') et proscrit tout pivotage. 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). Le stockage matriciel hors solveur est MORSE. En entrée du solveur, on fait la conversion au format interne de LDLT : 'ligne de ciel' ('SKYLINE').

/'MUMPS' **Solveur direct de type multifrontale** avec pivotage. Ce solveur est obtenu en appelant le **produit externe MUMPS** développé par CERFACS/IRIT/INRIA/CNRS. Le stockage matriciel hors solveur est MORSE. En entrée du solveur, on fait la conversion au format interne de MUMPS: i, j, K_{ij}, centralisée ou distribuée.
Pour *Code_Aster*, **son intérêt principal réside dans sa capacité à pivoter** lignes et/ou colonnes de la matrice lors de la factorisation en cas de pivot petit. Cette possibilité est utile (voire indispensable) pour 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** $ILU(k)$ **ou basé sur une factorisée simple précision (*via* MUMPS).** Le stockage de la matrice est alors MORSE.

/'PETSC' **Solveurs itératifs issus de la librairie externe PETSc** (Laboratoire Argonne). Le stockage matriciel hors solveur est MORSE. 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).

#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 inversions des matrices de rigidité 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).

Les valeurs par défaut des autres mot-clés sont alors prises automatiquement en fonction de la méthode choisie.

La méthode par défaut reste la multifrontale interne MULT_FRONT. Mais pour pleinement bénéficier des gains CPU et RAM que procure le **parallélisme**, ou pour résoudre un **problème difficile** nécessitant des fonctionnalités avancées, **on préconise plutôt l'utilisation de MUMPS**.

3.2 Paramètres communs à plusieurs solveurs

◊ SYME = / 'OUI'
/ 'NON' [**DEFAUT**]

Ce paramètre est mutualisé entre 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:

• La symétrisation de la matrice $\mathbf{K}$ conduit donc à résoudre un autre système linéaire que celui que l'on cherche à résoudre ! En réalité, cette possibilité (SYME='OUI') n'est utile que dans les commandes non-linéaires (comme STAT_NON_LINE par exemple), pour lesquelles la convergence vers la solution est obtenue par itérations successives. Chaque itéré est obtenu par "estimation" et l'on vérifie ensuite qu'il est "solution". Dans ce cas, une erreur légère sur les itérés n'empêche pas forcément de converger vers la bonne solution. L'intérêt de ce mot clé est de gagner du temps lors de la résolution des systèmes linéaires car les solveurs directs sont généralement beaucoup plus coûteux en non symétrique qu'en symétrique.
• De manière générale, l'opérateur détecte le caractère symétrique, ou non, de la matrice de travail et oriente automatiquement vers la bonne variante du solveur.

Les deux paramètres suivants sont mutualisés entre tous les solveurs linéaires directs et hybride (LDLT, MULT_FRONT, MUMPS et FETI). Ils servent à contrôler le déroulement de la factorisation numérique (l'ingrédient majeur de ces solveurs) et la qualité de la solution du système linéaire. En premier lieu, il s'agit de prévenir l'apparition de systèmes matriciels singuliers du fait d'une mauvaise mises en données du problème (conditions limites absentes ou redondantes). Leurs résolutions pouvant conduire à des ERREUR_FATALE, ou pire, à des résultats très imprécis voire faux.
En fait, la factorisation numérique d'une matrice peut achopper dans deux cas de figures : problème de construction de la factorisée (matrice structurellement ou numériquement singulière) et détection numérique d'une singularité (processus approximé plus sensible). Le comportement du code va dépendre du cas de figure, du paramétrage de NPREC/STOP_SINGULIER/RESI_RELA et du solveur utilisé. La combinatoire des cas de figures est décrite dans le tableau A1-1 de l'Annexe 1.

◊ STOP_SINGULIER = / 'OUI' [**DEFAUT**]
/ 'NON'
/ 'DECOUPE' (uniquement avec DYNA/STAT_NON_LINE, CALC_PRECONT et MACR_ASCOUF/ASPIC_CALC)

Ce paramètre est mutualisé entre tous les solveurs linéaires directs et hybride. Il est appelable dans tous les opérateurs ayant besoin de détecter l'éventuelle singularité des matrices à factoriser pour prévenir l'utilisateur d'une mauvaise mise en donnée. Il n'est donc pas disponible dans les opérateurs modaux (la singularité est gérée par la calcul modal proprement dit).

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 $n > nprec$ (mot clé NPREC ci-dessous), on considère que la matrice est singulière. Si l'utilisateur a indiqué :

STOP_SINGULIER='OUI'	Le code s'arrête alors en ERREUR FATALE.
STOP_SINGULIER='NON'	L'exécution se poursuit avec 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/testés» par le processus de Newton englobant.
STOP_SINGULIER='DECOUPE'	(opérateur 'STAT_NON_LINE' uniquement), le processus de découpe automatique du pas de temps est déclenché.

Pour MUMPS :
En toute rigueur, le critère de détection de singularité n'est pas mis en œuvre de la même manière suivant les solveurs directs. Pour MUMPS, il a été adapté à l'API de ce produit externe. Ce dernier traque ainsi les pivots dont la norme infinie de la ligne (ou de la colonne) est inférieure au seuil

$$10^{-nprec}$$

On a ainsi accès au numéro de ligne incriminée mais pas à une éventuelle perte de décimale lors du calcul des termes de la factorisée. 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 ne coûte rien et, ces solveurs ne contrôlant pas la qualité de leur solution, on a besoin de ce «garde-fou». En effet, hormis un éventuel processus englobant tel que Newton en non linéaire, on n'a pas d'autre moyen pour prévenir des résultats trop imprécis en cas de matrice quasi-singulière.*
  Concernant MUMPS, même en linéaire, on a le critère supplémentaire qui calibre la qualité de la solution (RESI_RELA) pour éviter toute dérive. On a donc laissé la liberté de désactiver ce critère (en posant $nprec<0$).
- *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. La souplesse de l'outil permet de les débrancher, mais ce n'est pas conseillé sans bonne raison.*

◊ NPREC = / nprec
/ 8 [**DEFAUT**]

Ce paramètre est mutualisé entre tous les solveurs linéaires directs et hybride. Il est désactivé par une valeur négative. Cette désactivation n'est pas possible avec les opérateurs modaux car ils ont absolument besoin de détecter les éventuelles singularités des matrices traitées.

C'est le nombre qui sert à calibrer le processus de détection de singularité (cf. mot clé STOP_SINGULIER ci-dessus). 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 8 généralement utilisée.
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).

3.3 METHODE='MULT_FRONT'

Périmètre d'utilisation:
Solveur universel utilisable partout. A déconseiller pour les modélisations nécessitant du pivotage (EF mixtes de X-FEM, 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 :
Ce solveur n'est pas utilisable dans un calcul modal avec des matrices issues d'un NUME_DDL_GENE car les stratégies de renumérotation utilisées dans MULT_FRONT s'appuient sur la notion de noeuds d'un maillage alors que les inconnues d'un NUME_DDL_GENE ne proviennent pas forcément d'un calcul avec maillage (par ex. d'une donnée expérimentale).

3.4 METHODE='LDLT'

Périmètre d'utilisation:
Solveur universel (mais très lent sur de gros cas). A déconseiller pour les modélisations nécessitant du pivotage (EF mixtes de X-FEM, 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. Sauf les options de calcul 'SEPARE' et 'AJUSTE' de MODE_ITER_INV.

Le solveur MUMPS actuellement développé par CERFACS/CNRS/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.

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.

◊ PCENT_PIVOT = / pcent
/ 10% [**DEFAUT**]
Ce mot-clé permet de choisir un pourcentage de mémoire que MUMPS réservera en début de calcul pour ses pivotages. La valeur par défaut est de 10% qui correspond à un nombre de pivotages raisonnable. Si par exemple MUMPS estime à 100 la place nécessaire à une factorisation sans pivotage, il allouera *in fine* 110. Par la suite, si l'espace mémoire requis par les pivotages s'avère plus important, la place mémoire allouée sera insuffisante et le code s'arrêtera en ERREUR_FATALE en demandant d'augmenter ce critère. Une valeur dépassant les 50% doit rester exceptionnelle.

◊ 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.

◊ 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 ex. FACTORISER et IMPR_STURM).

En précisant une valeur strictement positive à ce mot-clé (par ex. 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 résultant 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

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

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 ($\eta\left(\mathbf{K}\right)<10^{+6}$). 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 pourcents 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 paramètres du paragraphe suivant (OUT_OF_CORE , MATR_DISTRIBUEE et LIBERE_MEMOIRE).

◊ 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*.

/ ' 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.

◊ 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 IMPR_STURM).

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 IC. Par contre, elles peuvent être coûteuses en OOC 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 (MUMPS et JEVEUX)

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):

•A 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.
•A précision numérique constante mais avec potentiellement des pertes en temps de calcul: l'activation des facultés OOC de MUMPS (cf. mot-clé OUT_OF_CORE ci dessous) et l'activation de l'écriture disque de tous les objets JEVEUX libérables (cf. mot-clé LIBERE_MEMOIRE ci dessous).
•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).

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

Pour activer ou désactiver les facultés OOC de MUMPS qui va alors décharger entièrement sur disque les blocs de factorisée gérés par chaque processeur. L'activation de l'OOC contribue à réduire la mémoire RAM requise par processeur, mais cela peut parfois 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 cette étape algorithmique, on passe autant de temps à manipuler les données qu'à aller les chercher.
On ne perd pas en précision de calcul en activant cette fonctionnalité.

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

Ce paramètre est pour l'instant limité aux opérateurs MECA_STATIQUE et STAT_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.

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

Ce paramètre permet de gérer les déchargements sur disque d'objets JEVEUX qui sont opérés avant chaque appel au solveur MUMPS. Ces déchargements ont pour objectif de libérer de la mémoire RAM pour le produit externe.

/'OUI' On décharge sur disque tous les objets JEVEUX libérables, c'est-à-dire non utilisés en lecture/écriture.

/'NON' [**DEFAUT**] On sélectionne les entités JEVEUX déchargées. On ne décharge que les plus gros objets liés à la matrice (MATR_ASSE) et à la description des inconnues (NUM_DDL).

En activant ce mot-clé, on ne perd pas en précision numérique, mais on peut perdre beaucoup de temps dans les déchargements/rechargements d'objets JEVEUX. Cela peut arriver, par exemple, en mode parallèle, lorsque les processus sont distribués sur des coeurs contigües en machine (engorgement des accès coeurs/RAM) ou, même en mode séquentiel, lorsqu'on effectue de nombreuses constructions et inversions de systèmes linéaires (coût des rechargements des objets liés aux calculs élémentaires, cf. cas-test ssnl133a).

3.6 METHODE='GCPC'

Périmètre d'utilisation:
Problèmes symétriques réels sauf ceux requérant obligatoirement une détection de singularité (calcul modal, CRIT_FLAMB). 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 $\mathbf{LDL}^T$ 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 =
Ce paramètre ne concerne que le préconditionneur LDLT_INC . Cet argument permet de renuméroter les nœuds du modèle pour diminuer la taille de la factorisée incomplète (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.

◊ 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.

◊ 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:

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

$\mathbf{r}_m$ est le résidu 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:
Tout types de p roblème réel sauf ceux requérant obligatoirement une détection de singularité (calcul modal, CRIT_FLAMB).

◊ ALGORITHME =
Nom des solveurs itératifs (de type Krylov) de PETSc accessibles depuis *Code_Aster* :

/'BCGS'	Gradient bi-conjugué stabilisé
/'BICG'	Gradient bi-conjugué
/'CG' [**DEFAUT**]	Gradient conjugué standard
/'CR'	Résidu conjugué
/'GMRES'	'Generalised Minimal RESidual'
/'TFQMR'	'Transpose Free Quasi Minimal Residual'

Les méthodes 'CG' et 'CR' sont à réserver aux modélisations conduisant à des matrices symétriques. En non symétrique, il faut faire appel aux autres méthodes. 'GMRES' et 'TFQMR', malgré leur surcoût, semblent les plus robustes.

◊ PRE_COND =
Nom des préconditionneurs de PETSc accessibles depuis *Code_Aster:*

/'LDLT_INC' [**DEFAUT**]	Factorisation incomplète par niveau
/'LDLT_SP'	Factorisation simple précision *via* l'outil externe MUMPS
/'JACOBI'	Pré-conditionneur diagonal standard
/'SOR'	'Successive Over Relaxation'

Tous les préconditionneurs de PETSc proposent une kyrielle de paramètres. Pour l'instant, seul ceux liés aux remplissage du ILU sont accessibles à l'utilisateur *Code_Aster*. Pour les autres, on garde les valeurs par défaut. D'autre part, en mode parallèle, seul JACOBI offre 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 processeur. C'est plus simple à mettre en oeuvre mais sous-optimal d'un point de vue algorithmique. Ces problèmes de préconditionnement parallèle rendent délicats toute évaluation du *speed-up*.
L'approche 'LDLT_SP' 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).

Remarque :
Avec les options par défaut ('CG'+'ILU'+'NIVE_REMPLISSAGE=0'+'RENUM='RCMK') on retrouve un algorithme très proche de celui du GCPC natif de Code_Aster.

◊ 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 §3.8). La référence est fixé 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 erronée ou non fournie, la librairie redimensionne d'elle-même les objets, mais cette opération est plus coûteuse.

◊ RENUM = / 'SANS'
/ 'RCMK' [**DEFAUT**]

Renuméroteur de type 'Reverse Cuthill-Mackee' (comme 'LDLT') pour limiter le remplissage de la factorisée incomplète. D'autre part, l'usage de ce renuméroteur adapté pour tenir compte des Lagranges, permet d'utiliser PETSc sur des systèmes indéfinis.

◊ 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.

◊ 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 (*via* PETSC_DEFAULT/maxits=10^5).

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

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

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

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

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

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

$\mathbf{r}_m$ 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 le nombre maximal d'itérations définies ci-dessus.

◊ RENUM Voir [§3.4].

◊ STOP_SINGULIER Voir [§3.3].

◊ NPREC Voir [§3.3].

◊ 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 interface. 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 $10+\text{nombre\_itérations\_FETI}*4$. 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
/ 'FFFFFFFFFFFFFF' [**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**]

A 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:

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