Manuel d'Utilisation
Fascicule U4.5- : Méthodes de résolution
Document : U4.50.01

Mot clé S0LVEUR

1 But

Choisir le mode de stockage des matrices et l'algorithme de résolution. Ce mot clé facteur se retrouve dans un certain nombre de commandes conduisant à la résolution de systèmes linéaires. Pour les algorithmes de résolution il permet de choisir entre factorisation "classique" de type "GAUSS" (`'LDLT'`), factorisation multi-frontale (`'MULT_FRONT'` ou `'MUMPS'`), gradient conjugué préconditionné ILU(k) (`'GCPC'`) ou solveur FETI par décomposition de domaines (`'FETI'`).

Pour chaque type de solveur, certains paramètres numériques facultatifs sont accessibles et sont décrits ici. Par défaut, c'est le solveur `'MULT_FRONT'` qui est utilisé. Le solveur '`FETI`', quant à lui, est encore en phase de développement, il n'est donc pas conseillé de l'utiliser sans conseils préalables de l'équipe de développement.

2 Syntaxe

```
◊  SOLVEUR = _F (

   # Factorisation de type "multi-frontale" :
   /  METHODE = 'MULT_FRONT' ,                              [DEFAUT]

      # Paramètre numérique
            ◊  RENUM           =  /  'METIS' ,             [DEFAUT]
                                  /  'MD'    ,
                                  /  'MDA'   ,
      # Paramètres fonctionnels
            ◊  STOP_SINGULIER  =  /  'OUI'  ,             [DEFAUT]
                                  /  'NON'  ,
            ◊  NPREC           =  /  8      ,             [DEFAUT]
                                  /  nprec  ,             [I]


   # Factorisation "classique" de type Gauss :
   /  METHODE = 'LDLT' ,

      # Paramètre numérique
            ◊  RENUM           =  /  'RCMK' ,             [DEFAUT]
                                  /  'SANS' ,
      # Paramètres fonctionnels
            ◊  STOP_SINGULIER  =  /  'OUI'  ,             [DEFAUT]
                                  /  'NON'  ,
            ◊  NPREC           =  /  8      ,             [DEFAUT]
                                  /  nprec  ,             [I]


   # Factorisation de type "multi-frontale"  avec MUMPS:
   /  METHODE = 'MUMPS'   ,
            ◊  TYPE_RESOL      =  /  'AUTO' ,             [DEFAUT]
                                  /  'NONSYM'  ,
                                  /  'SYMGEN'  ,
                                  /  'SYMDEF'  ,

            ◊  RESI_RELA    =     / 10^-6  ,              [DEFAUT]
                                  / resi ,                [R]

            ◊  PCENT_PIVOT     =  / 20   ,                [DEFAUT]
                                  / pcpiv,                [R]


   # Méthode itérative du gradient conjugué :
   /  METHODE = 'GCPC' ,

      # Paramètres numériques
            ◊  PRE_COND     =     'LDLT_INC' ,            [DEFAUT]
            ◊  NIVE_REMPLISSAGE = / 0    ,                [DEFAUT]
                                  / niv  ,
      # Paramètres fonctionnels
            ◊  NMAX_ITER    =     / 0,                    [DEFAUT]
                                  / niter,                [I]
            ◊  RESI_RELA    =     / 10^-6  ,              [DEFAUT]
                                  / resi ,                [R]
```

```
# Méthode de décomposition de domaines FETI :
/  METHODE = 'FETI' ,

   # Paramètres fonctionnels
         ♦  PARTITION     =      sdfeti
         ◊  NMAX_ITER     =      / 0,                [DEFAUT]
                                 / niter,            [I]
         ◊  RESI_RELA     =      / 10-6  ,           [DEFAUT]
                                 / resi ,            [R]
   # Paramètres du problème d'interface
         ◊  PRE_COND      =      / 'LUMPE' ,         [DEFAUT]
                                 / 'SANS'  ,
         ◊  SCALING       =      / 'MULT'  ,         [DEFAUT]
                                 / 'SANS'  ,
         ◊  TYPE_REORTHO_DD=     / 'GSM'   ,         [DEFAUT]
                                 / 'GS'    ,
                                 / 'IGSM'  ,
                                 / 'SANS'  ,
         ◊  NB_REORTHO_DD=       / 0,                [DEFAUT]
                                 / nb_reortho,       [I]
   # Paramètres des problèmes locaux
         ◊  RENUM            =   /  'METIS' ,        [DEFAUT]
                                 /  'MD'    ,
                                 /  'MDA'   ,
         ◊  STOP_SINGULIER   =   /  'OUI'  ,         [DEFAUT]
                                 /  'NON'  ,
         ◊  NPREC            =   /  8      ,         [DEFAUT]
                                 /  nprec  ,         [I]
   # Paramètres de tests, divers
         ◊  VERIF_SDFETI  =      /'OUI' ,            [DEFAUT]
                                 /'NON' ,
         ◊  TEST_CONTINU     =   /10-6   ,           [DEFAUT]
                                 /test_continu,      [R]
         ◊  STOCKAGE_GI      =   /'CAL' ,            [DEFAUT]
                                 /'OUI',
                                 /'NON',
         ◊  INFO_FETI        =   /'FFFFFFFFFF',      [DEFAUT]
                                 /info_feti,         [K9]
   # Paramètre pour le parallélisme
         ◊  NB_SD_PROC0      =   / 0,                [DEFAUT]
                                 / nb_sdproc0,       [I]
   # Paramètre pour les accélérations (problème de type multiples seconds membres)
         ◊  ACCELERATION_SM  =   /'OUI' ,            [DEFAUT]
                                 /'NON' ,
         ◊  NB_REORTHO_INST  =   /0 ,                [DEFAUT]
                                 /nb_reortho_inst,   [I]


# Paramètre commun à tous les solveurs
◊  SYME      =   /   'NON'  ,                       [DEFAUT]
                 /   'OUI'  ,

                 ),
```

3 Opérandes

3.1 Opérande METHODE

◊ METHODE =

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

/ 'MULT_FRONT' (**défaut**) Solveur direct de type 'multi-frontale'. Le stockage matriciel est 'MORSE' et donc 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). La matrice initiale est stockée dans un seul objet JEVEUX et sa factorisée est répartie sur plusieurs, donc peut être déchargée partiellement et automatiquement sur disque.

/ 'LDLT' Solveur direct avec factorisation de Crout par blocs (sans pivotage). Le stockage matriciel est 'ligne de ciel' ou 'SKYLINE'. On a une pagination de la mémoire complètement paramétrable (la matrice est décomposée en blocs gérés en mémoire de façon indépendante et déchargés sur disque au fur et à mesure) qui permet de passer de gros cas mais qui se paye par des accès disques coûteux.

/ 'MUMPS' Solveur direct de type 'multi-frontale' avec pivotage. Ce solveur est obtenu en se "branchant" sur la bibliothèque MUMPS développée par CERFACS-ENSEEIHT-INRIA-Parallab (voir copyright plus bas). 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...).
Informatiquement, ce solveur pose deux problèmes : il nécessite un compilateur fortran90 et il faut partager la mémoire entre JEVEUX et la bibliothèque MUMPS (usage du bouton "mem_aster" du menu OPTIONS d'ASTK).

/ 'GCPC' Solveur itératif de type gradient conjugué avec préconditionnement ILU(k). Le stockage de la matrice est alors 'MORSE'. La matrice initiale et sa factorisée incomplète sont stockées, chacune, dans un seul objet JEVEUX.

/ 'FETI' Solveur par décomposition de domaines de type FETI : gradient conjugué préconditionné projeté (GCPPC) pour le problème d'interface et solveur direct multi-frontale pour les inversions des matrices de rigidité locales.
Les problèmes locaux étant inversés par la multi-frontale 'MULT_FRONT', leurs matrices associées, matrices de rigidité locales et factorisées, sont traitées comme tel (cf. ci-dessus). Cette méthode est parallélisée en mémoire distribuée (MPI).

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

	Robustesse	Mémoire (RAM/Disque)	CPU	Paramétrage	Petit cas (<10^3 DDL)	Cas standards (<10^6 DDL)	(Très) gros cas (>10^6 DDL)
MULT_FRONT DEFAUT	Bonne	RAM : faible Disque : importante	Bon	Rien à faire	non	oui	Oui et plutôt avec la version parallèle
LDLT	Bonne	Du fait de la pagination on peut moduler la répartition RAM/disque	Coûteux	Rien à faire	oui	non	non
MUMPS	Très bonne	RAM : importante Disque : faible	Bon	Rien à faire	oui	oui	non
GCPC	Très variable	Très variable suivant le niveau de préconditionnement	Très variable suivant le niveau de précondi tionnement	A adapter au cas par cas	oui	Oui si thermique ou pb mécanique bien conditionné	Plutôt non
FETI	Plutôt bonne	RAM : faible Disque : importante en séquentiel, plus faible en parallèle.	Bon	A adapter au cas par cas	non	Oui avec au moins **10^4** DDL par sous-domaine et une interface de proportion raisonnable par rapport à la taille du problème (<10%)	Oui avec au moins **10^4** DDL par sous-domaine et avec une interface de proportion raisonnable par rapport à la taille du problème (<10%)

3.2 METHODE : 'MULT_FRONT'

◊ RENUM =

Cet argument permet de renuméroter les nœuds du modèle :

/ '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 ddls).

/ 'METIS' (**défaut**) Autre méthode de numérotation basée sur une dissection emboîtée. Cette méthode n'est possible que sur le serveur de calcul dédié au projet Aster (Alpha Serveur TRU64) sauf à installer soi-même l'exécutable METIS. Sur cette machine, c'est la méthode la plus efficace (en temps CPU et en mémoire).

◊ STOP_SINGULIER = 'OUI' (**défaut**) / 'NON'

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, sinon l'exécution se poursuit avec émission d'une alarme.

Remarque :

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), ...*

◊ NPREC = nprec (**défaut**=8)

C'est le nombre qui sert à déterminer si la matrice est singulière (ou non) (cf. mot clé STOP_SINGULIER ci-dessus).

3.3 METHODE : 'LDLT'

◊ RENUM =

Cet argument permet de renuméroter si on le désire les nœuds du modèle :

'SANS' On garde l'ordre initial donné dans le fichier de maillage,

'RCMK' (**défaut**) "Reverse Cuthill-MacKee", cet algorithme de renumérotation est souvent efficace pour réduire la place nécessaire au stockage "ligne de ciel" de la matrice assemblée et pour réduire le temps nécessaire à la factorisation de la matrice.

◊ STOP_SINGULIER

Voir [§3.2].

◊ NPREC

Voir [§3.2].

3.4 METHODE : 'MUMPS'

3.4.1 Généralités

Le solveur MUMPS développé par CERFACS-ENSEEIHT-INRIA-Parallab est un solveur direct de type multi-frontale parallélisé (MPI) et robuste car il permet de pivoter les lignes et colonnes de la matrice lors de la factorisation numérique.

Bien qu'il soit un solveur direct, son usage dans *Code_Aster* s'apparente plus au solveur itératif GCPC : on ne peut pas l'utiliser dans les opérateur modaux ni en conjonction du mot clé STOP_SINGULIER='DECOUPE'.
La raison en est que MUMPS (appelé par *Code_Aster*) ne donne pas d'information sur la qualité de la factorisation (comme le font MULT_FRONT et LDLT : voir le mot clé NPREC). On ne sait donc pas déterminer si une matrice est "proche" de la singularité lors de sa factorisation.

3.4.2 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 la 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.

Si l'utilisateur laisse la valeur par défaut ('AUTO'), 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 de pivotage en brisant la symétrie initiale.

3.4.3 RESI_RELA

Ce mot clé permet de choisir la précision attendue pour la résolution. Cette valeur est le résidu relatif maximum acceptable (1.d-6 par défaut). Si cette précision n'est pas atteinte, le code s'arrête en erreur fatale.

3.4.4 PCENT_PIVOT

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 20% 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 en réalité 120. Par la suite, si le nombre de pivotages est plus important que prévu, la place mémoire allouée sera insuffisante et le code s'arrêtera en erreur fatale en demandant d'augmenter PCENT_PIVOT. Pour certains calculs, il faut "pousser" PCENT_PIVOT jusqu'à 500 ! ce qui veut dire que le sur-coût du pivotage est de 500%.

3.4.5 COPYRIGHT

```
COPYRIGHT (c) 1996-2003 P. R. Amestoy, I. S. Duff, J. Koster,
                        J.-Y. L'Excellent

  CERFACS      , Toulouse    (France)  (http://www.cerfacs.fr)
  ENSEEIHT-IRIT, Toulouse    (France)  (http://www.enseeiht.fr)
  INRIA                      (France)  (http://www.inria.fr)
  PARALLAB     , Bergen      (Norway)  (http://www.parallab.uib.no)

 All rights reserved.
```

```
  Your use or distribution of the package implies that you agree
  with this License. Up-to-date copies of the MUMPS package can be
  obtained from the Web page http://www.enseeiht.fr/apo/MUMPS/

  This package is provided to you free of charge. It was
  initially based on public domain software developed during
  the European Esprit IV project PARASOL (1996-1999).

  THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY
  EXPRESSED OR IMPLIED.  ANY USE IS AT YOUR OWN RISK.

  Permission is hereby granted to use or copy this
  package provided that the Copyright and this License is
  retained on all copies and that the package is used
  under the same terms and conditions. User documentation
  of any code that uses this software should include this
  complete Copyright notice and this License.

  You can modify this code but, at no time shall the right
  or title to all or any part of this package pass to you.
  All information relating to any alteration or addition
  made to this package for the purposes of extending the
  capabilities or enhancing the performance of this package
  shall be made available free of charge to the authors for
  any purpose.

  You shall acknowledge (using references [1] and [2])
  the contribution of this package in any publication
  of material dependent upon the use of the package.
  You shall use reasonable endeavours to notify
  the authors of the package of this publication.


  [1] P. R. Amestoy, I. S. Duff and  J.-Y. L'Excellent (1998),
  Multifrontal parallel distributed symmetric and unsymmetric solvers,
  in Comput. Methods in Appl. Mech. Eng., 184,  501-520 (2000).
  An early version appeared as a Technical Report ENSEEIHT-IRIT (1998)
  and is available at http://www.enseeiht.fr/apo/MUMPS/.

  [2] P. R. Amestoy, I. S. Duff, J. Koster and  J.-Y. L'Excellent,
  A fully asynchronous multifrontal solver using distributed dynamic
  scheduling, SIAM Journal of Matrix Analysis and Applications,
  Vol 23, No 1, pp 15-41 (2001).
  An  early version appeared as a Technical Report ENSEEIHT-IRIT,
  RT/APO/99/2      (1999)             and      is      available      at
http://www.enseeiht.fr/apo/MUMPS/.

  None of the text from the Copyright notice up to and
  including this line shall be removed or altered in any way.
```

3.5 METHODE : 'GCPC'

◊ PRE_COND = 'LDLT_INC' (**défaut**)

Méthode de préconditionnement : la matrice de préconditionnement est obtenue par une décomposition $\mathbf{LDLT}$ incomplète de la matrice assemblée.

◊ NIVE_REMPLISSAGE = / 0 (**défaut**)
/ niv

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 ($\mathbf{A}$).

Si niv = 0

$\mathbf{P}$ a le même stockage que $\mathbf{A}$. La factorisation est incomplète car on n'utilise pour les calculs que les termes que l'on peut stocker dans $\mathbf{A}$. $\mathbf{P}$ représente donc une approximation (médiocre) de $\mathbf{A}^{-1}$; son stockage est donc plus raisonnable.

Si niv = 1

On stocke dans $\mathbf{P}$ en plus des termes qui avaient leur place dans le stockage initial, les "descendants" de première génération des termes initiaux. En effet lors de la factorisation, un terme nul dans $\mathbf{A}$ peut devenir non nul dans $\mathbf{P}$. On obtient ainsi le remplissage de niveau 1.

Si niv = 2, ...

Le même procédé est repris : la matrice $\mathbf{P}$ remplie au niveau niv-1 crée les termes de la matrice $\mathbf{P}$ au niveau niv.

Plus niv est grand, plus la matrice $\mathbf{P}$ est proche de $\mathbf{A}^{-1}$ 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.
Les premiers essais ont montré (approximativement) que la taille de $\mathbf{P}$ valait :

- 1*taille($\mathbf{A}$) pour niv = 0
- 3,5*taille($\mathbf{A}$) pour niv = 1
- 7,5*taille($\mathbf{A}$) pour niv = 2

Notre expérience de ce mot clé est encore limitée et nous conseillons 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...

◊ NMAX_ITER = niter (**défaut**=0)

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 (**défaut**=10^{-6})

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

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

$\mathbf{r}_m$ est le résidu à l'itération m

$\mathbf{b}$ est le second membre et $\|\ \|$ la norme euclidienne

3.6 METHODE : 'FETI'

♦ PARTITION = sdfeti

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

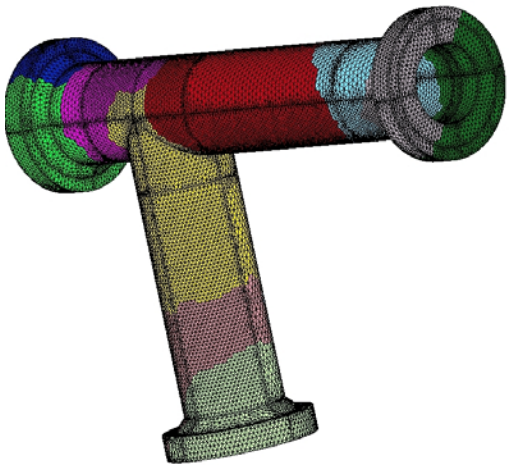

Figure 3.6-a : Exemple de structure (zone de mélange d'un circuit RRA) partitionnée en 10 sous-domaines

◊ NMAX_ITER = niter (**défaut**=0)

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.

◊ RESI_RELA = resi (**défaut**=10^{-6})

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 $\|\ \|$ la norme euclidienne

◊ PRE_COND =

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

'SANS'	Pas de préconditionnement.
'LUMPE' (**défaut**)	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' (**défaut)** 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' (**défaut)** Réorthogonalisation de Gram-Schmidt Modifié.
'IGSM' Réorthogonalisation de Gram-Schmidt Modifié Itératif.

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 (**défaut**=0)

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 [§3.2].

◊ STOP_SINGULIER

Voir [§3.2].

◊ NPREC

Voir [§3.2].

◊ VERIF_SDFETI = 'OUI' (**défaut**) / '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 incluse dans 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 (**défaut**=10^{-6})

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.

◊ STOCKAGE_GI = 'CAL' (**défaut**) / 'OUI' / 'NON'

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 + nombre_itérations_FETI * 4. Pour permettre à l'utilisateur d'adapter le compromis « taille mémoire/temps CPU », son stockage est paramétrable :

- Si STOCKAGE_GI = 'OUI', elle est calculée et stockée une fois pour toute. Cela nécessite plus de mémoire mais moins de temps calcul lorsqu'on s'en sert.
- Si STOCKAGE_GI = 'NON', c'est l'inverse, elle est recalculée à chaque fois que cela est nécessaire.
- Si STOCKAGE_GI = '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 (**défaut**='FFFFFFFFFF')
Chaîne de caractères permettant de paramétrer les affichages de l'algorithme FETI, de ses pré et post-traitements ainsi que de ses tests de cohérence. Ce monitoring est indépendant du mot-clé INFO. Etant souvent très verbeux et parfois coûteux en mémoire et en CPU, il doit être utilisé de préférence sur des petits cas et plutôt pour des activités de développements.

- Si INFO_FETI(1:1)='T' : déroulement général de l'algorithme.
- Si INFO_FETI(2:2)='T' : contenu des structures de données hors CHAM_NO et MATR_ASSE.
- Si INFO_FETI(3:3)='T' : contenu des structures de données CHAM_NO et MATR_ASSE.
- Si INFO_FETI(4:4)='T': affichage de variables intermédiaires.
- Si INFO_FETI(5:5)='T': détails des routines d'assemblages.
- Si INFO_FETI(6:6)= 'T': tests de validité des modes de corps rigides.
- Si INFO_FETI(7:7)= 'T': test de la définie-positivité de l'opérateur d'interface. On calcule alors les nmax_freq=min(nbi-2,nb_reortho) valeurs propres *via* l'algorithme IRAM[R5.01.01] en projetant sur un espace de taille dim_sous_espace=min(nbi,niter). Option licite uniquement en séquentiel.
- Si INFO_FETI(8:8)= 'T': test des orthogonalités du GCPPC.
- Si INFO_FETI(9:9)= 'T': profiling (temps CPU + système) des différentes étapes de résolution du problème d'interface FETI, le GCPPC, (projection, opérateur FETI, réorthogonalisation, redémarrages…).
- Si INFO_FETI(10:10)= 'T': affichages dédiés aux parallélisme MPI.
- Si INFO_FETI(11:11)= 'T' : affichages généraux (taille de l'interface, des matrices de rigidité locales et de leurs factorisées, nombre total de modes rigides…) et profiling des étapes amonts du solveur FETI (temps CPU + système des phases de calculs élémentaires, d'assemblages, de factorisations symbolique et numérique) détaillé par sous-domaine ou par processeur.

◊ NB_SD_PROC0 =nb_sdproc0 (**défaut**=0)

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 : nb_sdproc0>0 et nb_sdproc0<nbsd-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). 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).

'OUI' (**défaut)**	Accélération activée si les conditions sont réunies (voir plus bas).
'NON'	Accélération désactivée.

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 (**défaut**=0)

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 nb_reortho_inst = 0 alors ce nombre est calculé comme suit :

nb_reortho_inst = max(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 5$^{\text{iéme}}$ pas de temps on ne peut utiliser que les 4 pas de temps antérieurs) on le fixe dynamiquement à cette valeur :

nb_reortho_inst = num_pas_temps – 1.

3.7 Mot clé SYME

◊ SYME = / 'OUI'
/ 'NON'

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

Attention :

La symétrisation de la matrice $\mathbf{A}$ conduit donc à résoudre un autre problème 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 de converger vers la bonne solution. L'intérêt de ce mot clé est de gagner du temps lors de la résolution des systèmes linéaires. Le tout est de savoir si la symétrisation perturbe beaucoup (ou non) la solution du système linéaire ? On peut citer (par exemple) le cas des modèles 3D (ou coque) avec pression suiveuse pour lesquels la symétrisation fait gagner beaucoup de temps.

4 Exemples

4.1 Solveur par défaut

Il n'y a rien à écrire ! Mais on peut aussi écrire : SOLVEUR=_F( )

4.2 Gradient conjugué

On veut utiliser le gradient conjugué. On pense que la convergence sera plus efficace si l'on autorise un pré-conditionnement plus poussé (NIVE_REMPLISSAGE=1).

```
SOLVEUR=_F(METHODE ='GCPC', NIVE_REMPLISSAGE=1,)
```