# Macro-commande MACR_RECAL

## 1 But

Recaler des résultats de calculs sur des résultats expérimentaux ou sur d'autres résultats de calculs.

Considérons d'une part un ou plusieurs résultats d'essais et d'autre part un ou plusieurs calculs *Code_Aster* modélisant ces essais. MACR_RECAL permet de déterminer les paramètres de ces calculs (qui peuvent être des paramètres de loi de comportement, de chargement, etc ...) décrivant au mieux les essais.

Pour plus de précisions sur l'algorithmie mise en œuvre, se reporter à [R4.03.06].



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

# Table des matières

1 But....1
2 Syntaxe....4
3 Présentation générale....6
3.1 Principe du recalage....6
3.2 Organisation du recalage....6
3.3 Cas particulier d'utilisation : mode EXTERNE....7
3.4 Cas particulier du recalage d'un modèle dynamique....8
4 Opérandes....9
4.1 Opérande UNITE_ESCL....9
4.2 Opérandes RESU_EXP,RESU_CALC,FONC_EXP,NOM_FONC_CALC, PARA_X,PARA_Y,POIDS....9
4.3 Opérandes LIST_PARA, PARA_OPTI, NOM_PARA, VALE_INI, VALE_MIN,VALE_MAX....11
4.4 Opérande UNITE_RESU....11
4.5 Opérande ITER_MAXI....11
4.6 Opérande ITER_FONC_MAXI....12
4.7 Opérande RESI_GLOB_RELA....12
4.8 Opérande TOLE_FONC....12
4.9 Opérande TOLE_PARA....12
4.10 Opérande PARA_DIFF_FINI....12
4.11 Opérande GRAPHIQUE....12
4.12 Opérande METHODE....13
4.13 Mot-clé CALCUL_ESCLAVE....14
4.13.1 Opérande LANCEMENT....14
4.13.2 Opérande INCLUSION....14
4.13.3 Opérande DISTRIBUTION, MODE, MEMOIRE, TEMPS, CLASSE, UNITE_SUIVI....14
4.14 Opérandes NB_PARENTS et NB_FILS....15
4.15 Opérande ECART_TYPE....15
4.16 Opérande ITER_ALGO_GENE....15
4.17 Opérande RESI_ALGO_GENE....15
4.18 Opérande GRAINE....15
4.19 Opérande DYNAMIQUE....16
4.20 Opérande GRADIENT....16
5 Précautions d'emploi....18
6 Exemples d'utilisation....19
6.1 Identification des paramètres d'une loi de comportement élastoplastique sur un essai de traction ....19
6.1.1 Position du problème....19
6.1.2 Mise en données....19
6.1.2.1 Expérience....19



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

6.1.2.2 Calcul....20
6.1.2.3 MACR_RECAL....20
6.1.2.4 Mise en données alternative (listes Python)....21
6.1.2.5 Astk....21
6.1.3 Résultats....22
6.2 Identification des paramètres d'un modèle dynamique en exploitant la sensibilité des modes propres....25
6.2.1 Mise en données....25
6.2.2 Calcul esclave....26
6.3 Identification paramétrique de système conservatif en exploitant l'orthogonalité des modes propres....29
6.3.1 Mise en données....29
6.3.2 Calcul esclave....29
6.4 Identification paramétrique de système dissipatif en exploitant les relations de norme des déformées modales....30
6.4.1 Mise en données....30
6.4.2 Calcul esclave....30
7 Utilisation du mode EXTERNE....31
7.1 Avertissement....31
7.2 Méthodologie....31
7.2.1 Principe....31
7.2.2 Utilisation du fichier de lancement externe recal.py....32
7.3 Exemple : module d'optimisation de Matlab©....34



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

# 2 Syntaxe

```
lr =  MACR_RECAL                                                    [listr8]
          (  ♦  UNITE_ESCL       =  uni                             [I]
             ♦  /  RESU_EXP         =  resu_exp                     [assd]
                   RESU_CALC        =  resu_calc                    [assd]
                /  COURBE = _F(
                   ♦  FONC_EXP      =  fonc_exp                     [sd_fonction]
                   ♦  NOM_FONC_CALC =  nom_fonc_calc                [Kn]
                   ♦  PARA_X        =  para_X                       [Kn]
                   ♦  PARA_Y        =  para_Y                       [Kn]
                   ◊  POIDS         =  val_poids                    [R]
                   )
             ◊  LIST_POIDS          =  poids                        [assd]
             ♦  /  LIST_PARA        =  list_para                    [assd]
                /  PARA_OPTI = _F(
                   ♦  NOM_PARA      =  nom_para                     [Kn]
                   ♦  VALE_INI      =  val_ini                      [R]
                   ♦  VALE_MIN      =  val_min                      [R]
                   ♦  VALE_MAX      =  val_max                      [R]
                   )

             ◊  UNITE_RESU      =  /91                              [défaut]
                                   /uni_r                           [I]
             ◊  ITER_MAXI       =  /10                              [défaut]
                                   /it                              [I]
             ◊  ITER_FONC_MAXI  =  /100                             [défaut]
                                   /it                              [I]

             ◊  RESI_GLOB_RELA  =  /1.E-3                           [défaut]
                                   / resi                           [R]
             ◊  TOLE_PARA       =  /1.E-8                           [défaut]
                                   / resi                           [R]
             ◊  RTOLE_FONC      =  /1.E-8                           [défaut]
                                   / resi                           [R]
             ◊  PARA_DIFF_FINI  =  /1.E-5                           [défaut]
                                   / coef                           [R]
             ◊  GRAPHIQUE = _F(
                   ◊  UNITE     =  / 90                           [défaut]
                                   / uni_g                          [I]
                   ◊  FORMAT    =  / 'XMGRACE'                    [défaut]
                                   / 'GNUPLOT'
                   # Si FORMAT = XMGRACE
                   ◊  PILOTE    =  / ''                           [défaut]
                                   / 'POSTSCRIPT'                   [Kn]
                                   / 'EPS'
                                   / 'MIF'
                                   / 'SVG'
                                   / 'PNM'
                                   / 'PNG'
                                   / 'JPEG'
                                   / 'PDF'
                                   / 'INTERACTIF'
                                   / 'INTERACTIF_BG'
                   )

             ◊  AFFICHAGE =   / 'TOUTE_ITERATION'                   [défaut]
                              / 'ITERATION_FINALE'
```



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

```
◊  METHODE       =  / 'LEVENBERG'                [défaut]
                    / 'FMIN'
                    / 'FMINBFGS'
                    / 'FMINNCG'
                    / 'GENETIQUE'
                    / 'HYBRIDE'

◊  CALCUL_ESCLAVE   = _F(
      ◊  LANCEMENT = / 'INCLUSION'                 [défaut]
                     / 'DISTRIBUTION'
      # Si distribution
      ◊  MODE          = / 'INTERACTIF'            [défaut]
                         / 'BATCH'
      ◊  MEMOIRE       = mémoire                   [I]
      ◊  TEMPS         = temps                     [I]
      ◊  CLASSE        = classe                    [Kn]
      ◊  NMAX_SIMULT   = nmax                      [I]

      # Si MODE = INTERACTIF
      ◊  UNITE_SUIVI = unite                       [I]

# si METHODE = GENETIQUE, HYBRIDE
◊  NB_PARENTS       =  /10                         [défaut]
                       / nb_parents                [I]
◊  NB_FILS          =  /5                          [défaut]
                       / nb_fils                   [I]
◊  ECART_TYPE       =  /1.                         [défaut]
                       / ecart                     [R]
◊  ITER_ALGO_GENE   =  /10                         [défaut]
                       / itergene                  [I]
◊  RESI_ALGO_GENE   =  /1.E-3                      [défaut]
                       / resige                    [R]
◊  GRAINE           =  graine                      [I]

◊  DYNAMIQUE = _F(
      ◊  MODE_EXP  = mode_exp                      [assd]
      ◊  MODE_CALC = mode_calc                     [assd]
      ◊  APPARIEMENT_MANUEL  = / 'NON'             [défaut]
                               / 'OUI' )

◊  INFO          =  / 1
                    / 2                          [défaut]

# si METHODE = FMINBFGS
◊  GRADIENT      =  / 'NON_CALCULE'              [défaut]
                    / 'NORMAL'
                    / 'ADIMENSIONNE'

# si METHODE = FMINNCG
♦  GRADIENT      =  / 'NORMAL'
                    / 'ADIMENSIONNE'

# si METHODE = LEVENBERG
◊  GRADIENT      =  / 'NORMAL'                   [défaut]
                    / 'ADIMENSIONNE'
);
```



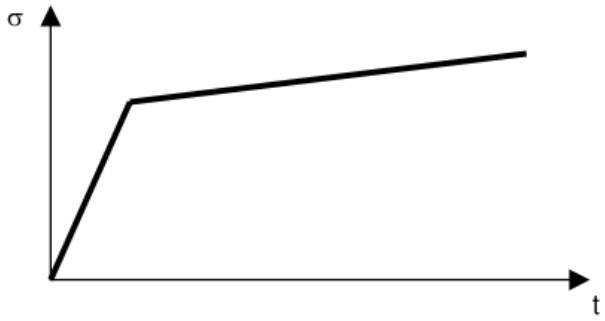
Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

# 3 Présentation générale

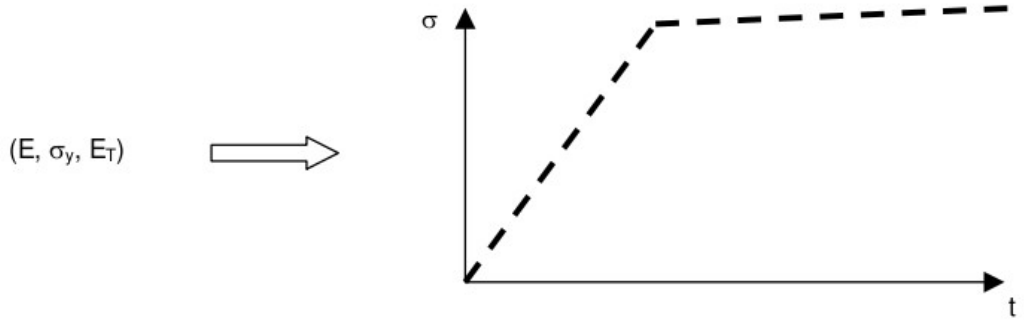
## 3.1 Principe du recalage

Considérons le problème modèle d'identification des caractéristiques élastoplastiques $E$, $\sigma_y$, $E_T$ (respectivement module d'Young, limite d'élasticité et module d'écrouissage) d'un matériau sur un essai de traction uniaxiale.

On a d'une part la courbe de traction expérimentale donnant l'évolution de la contrainte en fonction du temps et qui est une donnée :




On a d'autre part une **fonction** des 3 paramètres qui pour chaque valeur du triplet $E$, $\sigma_y$, $E_T$ renvoie une courbe de traction calculée :




L'objectif du recalage est alors de répondre à la question :

*Quelles sont les valeurs de* $(E, \sigma_y, E_T)$ *décrivant au mieux mon expérience ?*

## 3.2 Organisation du recalage

Pour mener à bien un recalage, il est nécessaire de disposer de l'ensemble des informations suivantes :

- les $N$ courbes expérimentales (à chacune de ces courbes peut être attribué un poids arbitraire),



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

- les $P$ paramètres à recaler ainsi que, pour chacun, une estimation de sa valeur initiale, sa valeur minimale et sa valeur maximale,
- le fichier de commandes modélisant les $N$ essais que l'on veut recaler,
- les noms des $N$ grandeurs à extraire du fichier de commandes ci-dessus et qui seront recalées sur les $N$ courbes expérimentales. Ces grandeurs doivent être contenues dans une table issue de POST_RELEVE_T.

La mise en données de ces informations nécessite alors l'organisation suivante :

- un fichier de commandes dit **maître** contenant les $N$ courbes expérimentales, les $P$ paramètres, les noms des grandeurs à recaler ainsi que d'autres informations propres au recalage, le tout renseigné dans MACR_RECAL. Les différents formats utilisés sont précisés dans ce qui suit,
- un fichier de commandes dit **esclave** modélisant les essais expérimentaux.

En effet, le recalage est un processus **itératif** : le fichier maître exécute le fichier esclave, il récupère les $N$ courbes calculées avec les valeurs courantes des $P$ paramètres, il compare les valeurs des courbes calculées à celles des courbes expérimentales, il en déduit de nouvelles valeurs pour les $P$ paramètres et relance le fichier esclave. Ce processus continue jusqu'à obtention de la convergence.

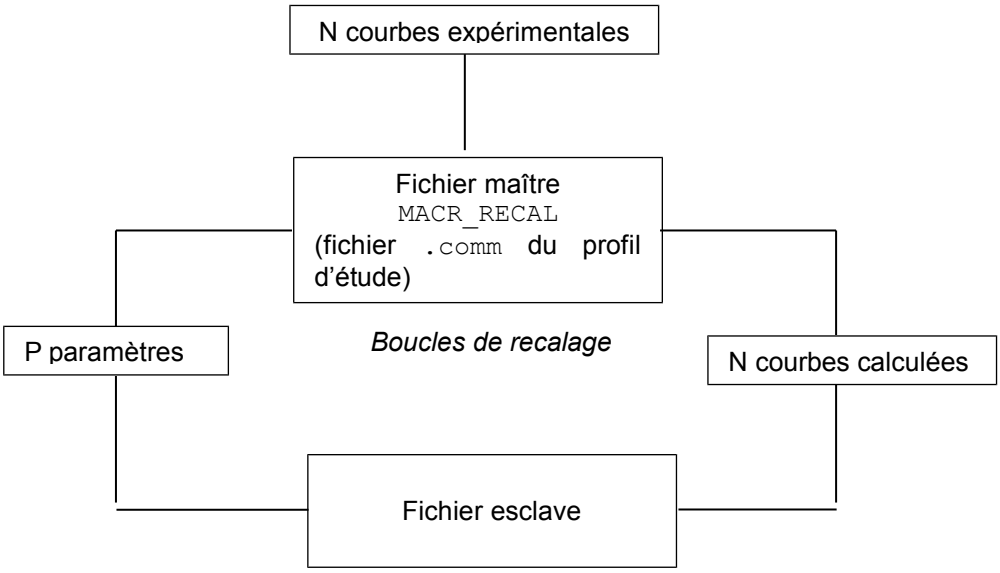



**Figure 3.2-a. Schéma de fonctionnement de la procédure de recalage pour le cas classique**

Dans la partie suivante sont décrites les opérandes de MACR_RECAL. On y fait référence à quelques notions du langage Python. Il n'est cependant nullement nécessaire de connaître Python pour utiliser cette macro commande. La partie « Exemple d'utilisation » est là pour éclairer l'utilisateur.

La structure de données produite est une liste de réels contenant les valeurs des paramètres à convergence en cas de convergence ou à la dernière itération dans le cas contraire.

## 3.3 Cas particulier d'utilisation : mode EXTERNE

Dans ce mode d'utilisation, l'algorithme d'optimisation est externe à *Code_Aster*.



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

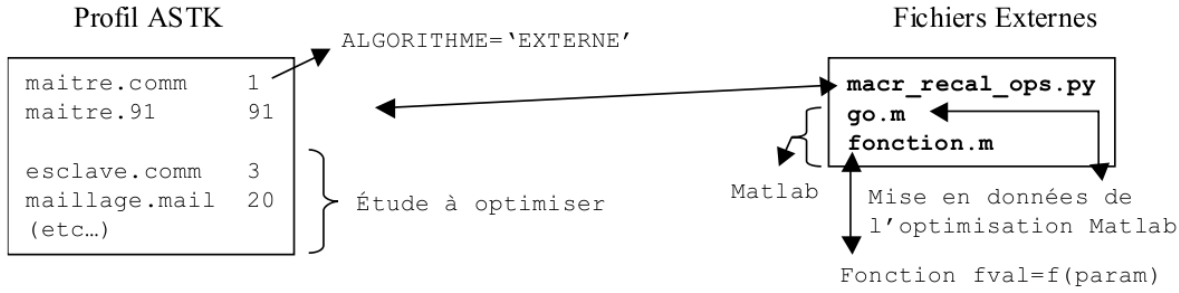



**Figure 3.3-a : MACR_RECAL méthode EXTERNE**

Code_Aster n'est qu'une boite noire qui prend en entrée un fichier texte contenant la liste des valeurs de paramètres, effectue le calcul de la fonctionnelle pour ce jeu de paramètres, réalise éventuellement le calcul des gradients par rapport aux paramètres, et renvoie à l'algorithme externe, par l'intermédiaire d'un fichier texte, la valeur de la fonctionnelle et éventuellement des gradients.

Dans ce mode de fonctionnement, le paragraphe 3.1 reste valable. Pour plus de détails, on renvoie l'utilisateur au paragraphe 7 .

## 3.4 Cas particulier du recalage d'un modèle dynamique

Dans le cas du recalage des paramètres d'un modèle dynamique en utilisant des données expérimentales issues de l'analyse dynamique, il existe quelques particularités pour la mise en œuvre. Les données expérimentales sont dans ce cas des fréquences propres et des vecteurs propres (déformées modales) contenues dans un concept `mode_meca` issu de la mesure.

L'utilisateur dispose de ce concept à partir d'un logiciel d'acquisition de données habituellement en format « `.unv` » et il doit construire un modèle dit « expérimental » afin de l'exploiter dans l'environnement Code_Aster. Ce modèle « expérimental» contient entre autres le maillage expérimental, c'est à dire le maillage des capteurs, plus grossier que le maillage du modèle numérique, qui doit être disponible également pour l'étude de recalage.

Donc l'utilisateur n'a plus à fournir les courbes expérimentales dans le fichier maître mais il va renseigner ici les noms des concepts qui les contiennent, concepts qui seront extraits par un premier calcul esclave depuis le fichier externe « unv ». Le format utilisé pour transmettre cette information à `MACR_RECAL` est le même que celui pour les `RESU_CALC` : une liste `Python` de $N$ listes `Python` contenant les noms des tables et des colonnes contenant les réponses expérimentales.

Ci-dessous en présente le schéma du processus de recalage dans le cas de la dynamique:



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

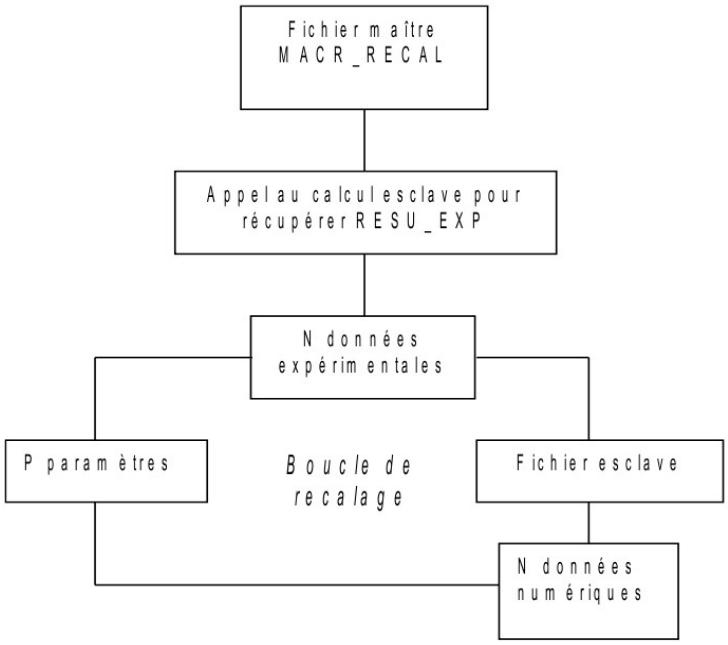



**Figure 3.4-a. Schéma de fonctionnement de la procédure de recalage en dynamique**

Deux autres particularités concernent le fichier de commande de calcul esclave:
- il contient aussi les commandes nécessaires à la construction du modèle expérimental. Le concept mode_meca utilisable pour extraire RESU_EXP est fourni par LIRE_RESU ;
- Les grandeur correspondant aux réponses expérimentales et numériques sont toujours contenues dans des tables mais plus nécessairement issues de POST_RELEVE_T. Les réponses liées aux déformées modales sont issues soit de CREA_TABLE (pour le critère de MAC expérimental), soit de MAC_MODES (pour le critère de MAC numérique). On rappelle que le critère de MAC [U4.52.15] sert à comparer deux bases modales, il s'agit ici de celle expérimentale et de celle numérique.

# 4 Opérandes

## 4.1 Opérande UNITE_ESCL

♦ UNITE_ESCL
Numéro d'unité logique du fichier esclave, attribué dans l'interface Astk (colonne UL). L'extension de ce fichier peut être quelconque.

## 4.2 Opérandes RESU_EXP,RESU_CALC,FONC_EXP,NOM_FONC_CALC, PARA_X,PARA_Y,POIDS

L'utilisateur dispose de 2 modes de saisie pour renseigner les données expérimentales et les noms des tables et des colonnes des résultats calculés.
Il a le choix entre utiliser :
- RESU_EXP et RESU_CALC et éventuellement LIST_POIDS qui nécessitent la saisie de listes python,
- les mot-clés FONC_EXP , NOM_FONC_CALC , PARA_X , PARA_Y , POIDS du mot-clé facteur COURBE qui ont l'avantage d'utiliser des concepts Aster.

◊ RESU_EXP

Nom de la liste Python de $N$ tableaux numpy contenant les $N$ courbes expérimentales. Pour un modèle statique la liste est définie préalablement sous la forme :



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

```
resu_exp=[   numpy.array([[x0,y0],
                          [x1,y1],
                           ...
                          [xn ,yn ]  ]),
         .....
             numpy.array([[u0 ,v0 ],
                          [u1 ,v1 ],
                          ...
                          [un,vn] ])
         ]
```

Pour le recalage d'un modèle dynamique avec des données expérimentales modales (fréquences et vecteurs propres), il s'agira du nom de la liste Python de $N$ listes Python contenant les noms des tables et des colonnes contenant les réponses expérimentales. Par exemple:

```
resu_exp=[['REPEX1','NUM_ORDR','FREQ'],['REPEX2','NUM_ORDR','MAC_EXP']]
```

◊ RESU_CALC

Nom de la liste de N listes Python contenant les noms des tables et des colonnes contenant les réponses numériques correspondant aux mesures expérimentales sur lesquelles on va effectuer le recalage. Par exemple :

```
resu_calc=[['TABLE1','INST','SIYY'], ['TABLE2','INST','V1'],,...])
```

◊ LIST_POIDS

Nom du tableau Numpy contenant les $N$ poids à affecter aux $N$ courbes expérimentales. Si le mot-clé n'est pas renseigné, alors chaque courbe a le même poids. La liste est définie préalablement sous la forme :

```
LIST_POIDS= numpy.array([p1, p2, ...])
```

◊ FONC_EXP

Nom de la fonction préalablement définie par l'opérateur DEFI_FONCTION qui correspond aux données expérimentales. Pour faire le lien avec RESU_EXP , la fonction fournie à ce mot-clé à l'itération $i$ correspond au $i$ ème tableau numérique de RESU_EXP.

◊ NOM_FONC_CALC

Chaîne de caractères correspondant au nom que l'on souhaite attribuer à la fonction après recalage des données expérimentales.

◊ POIDS

Valeur du poids à affecter à la courbe expérimentale. Si non renseigné, alors la valeur est 1.

◊ PARA_X

Nom du paramètre $X$ de la fonction calculée.

◊ PARA_Y

Nom du paramètre $Y$ de la fonction calculée. Par exemple:

```
COURBE=(
_F(FONC_EXP=fct1, NOM_FONC_CALC='TABLE1',PARA_X='INST',PARA_Y='SIYY',),
_F(FONC_EXP=fct2, NOM_FONC_CALC='TABLE2',PARA_X='INST',PARA_Y='V1',),)
```



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

## 4.3 Opérandes LIST_PARA, PARA_OPTI, NOM_PARA, VALE_INI, VALE_MIN,VALE_MAX

L'utilisateur dispose de 2 modes de saisie pour renseigner les noms et valeurs des paramètres.
Il a le choix entre utiliser :

- LISTE_PARA qui nécessitent la saisie de listes python,
- les mot-clés NOM_PARA, VALE_INI, VALE_MIN, VALE_MAX du mot-clé facteur PARA_OPTI qui ont l'avantage d'utiliser des concepts Code_Aster (valeurs numériques ou chaînes de caractères).

◊ LIST_PARA

Nom de la liste Python de $P$ listes Python contenant les noms des variables, leurs valeurs initiales, leurs valeurs minimales et leurs valeurs maximales. Cette liste est définie préalablement sous la forme :

```
List_para=[  ['PARA1__',INI_1,MIN_1,MAX_1],
             ['PARA2__',INI_2,MIN_2,MAX_2],
             ....
             ['PARAP__',INI_P,MIN_P,MAX_P]]
```

**Attention :**

*On demande que les noms des variables se terminent par deux blancs soulignés (par exemple : YOUN__ ).*

**Remarque :**

*Les bornes ne sont pas gérées par les algorithmes FMIN, FMINBFGS et FMINNCG.*

◊ NOM_PARA

Nom du paramètre. La chaîne de caractère fournie à NOM_PARA à l'itération $i$ correspond au premier élément de la $i$ ème liste python de la liste fournie à LIST_PARA.

◊ VALE_INI

Valeur initiale du paramètre. Le réel fourni à NOM_PARA à l'itération $i$ correspond au second élément de la $i$ ème liste python de la liste fournie à LIST_PARA.

◊ VALE_MIN

Valeur minimale du paramètre. Le réel fourni à NOM_PARA à l'itération $i$ correspond au troisième élément de la $i$ ème liste python de la liste fournie à LIST_PARA.

◊ VALE_MAX

Valeur maximale du paramètre. Le réel fourni à NOM_PARA à l'itération $i$ correspond au quatrième élément de la $i$ ème liste python de la liste fournie à LIST_PARA.

## 4.4 Opérande UNITE_RESU

◊ UNITE_RESU
Numéro d'unité logique du fichier de résultat du recalage (évolution des paramètres au cours des itérations, critères de convergence).

## 4.5 Opérande ITER_MAXI

◊ ITER_MAXI
Nombre d'itérations maximales de recalage.



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

## 4.6 Opérande ITER_FONC_MAXI

◊ ITER_FONC_MAXI
Nombre d'évaluations maximales de la fonctionnelle.

## 4.7 Opérande RESI_GLOB_RELA

◊ RESI_GLOB_RELA
Résidu global relatif du recalage.
Cette valeur est disjointe de celle renseignée pour les solveurs non linéaires STAT_NON_LINE et DYNA_NON_LINE.

## 4.8 Opérande TOLE_FONC

◊ TOLE_FONC
Critère d'arrêt de l'algorithme de recalage basé sur la variation de la fonctionnelle d'une itération à l'autre. Ce critère correspond à la valeur absolue de la norme de ma fonctionnelle.

## 4.9 Opérande TOLE_PARA

◊ TOLE_PARA
Critère d'arrêt de l'algorithme de recalage basé sur la variation des paramètres d'une itération à l'autre. Ce critère correspond à la norme $L2$ : racine carrée de la somme des carrés des différences de chaque paramètres.

## 4.10 Opérande PARA_DIFF_FINI

◊ PARA_DIFF_FINI
Le recalage nécessite le calcul des dérivées des réponses par rapport aux paramètres.

Ce calcul est réalisé par différences finies. PARA_DIFF_FINIES correspond à $\alpha$ dans la formule suivante :

$$\frac{\partial f}{\partial x} \approx \frac{f(x+\alpha x)-f(x)}{\alpha x}$$

## 4.11 Opérande GRAPHIQUE

◊ UNITE
Numéro d'unité logique des graphiques produits au cours du recalage. A chaque itération, MACR_RECAL produit $N$ fichiers graphiques (dont le format est défini par le mot-clé PILOTE) représentant les $N$ courbes expérimentales et calculées.

◊ PILOTE
Type d'affichage des graphiques.
Si PILOTE = 'INTERACTIF' ou 'INTERACTIF_BG', xmgrace est ouvert de façon interactive avec le graphique. Avec 'INTERACTIF_BG', xmgrace est appelé en arrière plan, le calcul se poursuit.
Si PILOTE vaut 'POSTSCRIPT','EPS', 'MIF', 'SVG', 'PNM', 'PNG', 'JPEG' ou 'PDF' alors xmgrace est utilisé pour générer le fichier de format correspondant, dans l'unité logique définie par UNITE.

**Attention :**

*Le mode INTERACTIF n'est possible que lorsque le recalage tourne en interactif et non en batch.*



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

◊ FORMAT
Choix du logiciel d'affichage des courbes en mode interactif : xmgrace ou gnuplot. L'utilisation d'Xmgrace est bloquante : il faut fermer la fenêtre Xmgrace pour continuer l'exécution.

◊ AFFICHAGE
Affichage des courbes à chaque itération ou uniquement à la fin.

## 4.12 Opérande METHODE

◊ METHODE
Méthode ou algorithme d'optimisation choisi :
1) LEVENBERG (défaut) : algorithme de Levenberg-Marquardt. Cet algorithme est celui préconisé dans les problèmes de minimisation quadratique type moindres carrés, comme des problèmes de recalage de paramètres matériaux.
2) FMIN : Nelder-Mead Simplex algorithm (n'utilise que des estimations de la fonctionnelle).
3) FMINBFGS : méthode Quasi-Newton (utilise la fonctionnelle et le gradient de la fonctionnelle).
4) FMINNCG : méthode Line-search Newton Conjugate Gradient (utilise la fonctionnelle, le gradient de la fonctionnelle et son hessien).
5) GENETIQUE: algorithme évolutionnaire basé sur le mécanisme de la sélection et du remplacement. C'est un algorithme qui coute cher en temps CPU, on conseille son utilisation seulement pour une exploration grossière de l'espace des paramètres dans le cadre de la technique hybride de recalage présentée dans le point suivant.
6) HYBRIDE: technique qui combine le stochastique avec le déterministe - l'algorithme évolutionnaire avec l'algorithme de Levenberg – Marquardt
7) Le mode EXTERNE : cette méthode permet d'utiliser un algorithme d'optimisation externe à *Code_Aster*, par exemple Matlab ou Scilab, et d'utiliser *Code_Aster* uniquement pour l'estimation de la fonctionnelle et éventuellement du gradient par différences finies. Ce n'est pas un mot-clé de MACR_RECAL car le mode EXTERNE s'utilise directement par l'appel du fichier bibpyt/Macro/recal.py

**Concernant le choix de l'algorithme d'optimisation**, il est fortement conseillé d'opter pour l'algorithme par défaut, **Levenberg-Marquardt**. Celui-ci est très souvent supérieur aux algorithmes FMIN* pour des problèmes de minimisation de type moindres carrés, comme le recalage de paramètres. En effet, il utilise la fonctionnelle sous sa forme vectorielle alors que les autres algorithmes utilisent une fonctionnelle scalaire, par conséquent moins riche. De plus, il utilise une méthode de contraintes actives afin de gérer des bornes sur les paramètres, alors que les autres algorithmes ne gèrent pas les bornes.

Les autres algorithmes peuvent néanmoins être utiles dans les cas où Levenberg-Marquardt est mis en difficulté. Par exemple, l'algorithme FMIN n'utilise pas de gradients, dont l'évaluation peut dans certains cas très particuliers générer des problèmes numériques (paramètres très sensibles, trop peu de valeurs expérimentales, ou autres). L'algorithme FMIN est nettement plus lent, mais pourra arriver à converger (pour faire un parallèle, la problématique est similaire à celle d'utiliser la matrice élastique à la place de la matrice tangente dans STAT_NON_LINE).

Pour l'algorithme de Levenberg-Marquardt, le document [R4.03.06] décrit plus précisément l'algorithmie mathématique mise en jeu.

Les algorithmes FMIN* ont été repris intégralement d'un module Python distribué sur Internet (http://pylab.sourceforge.net) sous licence GPL par Travis E. Oliphant, par ailleurs contributeur principal du projet Python-Scipy et responsable du module d'optimisation de Scipy. Les détails d'algorithmie et d'implémentation peuvent être trouvés sur la page http://pylab.sourceforge.net.

La méthode HYBRIDE est conseillée lorsqu'on a un degré élevé d'incertitude sur les valeurs optimales des paramètres ou lorsque la fonctionnelle présente des nombreuses minima locaux. Ainsi, dans le cadre de cette méthode, on lance d'abord une recherche grossière avec l'algorithme évolutionnaire,



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

ce qui permettra d'éviter les minima locaux, suivi par un affinement de l'optimisation avec l'algorithme de Levenberg-Marquardt.

## 4.13 Mot-clé CALCUL_ESCLAVE

### 4.13.1 Opérande LANCEMENT

◊ LANCEMENT
Méthode de lancement des fichiers esclaves : inclusion ou distribution. Les deux modes ont des avantages et des inconvénients et le choix de l'un ou de l'autre dépend principalement des temps de calculs des fichiers esclaves, ainsi que de leur compatibilité avec le mode Inclusion.

### 4.13.2 Opérande INCLUSION

◊ INCLUSION
Dans ce mode, le fichier esclave est inclus. Il n'y a donc pas de perte de temps pour la génération d'une nouvelle étude, la création du répertoire temporaire d'exécution, etc.. En contre-partie, seule une étude esclave pourra passer en même temps sur la machine. D'autre part, certaines études esclaves (par exemple celles utilisant des fichiers de données inclus, ou des profils d'exécution un peu complexes) ne sont pas compatibles.

**Remarque :**

*Dans le mode INCLUSION, la commande Aster INCLUDE ne peux pas être présente dans le fichier esclave (incompatibilité avec le superviseur Aster).*
*Il faut donc remplacer la commande :*
*INCLUDE(UNITE=n)*
*par la commande :*
*execfile(fort.n)*
*où n est l'unité logique du fichier à inclure.*

### 4.13.3 Opérande DISTRIBUTION, MODE, MEMOIRE, TEMPS, CLASSE, UNITE_SUIVI

◊ DISTRIBUTION
Dans ce mode, à chaque itération de l'algorithme d'optimisation, les $N+1$ calculs esclaves (pour un recalage de $N$ paramètres) sont exécutés en parallèles, en batch ou en interactif, en utilisant le module de calculs distribués d'as_run. Comparé au mode Inclusion, chaque étude est légèrement plus longue à s'exécuter, car il faut régénérer une nouvelle étude *Code_Aster*, créer les fichiers temporaires, etc.. En revanche, comme les études sont lancées en parallèles, suivant les caractéristiques des études esclaves (taille et durée), plus le nombre de paramètres est grand et plus le mode Distribué prend de l'intérêt sur le mode Inclusion.

En mode distribué, des paramètres supplémentaires sont disponibles pour contrôler les exécutions des calculs esclaves :

◊ MODE : INTERACTIF ou BATCH
◊ MEMOIRE : mémoire en Mo
◊ TEMPS : temps en secondes
◊ CLASSE : classe de batch, permet de forcer les calculs à utiliser une classe spécifique, par exemple « distr » sur le serveur *Code_Aster*
◊ UNITE_SUIVI : si ce mot-clé est préciser, il défini l'unité logique du fichier du profil dans lequel seront stockés tous les fichiers output des jobs esclaves
◊ NMAX_SIMULT : nombre de calculs esclaves lancés en parallèle en mode distribution (si aucune valeur n'est renseigné, le code décide automatiquement de ce nombre)

Il est possible d'utiliser le parallélisme MPI pour les études esclaves. Dans ce cas, il est nécessaire de lancer le calcul maître avec une version MPI et sur un seul processeur (en mode



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

interactif ou en batch). Les caractéristiques MPI des calculs esclaves doivent être indiquées par les mot-clés suivants :

◊ MPI_NBCPU : nombre de processeurs MPI pour chacun des calculs esclaves lancés en parallèle

◊ MPI_NBNOEUD : nombre de nœuds pour chaque calcul esclave lancé en parallèle

A noter que des problèmes d'exploitation peuvent venir perturber le lancement des calculs esclaves distribués (par exemple, les classes de batch sont mal définies et les calculs ne peuvent s'exécuter). Dans ce cas, l'erreur obtenue dans le .mess du calcul maître peut être assez sobre (un message d'erreur du style « au moins un des calculs esclaves n'a pas pu démarrer »). Pour obtenir des informations supplémentaire concernant le module distribué d'as_run, il faut mettre simultanément UNITE_SUIVI et INFO=2.

## 4.14 Opérandes NB_PARENTS et NB_FILS

◊ NB_PARENTS
Pour la méthode GENETIQUE ou HYBRIDE, cet opérande défini la taille de la population des paramétrés. Initialement tous les individus sont identiques et initialisés avec les valeurs initiales des paramètres fournies par l'utilisateur dans LIST_PARA. Au cours de l'optimisation la population évolue, les individus les moins « adaptés » étant remplacés par des par d'autres qui ont fourni une meilleure valeur de la fonctionnelle.

◊ NB_FILS
Représente le taux de remplacement de la population. Plus exactement, le meilleur « parent » (celui pour lequel la valeur de la fonctionnelle est minime) a le droit de se reproduire générant donc NB_FILS « fils ». A ce moment la taille de la population est NB_PARENTS+NB_FILS. Suivant hiérarchie des valeurs de la fonctionnelle, seuls les meilleurs individus de toute cette population sont retenus et on revient à la taille initiale: NB_PARENTS.

## 4.15 Opérande ECART_TYPE

◊ ECART_TYPE
C'est la valeur de l'écart type que l'utilisateur impose pour les tirages au sort quasi-aléatoires des « fils ». Plus on veut explorer l'espace topologique des paramètres, plus il faut augmenter cette valeur. Corroboré à la taille de la population et au taux de remplacement, cet opérant permet de piloter l'algorithme évolutionnaire suivant la complexité du modèle et le degré d'incertitude sur les valeurs optimales des paramètres. Si on connaît peu de choses sur les valeurs des paramètres à recaler, il est recommandable d'utiliser une taille élevée de la population, un taux de remplacement également élevé et un écart type grand. La contrepartie sera un temps CPU très élevé.

## 4.16 Opérande ITER_ALGO_GENE

◊ ITER_ALGO_GENE
Nombre d'itérations maximales pour l'algorithme évolutionnaire. Si on utilise la méthode HYBRIDE, cette valeur (ou RESI_ALGO_GENE) va déterminer le passage à l'algorithme de Levenberg-Marquardt.

## 4.17 Opérande RESI_ALGO_GENE

◊ RESI_ALGO_GENE
Résidu relatif du recalage l'algorithme évolutionnaire. Si on utilise la méthode HYBRIDE, cette valeur (ou ITER_ALGO_GENE) va déterminer le passage à l'algorithme de Levenberg-Marquardt.

## 4.18 Opérande GRAINE



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

◊ GRAINE

Valeur imposée par l'utilisateur pour la graine du générateur des tirages au sort dans l'algorithme évolutionnaire. Si on renseigne une valeur pour ce mot-clé, on force le générateur de nombres aléatoires présent dans l'algorithme évolutionnaire de générer toujours les mêmes tirages au sort, donc on aura une répétition de la solution. Son emploi est réservé **seulement** aux cas-tests pour des raisons de suivi de la non-régression du code.

## 4.19 Opérande DYNAMIQUE

On renseigne cet opérande pour le recalage des paramètres d'un modèle dynamique par analyse modale.

◊ MODE_EXP

Nom du concept mode_meca qui contient les données modales expérimentales. Ce concept est extrait dans le calcul esclave par un LIRE_RESU.

◊ MODE_CALC

Nom du concept mode_meca qui contient les données modales numériques. Ce concept est calculé dans le fichier esclave par un CALC_MODES.

◊ APPARIEMENT_MANUEL

Choix d'afficher en mode interactif d'une fenêtre graphique qui permettra d'apparier manuellement les modes propres. On évite ainsi par exemple l'appariement mauvais automatique des modes propres double croisés.
La capture d'écran présentée dans la Figure 4.19-a illustre cette fenêtre graphique.

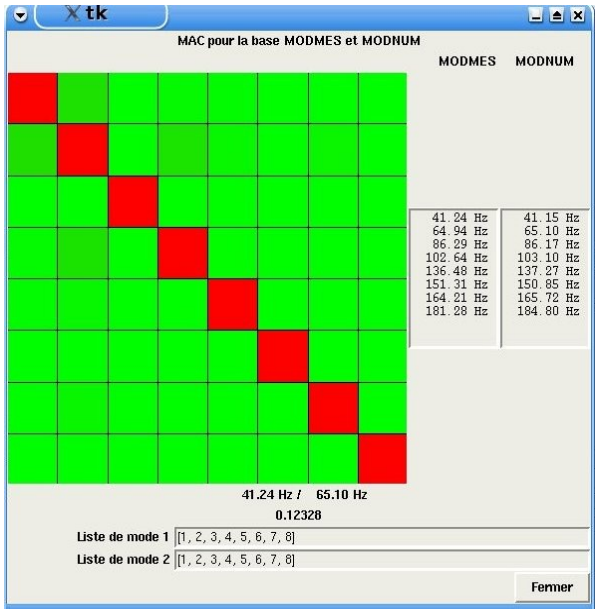



**Figure 4.19-a. Fenêtre graphique pour l'appariement manuel des MAC**

L'utilisateur voit ainsi, à chaque génération de l'algorithme évolutionnaire ou à chaque itération de l'algorithme de Levenberg-Marquardt, la matrice de MAC et il peut décider de changer d'appariement en modifiant l'ordre des modes dans les listes situées en bas de la fenêtre. Cette fenêtre est bloquante pour l'exécution des commandes *Code_Aster* donc il faut la fermer pour que le processus de recalage continue. La fermeture de la fenêtre en cliquant sur le bouton *Fermer* permet de récupérer les nouvelles listes des modes dont l'ordre a été éventuellement modifiée par l'utilisateur.

## 4.20 Opérande GRADIENT



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

◊ GRADIENT

Pour les méthodes FMINBFGS, FMINNCG ou EXTERNE, ce mot-clé permet d'indiquer à *Code_Aster* la façon de calculer les gradients (adimensionné ou non), ou bien de ne pas les calculer.

Note : pour les algorithmes qui utilisent les gradients, ceux-ci peuvent être calculés par différences finies automatiquement par *Code_Aster*, ou bien calculés dans le fichier esclave en utilisant, par exemple des calculs de sensibilité (mot-clé LIST_DERIV).

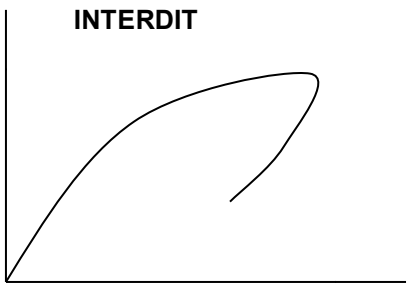


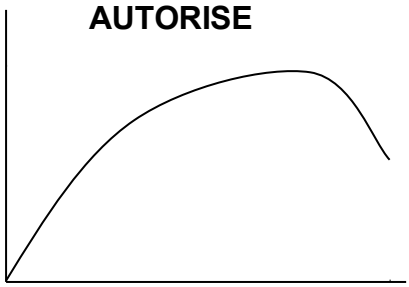
Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

# 5 Précautions d'emploi

Voici un ensemble de conseils **indispensables** à la bonne utilisation du recalage.

- Les courbes expérimentales sont définies comme des tableaux à deux colonnes : une pour les abscisses et une pour les ordonnées.
- Les courbes expérimentales doivent être des fonctions : à une abscisse ne doit correspondre qu'une ordonnée. Si un courbe expérimentale comporte des cycles, (par exemple contrainte en fonction de la déformation en charge-décharge), il faut alors scinder cette courbe paramétrée en deux courbes, exprimant d'une part les abscisses, d'autre part les ordonnées de la courbe cyclique en fonction du paramètre (par exemple déformation en fonction du temps et contrainte en fonction du temps).




- On doit recaler $N$ courbes calculées sur $N$ courbes expérimentales.
- La première courbe calculée sera recalée sur la première courbe expérimentale, la deuxième courbe calculée sera recalée sur la deuxième courbe expérimentale, et ainsi de suite dans l'ordre renseigné pour les opérandes RESU_EXP et RESU_CALC.
- Les grandeurs calculées renseignées sous l'opérande RESU_CALC doivent être issues de POST_RELEVE_T (sauf pour la dynamique où on peut avoir des tables issues de CREA_TABLE et MAC_MODES)
- Les paramètres du recalage doivent être déclarés en bloc au début du fichier de commandes esclave. Par exemple :

```
DEBUT();
DSDE__ = 200.;
YOUN__ = 8.E4;
SIGY__ = 10.;
......
```

- Les valeurs initiales des paramètres du recalage sont celles renseignées pour l'opérande LIST_PARA et non celles présentes dans le fichier esclave de l'utilisateur.
- A chaque itération de recalage, les calculs définis dans le fichier esclave doivent converger. Dans le cadre de recalage de calculs non linéaires, il est donc fortement recommandé d'utiliser la découpe automatique du pas de temps.
- Dans le cadre de recalage de calculs non linéaires avec découpe automatique du pas de temps, il est **indispensable** de définir une liste d'archivage sous l'opérande LIST_ARCH.
- Le recalage est un moyen puissant d'obtenir des valeurs de paramètres à partir d'essais. Il n'est cependant pas miraculeux : les courbes **expérimentales** doivent contenir suffisamment d'informations pour identifier les paramètres. Il est par exemple impossible d'identifier des paramètres élastoplastiques avec un essai restant dans le domaine élastique. Les essais expérimentaux doivent donc exciter les paramètres à identifier.
- Dans le même logique, il est souhaitable que les courbes **expérimentales** contiennent des points en nombre suffisant pour bien décrire l'action des paramètres à identifier.
- Enfin, dans le cas de l'utilisation de plusieurs courbes expérimentales, le fait qu'elles aient le même nombre de points équilibre l'information qu'elles apportent.



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

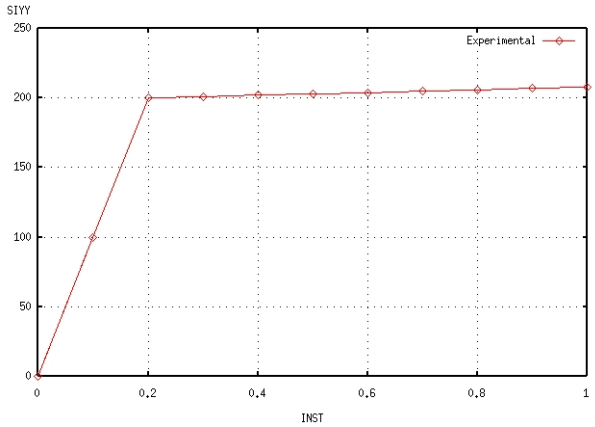
# 6 Exemples d'utilisation

## 6.1 Identification des paramètres d'une loi de comportement élastoplastique sur un essai de traction

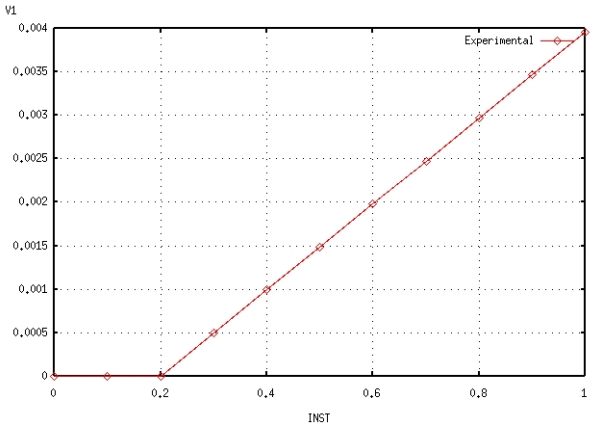
Cet exemple est traité par le test ZZZZ159A [V1.01.159].

### 6.1.1 Position du problème

On dispose des résultats d'un essai de traction. Il s'agit de l'évolution de la contrainte $\sigma_{yy}$ au cours du temps ainsi que de l'évolution de la déformation plastique cumulée $p$ au cours du temps.



Contrainte SIYY



Déformation plastique cumulée

On désire recaler sur ces essais le module de Young, la limite d'élasticité et la pente d'écrouissage d'une loi de comportement élastoplastique à écrouissage isotrope linéaire.

### 6.1.2 Mise en données

#### 6.1.2.1 Expérience

On commence tout d'abord par définir nos résultats d'essais. Ils consistent en deux courbes que l'on définit comme suit.

```
exper1=DEFI_FONCTION(NOM_PARA='INST',
                     NOM_RESU='SIYY',
                     VALE=(0.00000E+00  , 0.00000E+00 ,
                           5.00000E-02  , 5.00000E+01 ,
                           ...
                           9.50000E-01  , 2.07500E+02 ,
                           1.00000E+00  , 2.08000E+02 ),)

exper2=DEFI_FONCTION(NOM_PARA='INST',
                     NOM_RESU='V1',
                     VALE=(0.00000E+00  , 0.00000E+00 ,
                           5.00000E-02  , 0.00000E+00 ,
                           ...
                           9.50000E-01  , 3.71250E-03 ,
                           1.00000E+00  , 3.96000E-03 ),)
```



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

exper1 et exper2 sont donc des fonctions de Code_Aster, qui représentent respectivement la contrainte $\sigma_{yy}$ et la déformation plastique cumulée $p$.

#### 6.1.2.2 Calcul

On écrit ensuite le fichier de commandes *Code_Aster* esclave modélisant cet essai de traction où vont apparaître nos 3 paramètres ainsi que les deux courbes à recaler.

```
DEBUT();
# AFFECTATION DES VALEURS DES PARAMETRES A RECALER
# LES VALEURS RENSEIGNEES ICI SONT SANS IMPORTANCE
# SEULES COMPTENT LES VALEURS RENSEIGNEES DANS LE FICHIER MAITRE
DSDE__ = 200.;

YOUN__ = 8.E4;

SIGY__ = 1.;

ACIER=DEFI_MATERIAU(ECRO_LINE=_F(D_SIGM_EPSI=DSDE__,
                                 SY=SIGY__,),
                    ELAS=_F(NU=0.3,
                            E=YOUN__,),);
.....

U=SIMU_POINT_MAT(COMPORTEMENT=_F(RELATION='VMIS_ISOT_LINE'),
                 MATER=ACIER,
                 INCREMENT=_F(LIST_INST=INSTANTS,),
                 NEWTON=_F(REAC_ITER=1),
                 EPSI_IMPOSE=_F(EPYY=epyy,),
                 );

# EXTRACTION DE LA REPONSE SIGMAYY(T)
REPONSE1=CALC_TABLE(TABLE = U,
               ACTION =_F(OPERATION='EXTR',
                          NOM_PARA=('INST','SIYY')))

# EXTRACTION DE LA REPONSE EPSP(T)
REPONSE2=CALC_TABLE(TABLE = U,
               ACTION =_F(OPERATION='EXTR',
                          NOM_PARA=('INST','V1')))

FIN();
```

#### 6.1.2.3 MACR_RECAL

Il nous faut maintenant définir dans le fichier maître les valeurs initiales et les plages de variations de nos paramètres. On désire :

| | | | | | |
|---|---|---|---|---|---|
| 1.E5 | < | Module d'Young initial = 1.E5 | < | 5.E5 |
| 5. | < | Limite d'élasticité initiale = 30. | < | 500 |
| 1.E3 | < | Module d'écrouissage initial = 1.E3 | < | 1.E4 |

Enfin il faut aussi définir dans le fichier maître les grandeurs à extraire du fichier de commandes esclave. Nous désirons d'une part extraire la colonne INST et la colonne SIYY de la table REPONSE1 et d'autre part la colonne INST et la colonne V1 de la table REPONSE2. Nous l'écrivons :

Voici comment nous renseignons ces informations dans le corps de MACR_RECAL :



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

```
RESU2=MACR_RECAL(
       UNITE_ESCL = 3,
       PARA_OPTI=(  _F(NOM_PARA='YOUN__',  VALE_INI=100000.0,
                       VALE_MIN=50000.0,   VALE_MAX=500000.0),
                    _F(NOM_PARA='DSDE__',  VALE_INI=1000.,
                       VALE_MIN=500.,      VALE_MAX=10000.),
                    _F(NOM_PARA='SIGY__',  VALE_INI=30.,
                       VALE_MIN=5.,        VALE_MAX=500.),),
       COURBE=(  _F(FONC_EXP=exper1, NOM_FONC_CALC='REPONSE1',
                    PARA_X='INST',   PARA_Y='SIYY'),
                 _F(FONC_EXP=exper2, NOM_FONC_CALC='REPONSE2',
                    PARA_X='INST',   PARA_Y='V1'),),
       )
```

#### 6.1.2.4 Mise en données alternative (listes Python)

On peut utiliser la mise en données à base de liste Python et d'objet numpy. Le fichier maître va alors s'écrire :

```
experience=[   numpy.array([[0.00000E+00  , 0.00000E+00 ],
                            [5.00000E-02  , 5.00000E+01 ],
                            .........
                            [9.50000E-01  , 2.07500E+02 ],
                            [1.00000E+00  , 2.08000E+02 ]]),
               numpy.array([[0.00000E+00  , 0.00000E+00 ],
                            [5.00000E-02  , 0.00000E+00 ],
                            .........
                            [9.50000E-01  , 3.71250E-03 ],
                            [1.00000E+00  , 3.96000E-03 ]]) ]

parametres =[['YOUN__',100000.,50000.,500000.],['DSDE__',1000.,500.,10000.],
             ['SIGY__',30.,5.,500.]]

calcul = [['REPONSE1','INST','SIYY'],['REPONSE2','INST','V1']]
```

où : `experience` est le nom d'une liste Python (définie entre crochets) de 2 tableaux numpy, `parametres` est une liste de listes Python contenant les valeurs initiales et les plages de variations de nos paramètres et `calcul` les grandeurs à extraire du fichier de commandes esclave.

Le corps de la commande s'écrit alors :

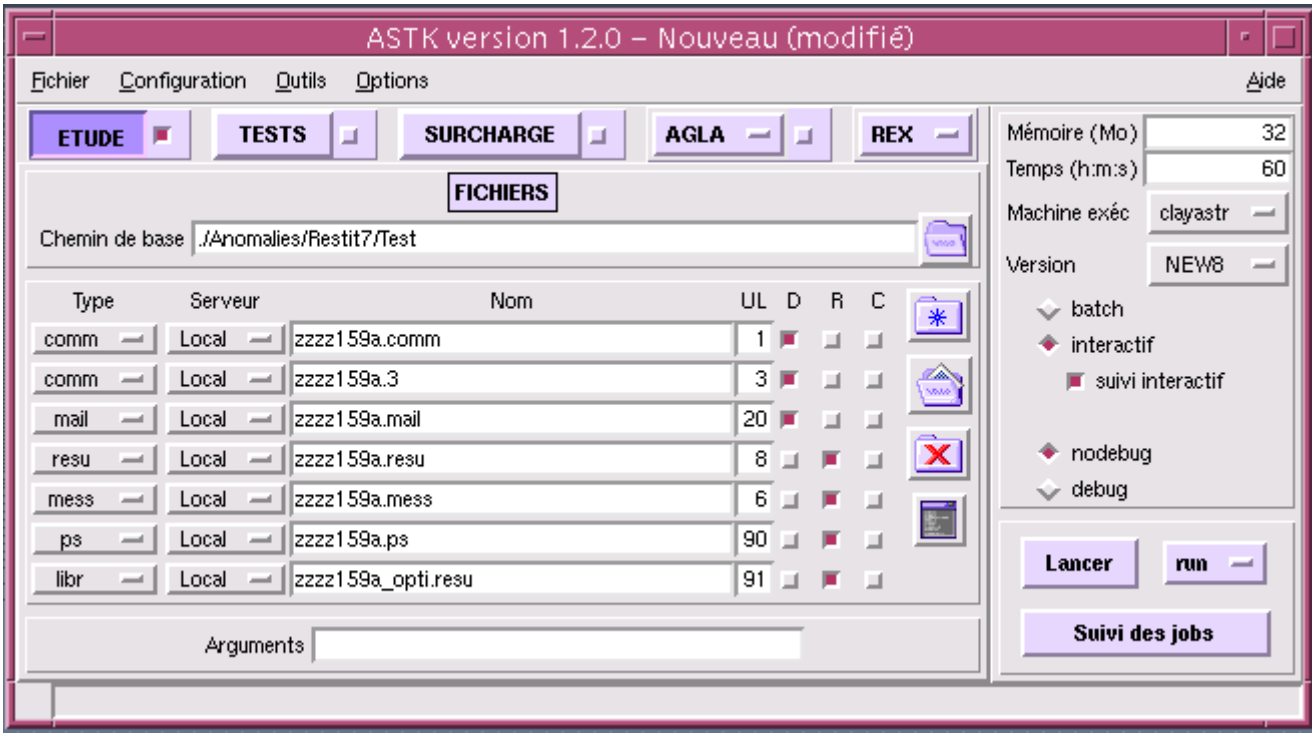
```
RESU=MACR_RECAL(
         UNITE_ESCL       =3,
         RESU_EXP         =experience,
         LIST_PARA        =parametres,
         RESU_CALC        =calcul,);
```

#### 6.1.2.5 Astk

On définit enfin le profil d'étude suivant :



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




### 6.1.3 Résultats

Une fois l'étude réalisée, le fichier de résultat du recalage ZZZZ159_opti.resu contient les informations suivantes :

```
Calcul de la sensibilité par rapport à = YOUN__ DSDE__ SIGY__
=========================================================
Iteration 0 =

=> Fonctionnelle = 1.0
=> Résidu        = 1.0
=> Paramètres    =
         YOUN__ = 100000.0
         DSDE__ = 1000.0
         SIGY__ = 30.0
=========================================================


Calcul de la sensibilité par rapport à = YOUN__ DSDE__ SIGY__
=========================================================
Iteration 1 =

=> Fonctionnelle = 0.259742161795
=> Résidu        = 0.30865397471
=> Paramètres    =
         YOUN__ = 300857.888503
         DSDE__ = 9135.12770111
         SIGY__ = 152.548047532
=========================================================


Calcul de la sensibilité par rapport à = YOUN__ DSDE__ SIGY__
=========================================================
```



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

```
Iteration 2 =

=> Fonctionnelle = 0.0757636994765
=> Résidu        = 0.473053125246
=> Paramètres    =
         YOUN__ = 157723.378846
         DSDE__ = 2022.7431335
         SIGY__ = 213.155325073
============================================================


Calcul de la sensibilité par rapport à = YOUN__ DSDE__ SIGY__
============================================================
Iteration 3 =

=> Fonctionnelle = 0.00190706595529
=> Résidu        = 0.0520849911718
=> Paramètres    =
         YOUN__ = 192302.166747
         DSDE__ = 895.845518907
         SIGY__ = 203.753909707
============================================================


Calcul de la sensibilité par rapport à = YOUN__ DSDE__ SIGY__
============================================================
Iteration 4 =

=> Fonctionnelle = 2.70165453323e-06
=> Résidu        = 0.00172172540305
=> Paramètres    =
         YOUN__ = 199801.572817
         DSDE__ = 1928.08902726
         SIGY__ = 200.274590793
============================================================

Calcul de la sensibilité par rapport à = YOUN__ DSDE__ SIGY__
============================================================
Iteration 5 =

=> Fonctionnelle = 2.65431115925e-12
=> Résidu        = 1.83121468206e-06
=> Paramètres    =
         YOUN__ = 199999.975047
         DSDE__ = 1999.86955101
         SIGY__ = 200.000462987
============================================================

============================================================
                   CONVERGENCE ATTEINTE
============================================================

Valeurs propres du Hessien:
[  7.17223479e+00   3.67264061e-01   6.25194340e-04]

Vecteurs propres associés:
[[ 0.98093218 -0.00549396 -0.19427266]
 [-0.19418112 -0.06940835 -0.97850712]
 [ 0.00810827 -0.9975732   0.0691517 ]]
```



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

--------

On peut en déduire que :

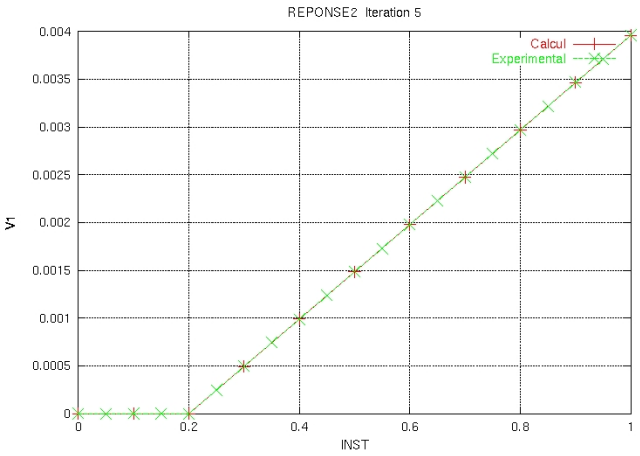
Les combinaisons suivantes de paramètres sont prépondérantes pour votre calcul :

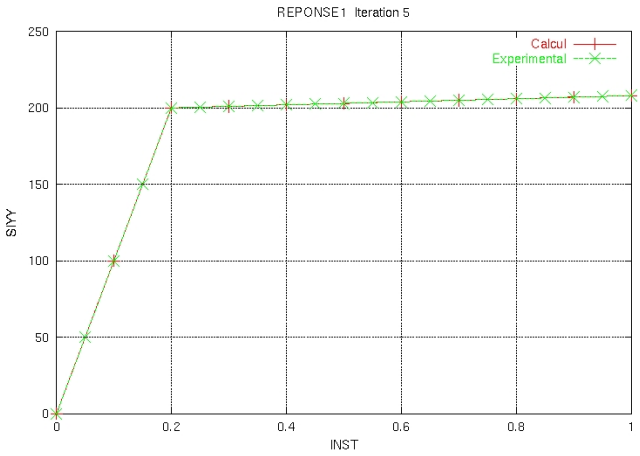
1) +9.8E-01 * YOUN__ -1.9E-01 * DSDE__
associée à la valeur propre 7.2E+00

Les combinaisons suivantes de paramètres sont insensibles pour votre calcul :

1) -1.9E-01 * YOUN__ -9.8E-01 * DSDE__
associée à la valeur propre 6.3E-04

Et le fichier POSTSCRIPT ZZZZ159.ps contient :






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

## 6.2 Identification des paramètres d'un modèle dynamique en exploitant la sensibilité des modes propres

Cet exemple est traité par le test SDLS121A [V2.03.121].
On souhaite recaler l'épaisseur d'une plaque et la valeur d'une masse discrète qui est située dessus en utilisant des mesures expérimentales de modes propres.

### 6.2.1 Mise en données

Dans le fichier maitre on renseigne d'abord les noms des tables qui contiendront à la fois les résultats calculés ainsi que les résultats numériques:

```
calcul = [    ['REPONSE1', 'NUME_ORDRE','FREQ'],
              ['REPONSE2', 'NUME_ORDRE','MAC']]

experience=[ ['REPEXP1',   'NUME_ORDRE','FREQ'],
              ['REPEXP2',   'NUME_ORDRE','MAC_EXP']]
```

Il est important de noter ici que, pour les résultats calculés, la REPONSES2 est donnée par la diagonale de la matrice des MAC issue de la commande MAC_MODES comme on verra un peu plus tard dans ce paragraphe. Le nom du paramètre 'MAC' est donc un nom réservé, choisi ainsi dans le Fortran, et donc il ne peut plus être utilisé par la suite (pour la réponse expérimentale correspondant on a choisi 'MAC_EXP')

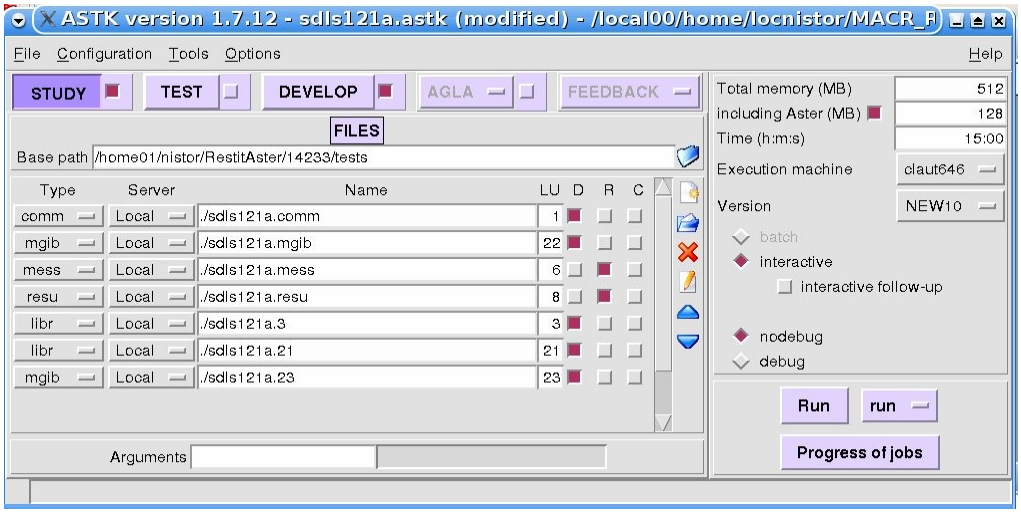
Les options retenues dans la commande MACR_RECAL, toujours dans le fichier maitre sont:

```
RESU=MACR_RECAL(
                    UNITE_ESCL       =3,
                    RESU_EXP         =experience,
                    LIST_PARA        =parametres,
                    RESU_CALC        =calcul,
                    POIDS            =poids,
                    METHODE          ='HYBRIDE',
                    ITER_FONC_MAXI   =500,
                    NB_PARENTS       =10,
                    NB_FILS          =5,
                    ECART_TYPE       =10.,
                    ITER_ALGO_GENE   =2,
                    DYNAMIQUE =_F(
                                MODE_EXP='MODMES',
                                MODE_CALC='MODNUM',
                                APPARIEMENT_MANUEL='NON',),) ;
```

On a choisi donc de lancer le recalage en utilisant la méthode HYBRIDE avec deux itérations pour l'algorithme évolutionnaire. Finalement on définit le profil suivant pour l'étude:



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




### 6.2.2 Calcul esclave

La principale particularité du calcul esclave est la présence dans le fichier de commande (unité 3) de deux modèles: le modèle expérimental et celui numérique. Ci-dessous on présente, pour ce cas-test, le fichier de commande esclave avec les explications nécessaires.

- lecture du maillage expérimental (le maillage des capteurs):

```
PRE_GIBI(UNITE_GIBI=23);

MAILEXP1=LIRE_MAILLAGE();
```

- création d'un groupe de nœuds qui servira plus tard à définir un éléments discret de masse:

```
MAILEXP1=DEFI_GROUP(   reuse=MAILEXP1,
                       MAILLAGE=MAILEXP1,
                       CREA_GROUP_NO=_F(
                       NOM='NO_MA',
                       OPTION='ENV_SPHERE',
                       POINT=(2.0,3.0),
                       RAYON=0.1,
                       PRECISION=0.1,
                    ),);
```

- création du groupe de nœuds pour définir les conditions aux limites:

```
MAILEXP1=DEFI_GROUP(   reuse =MAILEXP1,
                       MAILLAGE=MAILEXP1,
                       CREA_GROUP_NO=_F(   GROUP_MA='BORDS',
                                           NOM='BORDS',),);
```

- création d'un éléments POI1 pour introduire la masse discrète:

```
MAILEXP2= CREA_MAILLAGE(   MAILLAGE=MAILEXP1,
                           CREA_POI1 =(_F(  NOM_GROUP_MA = 'MASS',
                                            GROUP_NO = 'NO_MA'),),)
```

- affectation du modèle expérimental:

```
MODEXP=AFFE_MODELE( MAILLAGE=MAILEXP2,
                    AFFE=(_F( GROUP_MA = 'TOUT_ELT',
```



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

```
                                PHENOMENE='MECANIQUE',
                                MODELISATION='DST',),
                    _F(GROUP_MA = 'MASS',
                       PHENOMENE='MECANIQUE',
                       MODELISATION='DIS_T',),),);
```

- définition des caractéristiques élémentaires, matériau, etc.

```
CAREXP=AFFE_CARA_ELEM(...);
ACIER=DEFI_MATERIAU(...);
MATEX=AFFE_MATERIAU(...);
```

- calculs élémentaires et assemblages des matrices.

```
KELEXP=CALC_MATR_ELEM(OPTION='RIGI_MECA',
                ...);
MELEXP=CALC_MATR_ELEM(OPTION='MASS_MECA',
                ...);
NUMEXP=NUME_DDL(...);

KASSEXP=ASSE_MATRICE(...);
MASSEXP=ASSE_MATRICE(...);
```

- création de la sd_mode_meca avec les modes propres expérimentaux:

```
MODMES=LIRE_RESU(TYPE_RESU='MODE_MECA',
          FORMAT='IDEAS',
          MODELE=MODEXP,
          UNITE=21,
          NOM_CHAM='DEPL',
          MATR_RIGI =KASSEXP,
          MATR_MASS =MASSEXP,
          FORMAT_IDEAS=_F( NOM_CHAM='DEPL',
                           NUME_DATASET=55,
                           RECORD_6=(1,2,3,8,2,6,),
                           POSI_ORDRE=(7,4,),
                           POSI_NUME_MODE=(7,4),
                           POSI_FREQ=(8,1,),
                           POSI_MASS_GENE=(8,2),
                           POSI_AMOR_GENE=(8,3),
                           NOM_CMP=('DX','DY','DZ','DRX','DRY','DRZ'),),
          TOUT_ORDRE='OUI',);
```

Et on continue avec le modèle numérique, les mêmes étapes. Les paramétrés à recaler sont EP__ (l'épaisseur) et MP__ (la masse):

```
EP__=0.5
MP__=50000.

PRE_GIBI(UNITE_GIBI=22);

…
…
…

#nombre de fréquences
NF=8

MODES=CALC_MODES(MATR_RIGI=M_AS_RIG,
```



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

```
                    MATR_MASS=M_AS_MAS,
                    OPTION='PLUS_PETITE',
                           CALC_FREQ=_F(NMAX_FREQ=NF),
                           SOLVEUR_MODAL=_F(METHODE='SORENSEN'),
                           VERI_MODE=_F(STOP_ERREUR='NON'),
                          );
```

Le maillage expérimental est toujours plus grossier que le maillage du modèle numérique donc il faut projeter le résultat numérique sur le maillage expérimental:

```
MODNUM = PROJ_CHAMP(    RESULTAT=MODES,
                        MODELE_1=MODEL,
                        MODELE_2=MODEXP,
                        NUME_DDL=NUMEXP)
```

On récupère le tableau des fréquences expérimentales

```
REPEXP1=RECU_TABLE( CO=MODMES,
                    NOM_PARA='FREQ');
```

On construit le tableau des MAC expérimentaux - en fait le MAC idéal qui est 1.0

```
liste_mac=[]
for i in range(NF):
       liste_mac.append(1.0)

REPEXP2=CREA_TABLE(LISTE=( _F(PARA='NUME_ORDRE',LISTE_I=range(1,NF+1),),
                           _F(PARA='MAC_EXP',LISTE_R=liste_mac,), ),);
```

Et finalement les tableaux avec les réponses calculées:

```
REPONSE1=RECU_TABLE(    CO=MODES,
                        NOM_PARA='FREQ');

REPONSE2=MAC_MODES( BASE_1=MODNUM,
                    BASE_2=MODMES,);
```

Les résultats (les valeurs recalées des paramètres) sont ensuite calculés de façon similaire avec le cas classique de recalage présenté dans le paragraphe précédant.



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

## 6.3 Identification paramétrique de système conservatif en exploitant l'orthogonalité des modes propres

Cet exemple est traité par la modélisation D du cas test SDLS121 [V2.03.121].
On souhaite recaler l'épaisseur d'une plaque et la valeur d'une masse discrète qui est située dessus en utilisant des mesures expérimentales de modes propres.

A partir des déformées modales relevées aux points d'observation, o n effectue une expansion sur le modèle numérique support. On essaie par la suite de retrouver les paramètres du modèle tels que les matrices de masse et de raideur généralisées relatives aux modes identifiés expérimentalement soient diagonales et que l'écart entre la pulsation propre mesurée et la pulsation propre calculée soit minimal.

### 6.3.1 Mise en données

La technique, illustrée ici, exploite les déformées modales identifiées étendues sur le modèle numérique support. Cette déformée modale peut être obtenue à l'aide d'un calcul indépendant en déposant les déformées étendues dans un fichier qui sera lu par la suite par le fichier esclave. Pour de raisons pratiques, on effectue l'expansion dans le fichier pilote. Il est plus facile ainsi de récupérer le modèle support qui sera utilisé dans le fichier esclave.

Dans le fichier maître on renseigne à la fois les valeurs cibles (valeurs expérimentales) et les noms des tables qui contiennent les résultats issus du calcul numérique effectué dans le fichier esclave.

Les différentes étapes sont rappelées ci-après.

- Création d'un modèle expérimental afin de pouvoir définir la localisation des points de mesure. Les nœuds du maillage peuvent être reliés entre eux par des éléments discrets pour faciliter la visualisation des informations mesurées. Les caractéristiques mécaniques que l'on affecte à ce modèle n'influent pas sur les résultats, elles permettent au modèle d'accueillir les modes propres identifiés
- Lecture des modes propres identifiés
- Création du modèle numérique support pour l'expansion de la mesure
- Définition de la base d'expansion (ici, une base modale)
- Calcul des coordonnées des modes identifiés dans la base d'expansion
- Expansion des informations mesurées sur le modèle support
- Ecriture des modes étendus dans un fichier afin de pouvoir l'exploiter dans le calcul esclave
- Définition des valeurs cibles

Les valeurs cibles sont déposées dans deux listes. La première liste contient les termes extra-diagonnaux qui devraient être nuls et la deuxième liste contient les fréquences propres identifiées.

- Définition des paramètres de recalage
- Lancement de la procédure d'optimisation

### 6.3.2 Calcul esclave

Le calcul esclave crée les tables qui contiendront les valeurs calculées à partir des paramètres d'entrée. Les différentes étapes de calcul sont décrites ci-après :



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

- Création du modèle support (même numérotation de ddl que celle qui a servi à l'expansion de la mesure)
- Lecture des déformées modales étendues
- Calcul des matrices de masse et de raideur generalisees via MAC_MODES
- Création des tables qui contiennent les termes extradiagonnaux des matrices généralisées
- Création de la table qui contient les fréquences propres estimées à l'aide des matrices généralisées

## 6.4 Identification paramétrique de système dissipatif en exploitant les relations de norme des déformées modales

Cet exemple est traité par la modélisation E du cas test SDLS121 [V2.03.121].

On souhaite recaler l'épaisseur d'une plaque, la valeur d'une masse ponctuelle sur la plaque et les amortissements discrets répartis sur la plaque.

A partir des déformées modales complexes relevées aux points d'observation, o n effectue une expansion sur le modèle numérique support. On essaie par la suite de retrouver les paramètres du modèle de telle sorte que les modes propres identifiés expérimentalement vérifient les relations de norme associées au modèle numérique. On se focalise uniquement sur les termes diagonaux de la matrice de normalisation.

### 6.4.1 Mise en données

La mise en données ressemble à celle présentée dans le chapitre précédent (6.3). La différence se situe au niveau de l'expansion de la déformée modale. Les déformées modales sont ici complexes. On effectue l'expansion en traitant la partie réelle et la partie imaginaire de façon indépendante. La déformée complexe étendue est obtenue après assemblage de la déformée réelle et de la déformée imaginaire étendues.

Les différentes étapes sont décrites ci-après :

- Création du modèle expérimental
- Lecture des modes propres identifiés (expérimentaux)
- Expansion de la mesure
- Définition des valeurs cibles (les valeurs cibles sont ici des vecteurs nuls)
- Définition des paramètres de recalage
- Lancement de la procédure d'optimisation

### 6.4.2 Calcul esclave

Le calcul esclave crée les tables qui contiendront les termes diagonaux de la matrice de normalisation.

Les différentes étapes de calcul sont décrites ci-après :



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

- Création du modèle support (même numérotation de ddl que celle qui a servi à l'expansion de la mesure)
- Lecture des déformées modales étendues
- Lecture des fréquences propres et amortissements réduits
- Calcul des produits matrice * vecteur par PROD_MATR_CHAM
- création de la table qui contient les termes relatifs à l'amortissement et à la masse : $\frac{y_\nu^T B \bar{y}_\nu + 2\Re(s_\nu) y_\nu^T M \bar{y}_\nu}{|n_\nu|} = z_{1\nu}$
- création de la table qui contient les termes relatifs à la rigidité et à la masse : $\frac{s_\nu \bar{s}_\nu y_\nu^T M \bar{y}_\nu - y_\nu^T K \bar{y}_\nu}{|n_\nu s_\nu|} = z_{2\nu}$
- création de la table qui contient les termes relatifs à la masse, la rigidité et l'amortissement : $\frac{s_\nu^2 y_\nu^T M y_\nu + s_\nu y_\nu^T B y_\nu + y_\nu^T K y_\nu}{n_\nu s_\nu} = z_{3\nu}$

# 7 Utilisation du mode EXTERNE

## 7.1 Avertissement

Nous attirons l'attention sur le fait que ce mode de fonctionnement est à réserver à un usage avancé. La plupart des cas devraient être traités avec les algorithmes fournis dans la commande MACR_RECAL, et notamment l'algorithme Levenberg-Marquardt, qui est le plus adapté aux problèmes de recalage de paramètres.

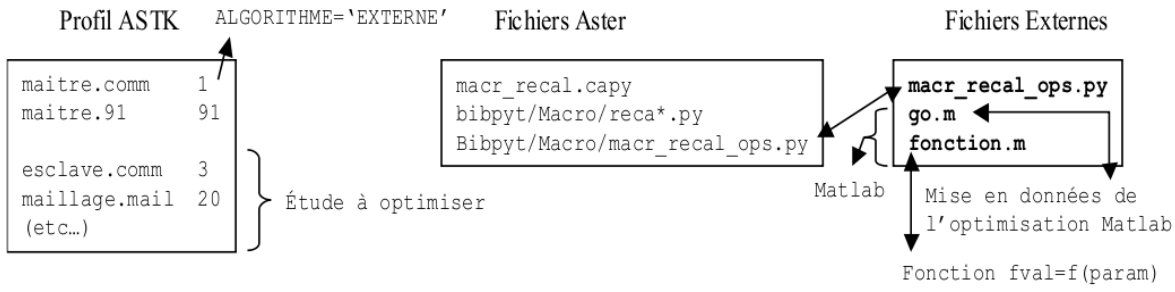
Ce mode de fonctionnement nécessite un logiciel externe à *Code_Aster* pour effectuer l'optimisation (code Python, Matlab, Scilab, logiciel type boite noire, etc.). De plus, il est nécessaire d'avoir quelques compétences Python.

**Enfin, l'utilisation du mode EXTERNE sort du périmètre AQ d'Aster**, et ne devrait pas être utilisé pour des études IPS.

## 7.2 Méthodologie

### 7.2.1 Principe

Dans ce mode d'utilisation, *Code_Aster* est uniquement utilisé pour l'évaluation de la fonctionnelle pour un logiciel d'optimisation qui est complètement externe à *Code_Aster*.






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

**Figure 7.2.1-a: MACR_RECAL « Algorithme externe »**

Généralement, les logiciels d'optimisation demandent que l'utilisateur écrive une procédure pour le calcul de la fonctionnelle : f = F(param).

Si le logiciel autorise la lecture/écriture de fichiers et l'exécution de code externe (commande « system » ou autres), alors il est potentiellement utilisable avec MACR_RECAL. On va encapsuler l'appel à la procédure Python **recal.py (anciennement MACR_RECAL_ops.py)** ainsi que l'écriture du fichier des paramètres et la relecture du fichier de la valeur de la fonctionnelle dans la routine de calcul de F.

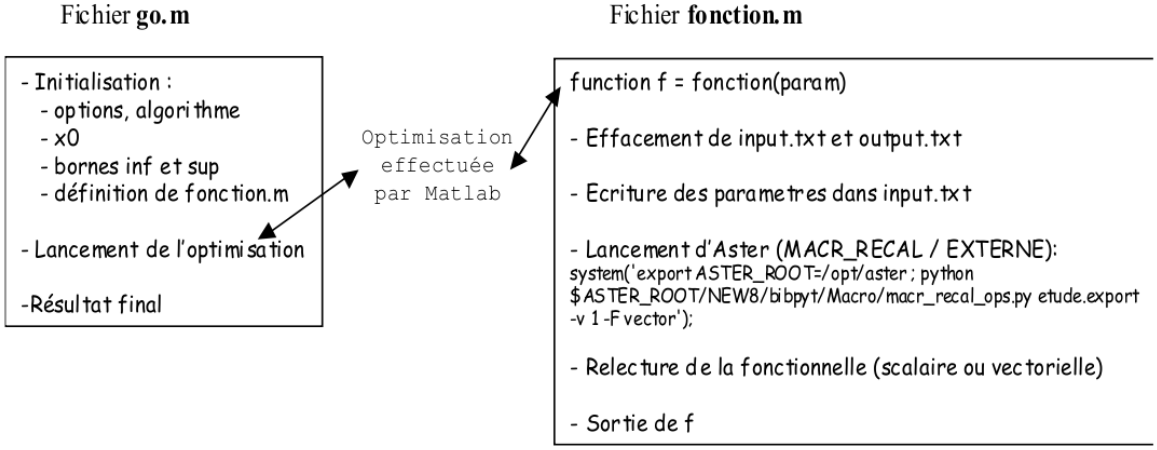



**Figure 7.2.1-b: MACR_RECAL « Principe du mode EXTERNE », exemple Matlab**

Le logiciel d'optimisation lance *Code_Aster* à chaque évaluation de la fonctionnelle. La routine d'évaluation de la fonctionnelle passe par des fichiers textes pour envoyer à *Code_Aster* les paramètres à utiliser et pour récupérer la valeur de la fonctionnelle. La routine Python recal.py est donc appelable indépendamment d'Aster et fait le lien entre le logiciel d'optimisation et *Code_Aster* (récupération du fichier des paramètres, remise en forme du fichier esclave, lancement d'Aster, récupération de la fonctionnelle).

En pratique, il faut disposer d'un profil Astk classique de l'étude esclave (et non pas le profil de l l'une utilisation de MACR_RECAL) . Ce profil doit être complètement fonctionnel.

Sous ce mode EXTERNE, le mode de fonctionnements est le suivant : la routine recal.py se charge de récupérer les paramètres depuis la ligne de commande ou depuis un fichier input.txt, va aller remplacer ces paramètres dans le profil esclave, puis va lancer l'exécution de l'étude esclave pour les nouveau paramètres, et enfin va renvoyer dans une fichier texte output.txt les valeurs de la fonctionnelle.

A ce jour, ce mode de fonctionnement a été testé avec plusieurs logiciels d'optimisation (la *toolbox* d'optimisation de Matlab, la *toolbox* Tomlab pour Matlab, et le module d'optimisation de Python-Scipy) mais il est possible d'utiliser à peu près n'importe quel logiciel à partir du moment où ce logiciel autorise l'exécution externe d'un script. On pense notamment à Zopt (Zebulon), SiDoLo.

## 7.2.2 Utilisation du fichier de lancement externe recal.py

Le fichier recal.py peut s'exécuter de manière autonome avec quelques paramètres optionnels sur la ligne de commande :

```
Usage: /aster/NEW10/bibpyt/Macro/recal.py fichier_export [options]
```



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

```
Options:
  -h, --help            show this help message and exit
  --input=INPUT         Chaine de texte contenant les parametres
  --input_step=INPUT_STEP
                        Chaine de texte contenant les pas de discretisation
                        des differences finies
  --input_file=INPUT_FILE
                        Fichier contenant les parametres
  --input_step_file=INPUT_STEP_FILE
                        Fichier contenant les pas de discretisation des
                        differences finies
  --output=OUTPUT       fichier contenant la fonctionnelle
  --output_grad=OUTPUT_GRAD
                        fichier contenant le gradient
  --aster_root=ASTER_ROOT
                        Chemin d'installation d'Aster
  --as_run=as_run       Chemin vers as_run
  --resudir=RESUDIR     Chemin par defaut des executions temporaires d'Aster
  --noclean             Erase temporary Code_Aster execution directory
  --info=INFO           niveau de message (0, [1], 2)
  --sources_root=SOURCES_ROOT
                        Chemin par defaut des surcharges Python
  --objective=OBJECTIVE
                        Fonctionnelle ([fcalc]/[error])
  --objective_type=OBJECTIVE_TYPE
                        type de la fonctionnelle (float/[vector])
  --gradient_type=GRADIENT_TYPE
                        calcul du gradient par Code_Aster ([no]/normal/adim)
  --mr_parameters=MR_PARAMETERS
                        Fichier de parametres de MACR_RECAL : parametres,
                        calcul, experience
  --study_parameters=STUDY_PARAMETERS
                        Fichier de parametre de l'etude : export
  --parameters=PARAMETERS
                        Fichier de parametres
```

Cette procédure a besoin :

1. du fichier .export de l'étude `Astk` définie précédemment (l'étude standard de `MACR_RECAL`)
2. d'un fichier texte contenant la liste des paramètres

A partir de ces deux fichiers, elle lance un calcul *Code_Aster* pour les paramètres spécifiés (ou deux calculs *Code_Aster*, voir plus haut) et génère un fichier texte contenant uniquement la valeur de la fonctionnelle (scalaire ou vectorielle).

Cette procédure peut être lancée sans argument. Elle va alors chercher **dans le répertoire courant** des fichiers par défaut :

1. s'il n'y a qu'un seul fichier .export dans le répertoire courant, celui-ci sera utilisé (dans le cas contraire la procédure s'arrête en erreur)
2. le fichier d'entrée par défaut sera cherché sous le nom « input.txt »
3. le fichier de sortie sera généré avec le nom « output.txt »

Les arguments –i (--input) et -o (--output) permettent de préciser les fichiers.

L'argument –g (--output_grad) : permet de préciser le fichier texte dans lequel sera écrit le gradient, si celui-ci est demandé (argument -G).

D'autres arguments sont disponibles :

1. --info : défini le niveau de message ; 0=muet, 1 ou 2 ;
2. --objective : permet de préciser si la fonctionnelle doit être retournée sous la forme vectorielle ou scalaire ;



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

3. --gradient : permet de préciser si on ne veut pas qu'Aster calcule et renvoie les gradient (no), et si on veut les gradients, de préciser si on les veut adimensionné ou non ;

Par défaut, un fichier *Code_Aster*`.output` est crée dans le répertoire d'exécution de la procédure, permettant d'avoir accès à l'output du dernier calcul *Code_Aster*.

Enfin, il faut configurer la variable d'environnement `ASTER_ROOT` (le chemin de base d'Aster) pour pouvoir lancer en `Python` le fichier `recal.py` ou aors utiliser l'argument `-aster_root`.

Le bon fonctionnement de la procédure `recal.py` peut être vérifié manuellement, en générant un fichier d'entrée avec des valeurs de paramètres et en lançant manuellement la procédure. Exemple sur la machine *Code_Aster* :

```
cd repertoire
export ASTER_ROOT=/aster
echo "10., 20., 30. " >input.txt
Python $ASTER_ROOT/NEW10/bibpyt/Macro/recal.py etude.export --info=2
cat ouput.txt
```

Le calcul *Code_Aster* devrait se lancer et terminer convenablement. Un fichier output.txt doit être généré dans le répertoire courant.

Le cas-test `zzzz159f` illustre l'utilisation externe de `MACR_RECAL` : dans ce test *Code_Aster*, on définit une fonction f et on rappelle une fois *Code_Aster* pour simuler une itération de recalage.

## 7.3 Exemple : module d'optimisation de Matlab©

Si on reprend le principe et qu'on l'applique à un algorithme d'optimisation qui serait écrit sous `Matlab`, on doit écrire deux fichiers `go.m` et `fonction.m` qui effectuent les actions décrites ci-dessous :

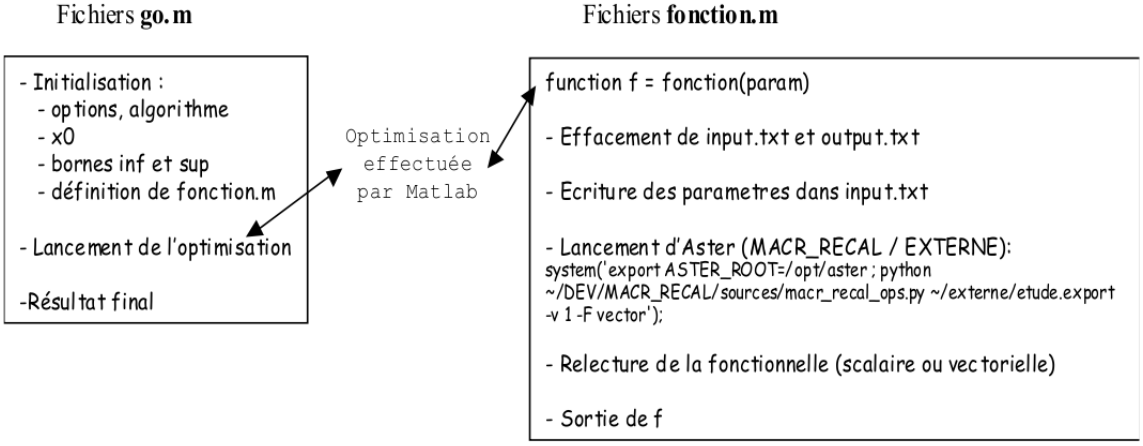



**Figure 7.3-a: MACR_RECAL « Principe du mode EXTERNE » appliqué à Matlab©**

L'exemple suivant utilise le module d'optimisation de `Matlab` et effectue les calculs *Code_Aster* sur le serveur local.

**Fichier go.m** (initialisation et lancement de l'optimisation)

```
clear all;
format long;
```



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

```
system('rm -f fort.91');

options   =   optimset('Display',   'iter',   'LevenbergMarquardt',   'on',
'TolFun', 1e-8, 'MaxFunEvals', 1000, 'MaxIter', 100);

% Params :  DSDE__, SIGY__,   YOUN__
x0 = [  1000.,    30.,  100000. ];
lb = [   500.,     5.,   50000. ];
ub = [ 10000.,   500.,  500000. ];

% vecteur
[x,resnorm,residual,exitflag,output,lambda,jacobian]                    =
lsqnonlin(@fonction,x0,lb,ub,options)

% scalaire
%[x,fval,exitflag,output]     =     fmincon(@fonction,x0,[],[],[],[],lb,ub,
[],options)
```

**Fichier fonction.m** (calcul de F(param)) / version Linux

```
function f = fonction(x)

system('rm -f input.txt');
system('rm -f output.txt');

% Ecriture des parametres
dlmwrite('input.txt', x, 'precision', '%.20f');

% Local
iret = system('export ASTER_ROOT=/opt/aster ; Python $ASTER_ROOT/NEW10/bibpyt/Macro/recal.py
etude.export --objective_type=vector');

f = dlmread('output.txt', ',', 0, 0);   % relit un scalaire ou un vecteur
```

Dans `go.m`, on donne deux exemples d'algorithmes. L'algorithme `lsqnonlin` minimise une fonctionnelle vectorielle et il faut donc décommenter la première ligne « iret= ». Cet algorithme est basé sur Levenberg-Marquardt à contraintes actives, et est donc similaire à celui implanté dans *Code_Aster*.

**Remarque :**

*Il est important de noter la présence de la commande `format long` et de l'argument `precision` pour la commande `dlmwrite` . Ceci permet de travailler avec un nombre suffisant de chiffres significatifs et de ne pas s'arrêter si la tolérance sur la fonction n'est plus respectée.*

**Remarque 2 :**

*Lorsque le logiciel d'optimisation n'est pas capable de gérer les codes retour de l'exécution externe, il faut faire attention à arrêter proprement la procédure en cas d'arrêt anormal du calcul Code_Aster. C'est la raison pour laquelle on efface les fichiers `input.txt` et `output.txt` juste après leur utilisation, afin de ne pas tomber dans des boucles sans fin.*



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