**Manuel d'Utilisation**
**Fascicule U2.08 : Fonction avancée et contrôle des calculs**
**Document : U2.08.05**

# Simulation numérique de Monte Carlo

**Résumé :**

Ce document donne les éléments pour la mise en oeuvre de simulations numériques de Monte Carlo à partir du fichier de commande et d'opérateurs de génération aléatoire. Les trois principaux ingrédients sont :

- Une boucle Python,
- Un générateur de variables aléatoires (GENE_VARI_ALEA) et/ou un générateur de matrices aléatoires (GENE_MATR_ALEA) pour la dynamique, et/ou un générateur de fonctions aléatoires (GENE_FONC_ALEA),
- Le calcul des estimateurs statistiques (CALC_FONCTION).

# 1 Généralités

La méthode numérique de Monte Carlo permet de calculer différentes grandeurs statistiques d'une variable aléatoire ou d'un processus stochastique. Dans le contexte d'un calcul mécanique (ou thermo-mécanique, …) , le principe est d'obtenir $ns$ réalisations de cette variable aléatoire ou de ce processus stochastique puis d'en déduire les estimations statistiques recherchées. Les trois principales étapes de la méthode de Monte Carlo sont :

1) Génération d'un échantillon de $ns$ réalisations des données aléatoires d'entrée du modèle mécanique,
2) Calcul des $ns$ grandeurs résultats correspondant à ces données,
3) Calcul des estimateurs statistiques des grandeurs recherchées.

Dans la méthode de Monte Carlo simple ou directe que l'on utilise, chacun des $ns$ calculs peut être fait indépendamment des autres. Afin de réduire la taille mémoire nécessaire, les $ns$ générations et calculs sont donc effectuées séquentiellement dans une boucle avec destructions des résultats intermédiaires inutiles.

# 2 Boucle Python dans le fichier de commande Aster

Afin de permettre l'utilisation d'une boucle python dans le fichier de commande, il faut tout d'abord positionner le mot clé PAR_LOT de la commande DEBUT sur la valeur 'NON' :

```
DEBUT(CODE=_F(  NOM = 'SDNS001') ,PAR_LOT='NON')
```

La boucle python en elle même commence par la commande `for`, et englobe toute les lignes de même indentation

```
for k in range(1,1000) :
    COMMANDE1
    for m in range(1,500):
                COMMANDE2
    COMMANDE3
```

Dans cet exemple, on trouve deux boucles python emboîtées. La première, sur la variable $k$ , permet d'exécuter 999 fois les instructions COMMANDE1, la deuxième boucle python, et COMMANDE3. La deuxième boucle python interne, sur la variable $m$ permet à COMMANDE2 d'être exécuté 499 fois pour chaque $k$ allant de 1 à 999.

**Remarque**

*Il n'y a pas d'instruction de fin de boucle. Les indentations seules marquent le corps de l'instruction « for ».*

# 3 Génération de variables aléatoires

La terminologie de générateurs aléatoire doit être ici prise au sens large. Ces variables aléatoires peuvent être à valeurs scalaires, matricielles, ou même fonctionnelles (processus stochastique). Le *Code_Aster* est capable de générer de telles variables aléatoires respectivement par les commandes GENE_VARI_ALEA, GENE_MATR_ALEA et GENE_FONC_ALEA.

Les variables aléatoires peuvent être des paramètres du modèle éléments finis (paramètres matériaux, valeurs d'un jeu, d'une raideur de butée élastiques, d'un module d'Young, etc). Dans ce cas on modélise les incertitudes de modélisation par une approche probabiliste paramétrique et on utilise alors GENE_VARI_ALEA.

En dynamique des structures, ces variables aléatoires peuvent aussi être les matrices généralisées de masse, de raideur et d'amortissement et/ou les paramètres locaux du modèle aux éléments finis. Dans ce cas, on modélise à la fois les incertitudes de modèle et de modélisation par une approche probabiliste non-paramétrique, et on utilise GENE_MATR_ALEA.

Ces variables aléatoires à valeurs scalaires ou matricielles suivent des lois de probabilités construites par l'utilisation du principe du maximum d'entropie et de l'information disponible (voir [R4.03.05]).

La variable aléatoire peut encore être une fonction. L'opérateur GENE_FONC_ALEA permet de générer des trajectoires d'un processus stochastique multi-varié mono-dimensionnel (i.e. à plusieurs composantes et indexé sur une seule variable) stationnaire de moyenne nulle à partir de sa densité spectrale de puissance. Dans le cas d'un calcul dynamique transitoire, on peut ainsi générer des chargements temporels connues par leur matrice interspectrale.

A moins d'une indication contraire à l'aide du mot clé INIT_ALEA, toutes les valeurs générés par les trois commandes GENE_VARI_ALEA, GENE_MATR_ALEA et GENE_FONC_ALEA sont statistiquement indépendantes entre elles à l'intérieur d'une même exécution de *Code_Aster*. *A contrario*, d'une exécution à l'autre, un fichier de commande strictement identique (même appels aux trois commandes dans le même ordre avec les mêmes arguments) fournira exactement les mêmes résultats. Si l'on souhaite générer des résultats statistiquement indépendants d'une exécution à l'autre, alors il faut utiliser le mot-clé INIT_ALEA avec des valeurs majorant le nombre de termes utilisés dans les exécutions antérieures.

**Attention :**

*Le générateur de variable aléatoire utilisé est celui du module "random" de Python. Il dépend de la version de Python exploité par Code_Aster. Des résultats non convergés statistiquement peuvent donc varier d'une version à l'autre de Code_Aster ou d'une plate-forme à l'autre, si la version de Python n'est pas la même et qu'entre les deux versions le module random a évolué (cas entre Python 2.1 et 2.3).*

**Remarque :**

*En version Python 2.3, la période du générateur est de* 2**19937-1 *(M. Matsumoto and T. Nishimura, Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30 1998.)*

**Remarque :**

*Le module "random" de Python fournit une alternative à la commande* GENE_VARI_ALEA *pour générer des variables aléatoires dont les densités ne sont pas disponibles dans cette commande.*

# 4 Estimateurs statistiques

D'un échantillon de $ns$ réalisations de la quantité d'intérêt, on peut en déduire les estimations de grandeurs statistiques comme la moyenne, l'écart type,… Les estimateurs doivent en général être calculés en deux temps. Dans un premier temps, à l'intérieur de la boucle des quantité intermédiaires sont calculées puis dans un second après la boucle les estimateurs eux-même sont calculés.

Prenons par exemple un échantillon de spectres d'oscillateurs $\{SRO(\omega,\xi\ ;\ \theta_p)\}1 \le p \le ns$, pour lequel pour chaque pulsation nous souhaitons calculer les moments d'ordre 1 et 2. Ces moments ont pour expression :

$$m_1(\omega,\xi) = \frac{1}{ns}\sum_{p=1}^{ns} SRO(\omega,\xi,\theta_p)\,,$$

$$m_2(\omega,\xi) = \frac{1}{ns}\sum_{p=1}^{ns} SRO(\omega,\xi,\theta_p)^2\ .$$

Les deux sommes ci-dessus sont aisément calculables à l'intérieur de la boucle, il est simplement nécessaire de différencier le cas de l'initialisation de la somme et les cas d'incrémentation de cette somme.

Voici, par exemple, un fichier de commandes épuré permettant d'évaluer $m_2(\omega,\xi)$ :

génération d'une réalisation aléatoire du spectre d'oscillateur

évaluation de la somme $\sum SRO(\omega,\xi,\theta_p)$

```
for k in range(1,ns+1) :
    MATM=GENE_MATR_ALEA(MATR_MOYEN=MASSE,DELTA=0.2)
    MATK=GENE_MATR_ALEA(MATR_MOYEN=RIGID,DELTA=0.2)
    MATD=GENE_MATR_ALEA(MATR_MOYEN=AMORT,DELTA=0.2)
    DYNA =DYNA_TRAN_MODAL( MASS_GENE=MATM,
                           RIGI_GENE=MATK,
                           AMOR_GENE=MATD,
                           ... )
    ACC1=RECU_FONCTION(RESU_GENE = DYNA, ... )
    SRO= CALC_FONCTION(SPEC_OSCI=_F(NATURE='ACCE',FONCTION=ACC1,...))
    if k==1:
       M2_3= CALC_FONCTION(PUISSANCE=_F(FONCTION=SRO, EXPOSANT=2),)
    else:
       M2_0 = CALC_FONCTION(PUISSANCE=_F(FONCTION=SRO, EXPOSANT=2),)
       M2_1 = CALC_FONCTION( COMB=(_F(FONCTION=M2_0,COEF=1.),
                                   _F(FONCTION=M2_3,COEF=1.)) )
       DETRUIRE(CONCEPT=_F(NOM=(M2_3,M2_0)))
       M2_3=  CALC_FONCTION(COMB=_F(FONCTION=M2_2, COEF=1.), )
       DETRUIRE(CONCEPT=_F(NOM=(M2_2)))
    DETRUIRE CONCEPT=_F(NOM=(MATM,MATK,MATD,DYNA,SRO,ACC1)))
M2=  CALC_FONCTION(COMB=_F( FONCTION=M2_3, COEF=1./ns),)
```

Lorsque `k==1`, on initialise la fonction `M2_3` avec le carré de la première réalisation produite. La puissance énième d'une fonction est effectuée par le mot clé `PUISSANCE` de la commande `CALC_FONCTION`. La somme $\sum SRO(\omega,\xi,\theta_p)^2$ est stockée, au fur est à mesure que les réalisations sont produites, dans `M2_3` à l'aide des mots clés `PUISSANCE` et `COMB` de la commande `CALC_FONCTION`, des fonctions intermédiaires `M2_0` et `M2_1 et` de la commande `DETRUIRE`. Tous les différents concepts produits (`MATM`, `MATK`, `MATD`, `DYNA`, `SPO`, `A CC1`, `etc.`) doivent être détruits à la fin de chaque itération à l'exception de `M2_3`, bien entendu. Enfin, une fois les $ns$ itérations effectuées, la fonction $m_2(\omega,\xi)$ est évaluée et correspond à l'objet `M2` produit à la fin de l'exemple.

**Remarque :**

*Lorsque l'on utilise la commande* GENE_FONC_ALEA, *il existe la possibilité de ne pas utiliser de boucle python. Le principe est alors de générer "bout à bout" plusieurs temporels (mot-clé* NB_TIRAGE *dans* GENE_FONC_ALEA*) et de post-traiter les résultats avec la commande* CALC_INTE_SPEC. *Le cas-test* ZZZZ180 [V1.01.180] *donne une exemple d'une telle utilisation.*

# 5 Exemple en dynamique transitoire

**Principe du calcul déterministe**
On s'appuie sur le cas-test SDNS01a concernant la réponse d'une plaque rectangulaire avec une butée élastique soumise à une charge impulsionnelle déterministe.
On construit la solution du modèle dynamique réduit moyen (déterministe) à l'aide d'un enchaînement classique d'opérateurs ( ASSE_MATRICE, MODE_ITER_SIMULT, MACRO_PROJ_BASE ….)
On s'intéresse à la réponse du système calculée par DYNA_TRAN_MODAL, et plus exactement aux spectres normalisés des réponses et aux observations temporelles (champs de déplacement, vitesse, accélération, contraintes, etc).

**Principe du calcul probabiliste**
Les raideurs de butées sont rendues aléatoires ainsi que les matrices généralisées de masse, de raideur et d'amortissement.
Les réalisations de la réponse transitoire stochastique correspondante sont calculées par la méthode de simulation numérique de Monte Carlo directe avec $ns$ simulations à l'aide d'une boucle Python dont la structure est :

**Début boucle, pour $p = 1, \ldots, ns$ :**

a) Génération des $p^{ième}$ réalisations des variables aléatoires (approche paramétrique) à l'aide de GENE_VARI_ALEA.
Génération des $p^{ième}$ réalisations des matrices généralisées aléatoires de masse, de raideur et d'amortissement à l'aide de GENE_MATR_ALEA (approche non paramétrique). **Ces matrices ne sont pas diagonales et nécessitent donc un stockage plein**.

b) Calcul de la $p^{ème}$ réalisation $\mathbf{Q}^n(t;p)$ solution du système matriciel stochastique avec non-linéarités de chocs, l'entier $n$ étant la dimension du modèle réduit. Cette réalisation est la solution du système matriciel classique dont les matrices sont les réalisations précédemment générées. Le calcul est donc effectué par DYNA_TRAN_MODAL.

c) Extraction des ddls physiques en déplacement $\ddot{Z}_i^n(t;p)$ pour des ddls prédéfinis, *via* l'opérateur RECU_FONCTION.
Construction des spectres normalisés $\omega \mapsto \mathrm{S}_j(\xi,\omega;p)$ des réponses des ddls $\ddot{Z}_i^n(t;p)$ par l'opérateur CALC_FONCTION(SPEC_OSCI).

d) Evaluation à l'aide de l'opérateur CALC_FONCTION des contributions aux estimateurs des moyennes $\hat{\mathrm{m}}_{1j}(\xi,\omega;p)$, des moments d'ordre deux $\hat{\mathrm{m}}_{2j}(\xi,\omega;p)$, des valeurs extrêmes max. $\hat{\mathrm{S}}_{j,\max}(\xi,\omega;p)$ et min. $\hat{\mathrm{S}}_{j,\min}(\xi,\omega;p)$ d'échantillon pour les spectres normalisés :

$$\hat{\mathrm{m}}_{1j}(\xi,\omega;p) = \mathrm{S}_j(\xi,\omega;p) + \hat{\mathrm{m}}_{1j}(\xi,\omega;p-1),$$
$$\hat{\mathrm{m}}_{2j}(\xi,\omega;p) = \mathrm{S}_j(\xi,\omega;p)^2 + \hat{\mathrm{m}}_{2j}(\xi,\omega;p-1),$$
$$\hat{\mathrm{S}}_{j,\max}(\xi,\omega;p) = \mathrm{Max}\left\{\mathrm{S}_j(\xi,\omega;p), \hat{\mathrm{S}}_{j,\max}(\xi,\omega;p-1)\right\},$$
$$\hat{\mathrm{S}}_{j,\min}(\xi,\omega;p) = \mathrm{Min}\left\{\mathrm{S}_j(\xi,\omega;p), \hat{\mathrm{S}}_{j,\min}(\xi,\omega;p-1)\right\}.$$

**Fin de boucle.**

A la suite de la boucle, les moyennes, les écart-types, les valeurs extrêmes max. et min. d'échantillon pour les spectres normalisés peuvent être évalués :

$$\mathrm{m}_{1j}(\xi,\omega) = \frac{1}{n_s}\hat{\mathrm{m}}_{1j}(\xi,\omega;n_s), \qquad \mathrm{m}_{2j}(\xi,\omega) = \frac{1}{n_s}\hat{\mathrm{m}}_{2j}(\xi,\omega;n_s).$$
$$\sigma_j(\xi,\omega) = \sqrt{m_{2j}(\xi,\omega) - m_{1j}(\xi,\omega)^2},$$

**Fichier de commandes épuré correspondant :**

```
# SDNS01A : MODELE PROBABILISTE NONPARAMETRIQUE D UNE PLAQUE AVEC BUTEE ELASTIQUE
DEBUT(PAR_LOT='NON')

# ----------- Construction du modèle moyen éléments finis puis réduction sur les
modes élastiques à l'aide des commandes suivantes :
AFFE_MODELE, AFFE_CARA_ELEM,  CALC_MATR_ELEM ,ASSE_MATRICE, MODE_ITER_SIMULT,...

MACRO_PROJ_BASE(  BASE=MODE200,  NB_VECT=5,
                        PROFIL = 'PLEIN',
      MATR_ASSE_GENE=(      _F( MATRICE = CO("MA_G"),  MATR_ASSE = MATM), …

# ----------- Préparation de la boucle des simulations de Monte-Carlo
ns=50     # 50 réalisations des processus stochastiques (50 tirages)
DELTA_M = 0.2  # coefficients de dispersion
DELTA_K = 0.2
DELTA_D = 0.2

# ----------- Début de la boucle des simulations de Monte-Carlo
for k in range(1,ns+1):

# Génération des réalisations aléatoires des matrices généralisées de masse,
# raideur et amortissement
    MATM = GENE_MATR_ALEA(   MATR_MOYEN =  MA_G,  COEF_VAR = DELTA_M)

    MATK = GENE_MATR_ALEA(   MATR_MOYEN =  RI_G,  COEF_VAR = DELTA_K)

    MATD = GENE_MATR_ALEA(   MATR_MOYEN =  AM_G,  COEF_VAR = DELTA_D)

# Génération d'une réalisation aléatoire de la raideur de choc
    KN =GENE_VARI_ALEA(   TYPE='GAMMA',
                          BORNE_INF=0.,
                          VALE_MOY=25000.,
                          COEF_VAR=0.01)
    VKN = KN['NBRE',1]

# Calcul d'une réalisation aléatoire du processus stochastique «réponse
# dynamique»
    DM=DYNA_TRAN_MODAL(  METHODE='EULER',
                           MASS_GENE  = MATM,
                           RIGI_GENE  = MATK,
                           AMOR_GENE  = MATD,
                           INCREMENT=_F(   INST_INIT = 0.,
                                           INST_FIN=4.,
                                           PAS = 0.00005),
                           EXCIT=_F(   VECT_GENE = IM_G,
                                       FONC_MULT = IMPULF),
                           CHOC=_F( NOEUD_1 = 'N3201',
                                    OBSTACLE = PLANZ,    JEU = 0.002,
                                    RIGI_NOR = VKN,
                  RIGI_TAN = 0.,    COULOMB = 0.),   )

# Calcul du SRO du ddl d'observation  3201
    ACC3201=RECU_FONCTION(RESU_GENE = DM,
                          NOM_CHAM='ACCE',
                          NOM_CMP='DZ',
                          NOEUD='N3201')
    SPO3201= CALC_FONCTION(SPEC_OSCI=_F(
                                 NATURE='ACCE',
                                 FONCTION=ACC3201,
                                 METHODE='NIGAM',
                                 NORME=9.81,
```

```
                                    LIST_FREQ=LFREQ,
                                           AMOR_REDUIT=(0.001) ),
                                    INTERPOL='LOG',)

    # Calcul des estimations statistiques
    if k==1:         #  initialisation à la première réalisation
       UP3201  = CALC_FONCTION(COMB=_F(FONCTION=SPO3201, COEF=1.),)
       INF3201 = CALC_FONCTION(COMB=_F(FONCTION=SPO3201, COEF=1.),)
       M1_3201 = CALC_FONCTION(COMB=_F(FONCTION=SPO3201, COEF=1.),)
       M2_3201 = CALC_FONCTION(PUISSANCE=_F(FONCTION=SPO3201, EXPOSANT=2),)

    else:
       UP1 = CALC_FONCTION(ENVELOPPE=_F(    FONCTION=(UP3201,SPO3201),
                                       CRITERE='SUP'),)  # Maximum  d'échantillon
       INF1= CALC_FONCTION(ENVELOPPE=_F(    FONCTION=(INF3201,SPO3201),
                                       CRITERE='INF'),)  # Minimum d'échantillon

       M1_2 =  CALC_FONCTION( COMB=(_F(FONCTION=SPO3201,COEF=1.),
                                    _F(FONCTION=M1_3201,COEF=1.))) # Somme
       M2 = CALC_FONCTION(PUISSANCE=_F(FONCTION=SPO3201,    EXPOSANT=2),
       M2_2 =  CALC_FONCTION( COMB=(_F(FONCTION=M2,COEF=1.),
                                 _F(FONCTION=M2_3201,COEF=1.))) # Somme des carrés

      # Renommages et destructions (car concepts non réentrant)
      DETRUIRE(CONCEPT=_F(NOM=(UP3201,INF3201,M1_3201 ,M2_3201,M2)))
      UP3201=  CALC_FONCTION(COMB=_F(FONCTION=UP1,      COEF=1.),)
      INF3201= CALC_FONCTION(COMB=_F(FONCTION=INF1,       COEF=1.),)
      M1_3201=  CALC_FONCTION(COMB=_F(FONCTION=M1_2,       COEF=1.),    )
      M2_3201=  CALC_FONCTION(COMB=_F(FONCTION=M2_2,        COEF=1.), )
      DETRUIRE(CONCEPT=_F(NOM=(M2_2,M1_2,INF1,UP1)),INFO=2)

    # fin if
    DETRUIRE(CONCEPT=_F(NOM=(DM,ACC3201,SPO3201, MATM, MATD, MATK100,KN)) INFO=2)

# fin for (boucle simulations M.C)

M13201=  CALC_FONCTION(COMB=_F(   FONCTION=M1_3201, COEF=1./ns),)  # Calcul de la
                                                                      moyenne
M23201=  CALC_FONCTION(COMB=_F(   FONCTION=M2_3201, COEF=1./ns),)  # Calcul de la
                                                                      variance
N3201=  CALC_FONCTION(NORME=_F(FONCTION=M13201),)       # Calcul de la norme L2

FIN()
```

Page laissée intentionnellement blanche.