


# Code\_Aster, un logiciel souple et ouvert



Calcul des effets de la houle sur un mat d'éolienne par un chaînage STREAM-GIBI-Code\_Aster.

L'utilisateur avancé peut intervenir facilement dans le code, en partie grâce à PYTHON, pour par exemple : écrire des applications métiers, introduire éléments finis et lois de comportement, définir de nouveaux formats d'échanges.

## Le langage de commandes

L'utilisateur de *Code\_Aster* décrit les paramètres et le cheminement de son étude dans un fichier texte. Celui-ci est composé de commandes contenant elles-mêmes des mots-clés qui reçoivent en arguments : textes, entiers, réels ... ou des noms de concepts précédemment créés par d'autres commandes. Ces concepts sont des objets nommés par l'utilisateur, produits par des commandes et potentiellement utilisables ou modifiables par d'autres.

La grammaire et le vocabulaire de ce langage de commande, propre à *Code\_Aster* mais écrit sur base PYTHON, sont décrits dans des catalogues. Pour composer des fichiers de commande corrects, l'utilisateur devra soit connaître les règles élémentaires d'écriture PYTHON, s'informer de la syntaxe de chaque commande dans la documentation, soit plus interactivement utiliser la saisie graphique des commandes d'EFICAS.

## Superviseur et PYTHON

Dans une utilisation plus avancée, l'utilisateur peut, grâce à PYTHON, introduire de la programmation dans son jeu de données : de la plus simple (structures de contrôle, boucles, tests), à la plus complexe exploitant toute la richesse de ce langage interprété (méthodes, classes, importation de modules exogènes comme TkInter, pour les IHM, numerical pour des usages mathématiques ...).

Sur cette illustration, au cours d'un unique job et sans manipulation de fichiers par l'utilisateur, le même calcul est conduit plusieurs fois en appelant un modeleur/mailleur distant avec modification du rayon de cintrage du coude. La boucle s'arrête conditionnellement sur une valeur limite de contrainte.

Tout objet élémentaire de la mémoire FORTRAN peut être récupéré dans l'espace de nom PYTHON par l'utilisateur : dans cet exemple, il s'agit d'un indicateur de contrainte maximum dans le coude du tuyau. Pour les concepts globaux les plus utiles et généraux (tables, résultats, champs, maillages), des passerelles existent et permettent d'en obtenir une représentation PYTHON à fins de manipulation dans ce langage.

D'autre part, il est possible de définir facilement des macro-commandes écrites directement en PYTHON dans le fichier de commande. Cela permet d'encapsuler une séquence de commandes récurrente et d'épurer ses entrées-sorties. L'utilisateur n'a donc plus à surcharger l'exécutable pour en profiter ! Le développement d'outils métier en est grandement facilité.

Cependant, un fichier contenant d'autres instructions que les seules commandes officielles ne pourra être ni produit, ni édité par EFICAS. L'emploi des fonctionnalités évoquées ci-dessus constitue une utilisation "avancée" du code qui n'est pas, pour l'instant, supportée par cet outil.

Pour le chaînage avec d'autres codes, tout est possible : lancement d'un exécutable tiers par EXEC\_LOGICIEL ou soumission directe en PYTHON ; pour les échanges de champs et de maillage, le format MED est à privilégier.

## Éléments finis et lois de comportement

Si votre problème ne s'accommode pas des 95 lois de comportement actuellement présentes, la programmation ou la modification d'une loi de comportement est aisée. Après avoir enrichi en conséquence les catalogues des commandes DEFI\_MATERIAU et STAT/DYNA\_NON\_LINE des mots clés permettant d'introduire ses paramètres d'entrée, il reste à écrire la routine réalisant l'intégration de cette loi de comportement. Elle fournira les données élémentaires indispensables à l'algorithme de résolution (tenseur des contraintes et variables internes réactualisées, matrice tangente ...). Différents utilitaires facilitent et fiabilisent cette intégration.

De même pour les éléments finis, si vous ne trouvez pas l'élément adéquat dans les 360 existants, vous pouvez créer le vôtre. Moyennant l'appropriation de la démarche décrite dans la documentation de développement, introduire un nouvel élément fini est modulaire et ne nécessite pas de maîtriser tout le code.

## Echanges pré/post dans différents formats

*Code\_Aster* est aussi un solveur souple par le nombre de ses formats d'échange et de stockage des données. Maillages, champs et résultats peuvent être lus et écrits dans la plupart des standards connus. Maquetter en PYTHON une interface de traduction de maillages et de résultats dans un nouveau format est aisé. L'image mémoire des objets calculés, éventuellement conservée en fin d'exécution, peut être écrite dans un format portable sur toute plate-forme.



Optimisation du rayon de cintrage d'un coude par un chaînage Gmsh- Code\_Aster.