CSMA 2017
13ème Colloque National en Calcul des Structures
15-19 Mai 2017, Presqu'île de Giens (Var)

`escale` : plateforme logicielle pour le dialogue essai – calcul

JD. Garaud[1], S. Feld-Payet[1], F. Bettonte[1], A. Tireira[1], Y. Le Sant[2], G. Le Besnerais[3], S. Belon[4]

ONERA – The French Aerospace Lab, France, centres de Châtillon[1], Meudon[2], Palaiseau[3] et Lille[4]. jean-didier.garaud@onera.fr

Résumé —
Une plateforme logicielle, dénommée `escale` a été développée dans le but d'adresser trois thématiques : la corrélation d'images, la confrontation essais-calculs et l'identification. Elle intègre actuellement deux noyaux de corrélation d'images, l'un local, l'autre par éléments finis ; elle interagit avec le code Z-set pour la comparaison corrélation d'images – calcul éléments finis ainsi que pour l'identification par FEMU, tout en étant aussi ouverte sur tout autre code, commercial par exemple. Les interactions entre ces composants sont gérées à travers le langage Python et des modèles de données Numpy.
Mots clés — dialogue essai-calcul, corrélation d'images, FEMU, interopérabilité, Python

1 Introduction

Les systèmes de mesure de champs connaissent actuellement un engouement très important au sein de la communauté concernée par la caractérisation mécanique des matériaux et des structures. On assiste en effet à une diffusion croissante de divers systèmes de mesure sans contact qui associent étroitement l'usage de caméras, de logiciels de traitement d'images ou de montages optiques bien spécifiques, ceci afin d'obtenir des champs de déplacements, de déformation ou de température à la surface (voire dans le volume) d'éprouvettes sollicitées mécaniquement.

Parmi les principales techniques de mesure de champs figure la corrélation d'images numériques (CIN), qui a émergé au début des années 1980. Le succès qu'elle rencontre dans le domaine de la mécanique des solides expérimentale depuis quelques années vient principalement de sa très grande simplicité de mise en œuvre, puisqu'elle consiste à la base à apparier des images pourvues d'un marquage aléatoire ou régulier de la surface analysée pour déduire un champ de déplacements. Dans le contexte de la caractérisation mécanique de matériaux et de structures, cette technique peut potentiellement être utilisée à différents niveaux.

La grande quantité de données expérimentales *a priori* disponibles par corrélation d'images a très vite été perçue comme un excellent moyen d'enrichir le dialogue entre modélisation et expérimentation. De fait, la comparaison entre résultats de calculs et mesures, alimentée par la volonté naturelle de rendre les deux les plus proches possibles, a ouvert la voie à l'identification à partir de mesures de champs. Par rapport à une démarche expérimentale "classique", celle-ci donne potentiellement la possibilité d'extraire des paramètres pilotant des lois de comportement à partir d'essais plus complexes (dans le sens où ils sont réalisés sur des éprouvettes de géométrie quelconque et soumises à des chargements moins conventionnels). Ceci permet d'extraire des paramètres plus proches du comportement des matériaux dans une structure en service. Dans le même esprit, il devient également possible d'identifier simultanément les paramètres de lois décrivant le comportement de matériaux hétérogènes.

Ces objectifs nécessitent une plateforme logicielle, capable d'intégrer des éléments de modélisation d'ordre phénoménologique et physique, ainsi que des données de sources différentes lors du traitement. Cette plateforme est conçue de manière suffisamment générique pour être déployée à l'échelle d'un centre de recherche tel que l'ONERA : laboratoires d'essais mécaniques pour des matériaux et conditions expérimentales très variées (échelles spatiales, températures, vitesses de sollicitations), installations d'essais en mécanique des fluides, souffleries. Elle réutilise des outils d'analyse d'images existants, tel que le code de stéréo-corrélation d'images FOLKI [1], intègre de nouveaux développements tels qu'une méthode de corrélation par éléments finis, elle est aussi interopérable avec le code Z-set [2] pour le dialogue entre CIN et simulations mécaniques par éléments finis.

D'un point de vue conception logiciel, cette plateforme a enfin deux caractéristiques importantes : d'une part en intégrant des composants externes via des interfaces simples et ouvertes, permettant au besoin le remplacement de ces composants ; d'autre part en permettant aux utilisateurs de contribuer et mutualiser leurs développements spécifiques au sein d'une forge logicielle.

2 Python pour l'assemblage de composants et l'interopérabilité

En architecture logicielle, un "composant" est un élément logiciel ayant vocation à être assemblé à d'autres composants pour réaliser une application. Le lien se fait au moyen d'une interface bien définie (données d'entrée/sortie, méthodes, protocole d'utilisation). Un composant peut prendre la forme d'une bibliothèque ou d'un programme effectuant un service, au travers d'une interface et de ses dépendances.

Les avantages clés d'un tel design sont la réutilisation de briques élémentaires, une simplification du travail collaboratif (plusieurs développeurs pouvant être responsables d'un composant différent), et une meilleure robustesse (en permettant la validation individuelle des composants). En d'autres termes, ce type de design permet et encourage une architecture modulaire bien identifiée. L'assemblage de composants est issu d'un cadre bien plus général dans les environnements informatiques ; Unix est par exemple construit suivant cette "règle de modularité" : écrire des éléments simples, connectés par des interfaces propres [3].

Le langage Python fait de nos jours consensus au sein de la communauté du calcul scientifique : il est clair d'utilisation, a de bonnes performances (pour peu qu'il soit bien utilisé : privilégier par exemple les opérations vectorielles Numpy par rapport aux boucles sur les tableaux). Son enseignement en classes préparatoires et en cycles universitaires est un atout supplémentaire. Son utilisation comme langage haut niveau pour l'assemblage de composants a deux principaux avantages, d'une part parce qu'il a été lui-même conçu et est présenté comme un assemblage de modules, d'autre part pour sa richesse en calcul scientifique (calcul matriciel, affichage, optimisation, etc.). Il encourage donc naturellement son utilisateur à avoir de bonnes pratiques pour le développement d'outils et composants logiciels (lire à ce titre le "Zen de Python" [4], ou Ramalho [5]).

Les différents composants nécessaires pour le dialogue essai – calcul (corrélation d'images, calcul éléments finis, optimisation, etc.) existent dans des langages différents (C, C++, CUDA, Python, Fortran). Cython [6] est un langage à mi-chemin entre le Python et le C, qui permet un dialogue relativement aisé entre tous ces langages. L'échange de données entre composants se fait par tableaux Numpy sur les types de base du C (int, char, float) ; ces tableaux sont nativement alignés sur les tableaux C, donc permettent un partage direct de zone mémoire, autrement dit des échanges de données à coût quasi nul. Pour la représentation sur disque, les deux formats retenus sont celui natif de Numpy et HDF5.

Les caractéristiques du langage Python (langage de programmation interprété) présentent enfin des avantages supplémentaires : du point de vue de l'utilisateur, l'automatisation de traitements est aisée et naturelle au travers de scripts, puisque c'est un langage de programmation. Python est aussi un langage de description de données au travers de ses conteneurs et classes, il est donc utilisable comme format de description d'un jeu de données tant en mémoire que sur disque (en utilisant le pattern "données en tant que module").

3 Ecosystème

L'architecture de la plateforme est la suivante : au cœur de l'application se trouve le script utilisateur, assemblant les différents composants pour une application dédiée.

En l'état actuel, les différents composants intégrés à la plateforme sont présentés sur la figure 1. Python et Cython servent de langage d'assemblage et Scipy au calcul scientifique ; les codes Z-set, Abaqus ou Europlexus sont utilisés pour la simulation par éléments finis ; les bibliothèques FOLKI, Z-set/DIC, scikit et openCV pour le traitement d'images ; la bibliothèque Qt/PySide et le module matplotlib pour l'interface graphique.

Un assemblage simple est par exemple la conjonction de scikit et FOLKI pour la corrélation d'un jeu d'images complet issu d'une campagne d'essai. Des assemblages plus complexes sont possibles, enchaînant par exemple corrélation, projection sur un maillage, comparaison avec un résultat de calculs

FIGURE 1 – Écosystème de composants de la plateforme

éléments finis et utilisation de ces résultats dans une boucle d'identification de paramètres matériau (méthode FEMU [7]). À ces composants de calcul scientifique et de manipulation de données, on peut encore, pour des applications spécifiques, bénéficier de la richesse des bibliothèques autour de Python, et utiliser des composants de calculs distribué ou de transferts de données pour délocaliser certaines parties de l'analyse.

Corrélation d'images

Deux modules sont disponibles pour la CIN et sont en partie interchangeables. Ils partagent une interface similaire : entrées et sorties, protocole d'utilisation (le paramétrage est par contre spécifique à chaque méthode).

Le module de corrélation d'images par méthode des éléments finis fournit un exemple complet de l'aspect modulaire de la plateforme. Si l'on s'y intéresse plus en détails, celui-ci a été développé à partir des éléments bibliographiques [8, 9, 10]. Le problème de corrélation peut s'énoncer de la manière suivante : étant données f et g deux images (initiale et déformée) d'une éprouvette, trouver u minimisant l'écart :

$$\eta^2 = \int_\Omega [f(x) - g(x+u(x))]^2$$

En projetant u sur la base des fonctions de forme $U = \sum_i u_i N_i$, on aboutit [8] à l'écriture discrète :

$$KU = b$$

avec

$$K_{ij} = \int_\Omega \nabla f N_i \nabla f N_j \quad \text{et} \quad b_i = \int_\Omega (f(x) - g(x+u)) \nabla f N_i$$

Ayant à disposition le code Z-set, le développement du module se base entièrement sur les classes de base de Z-set : maillage et bibliothèque d'éléments géométriques, matrices creuses et solveurs linéaires, méthodes de transfert et projection, entrées/sorties. De plus, cela permet de bénéficier spontanément de la suite de pré- et post-traitements (génération de maillages, visualisation).

La partie coûteuse (intégration numérique des matrices élémentaires et du second membre, assemblage et inversion du système linéaire global) de la résolution est réalisée en C/C++ pour des raisons de performances. Au contraire, la partie haut niveau de l'algorithme (résolution du problème non-linéaire de Newton) est gérée dans un code purement Python.

La figure 2 présente l'organisation de ce module. Partant des objets de base de Z-set : éléments, maillage, matrices (représentés dans la colonne de gauche de la figure), quelques objets spécialisent leur comportement pour la corrélation (2e colonne). En particulier, une classe dérive des éléments standards, pour y définir les règles d'intégration numérique spécifiques (telle que la règle de quadrature par subdivision proposée dans [10]).

La couche suivante est l'interface entre la bibliothèque C++ de Z-set et Python. Celle-ci est générée à l'aide de Cython, langage qui permet d'exposer à Python des objets C++ avec la possibilité éventuelle de

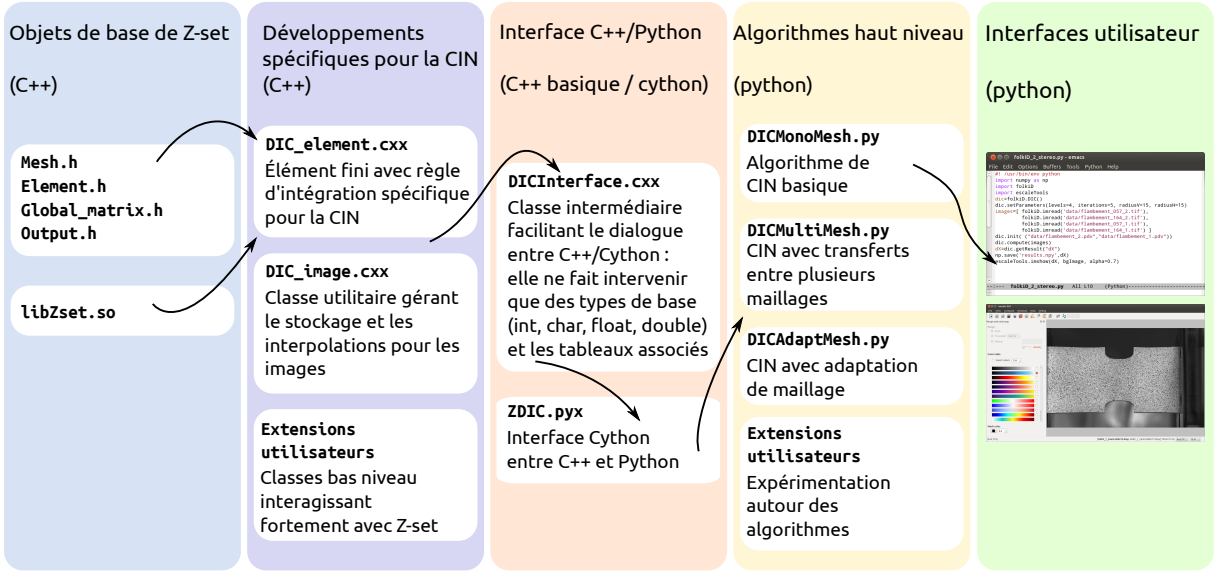

FIGURE 2 – Développement du module de CIN par éléments finis : de Z-set jusqu'à l'utilisateur

modifier (en vue de clarification, spécialisation ou simplification) son interface. Afin de ne pas tirer vers Python tous les objets de Z-set, on a choisi de redéfinir une légère couche interface en C++. Ce besoin est en fait une recommandation plus générale pour l'interopérabilité de différentes briques logicielles : il est plus simple de n'utiliser, au niveau des interfaces d'un composant, uniquement les types de base du C : int, char, double, afin de minimiser les inter-dépendances. Ce respect permet d'ailleurs une interopérabilité entre bibliothèques qui ne se limite pas seulement aux langages C/C++/Python ; on le retrouve dans les bibliothèques de calcul numérique telles que BLAS (Basic Linear Algebra Subprograms) ou MPI (Message Passing Interface) qui sont utilisées en C et Fortran.

La quatrième colonne de la figure 2 correspond aux algorithmes de résolution du problème non linéaire. Son implémentation en Python permet une transcription littérale de l'algorithme en "version papier" à sa version informatique. Ainsi, plusieurs variantes sont proposées : avec ou sans adaptation de maillage, déraffinement de l'image, etc. L'utilisateur avancé peut ajouter rapidement sa contribution à cette brique.

Enfin (5e colonne), l'interface utilisateur existe sous deux formes : une interface graphique (Fig. 3) ou un script, qui permet de paramétrer et automatiser la résolution pour du traitement par lot, ou pour le traitement d'essais complets en chargement variable.

La bibliothèque FOLKI de stéréocorrélation, initialement écrite entièrement en C++/CUDA a été portée en module Python sur le modèle des colonnes 3-4-5 de la figure 2, pour être intégrée à la plateforme, et pour en tester des variantes, par exemple en intégrant différentes solutions algorithmiques pour la reconstruction stéréoscopique. On réfère à [1] pour plus de détails sur cette méthode.

4 Conclusion

La plateforme `escale` de dialogue essai-calcul regroupe plusieurs composants de calcul scientifique pour la mécanique : corrélation d'images, transferts de champs, simulations, méthodes d'identification, optimisation. Ceux-ci sont encapsulés dans des modules Python pour faciliter leur utilisation et leurs interactions respectives. La définition d'interfaces simples, claires et documentées à ces composants permet une intégration dans la plateforme commune. Le langage Python est utilisé à la fois comme langage de développement pour les interfaces de haut niveau, pour l'assemblage, et comme interface utilisateur.

Un ensemble de bonnes pratiques a permis de développer et mettre en production une plateforme déjà suffisamment riche, et extensible par la suite. De plus, les langages et bibliothèques retenus permettent, moyennant un effort raisonnable, un déploiement sous Windows et Linux. Le trio Python/Cython/Sphinx fournit un canevas clair et bien cadré pour ces développements. L'ensemble s'est montré particulièrement

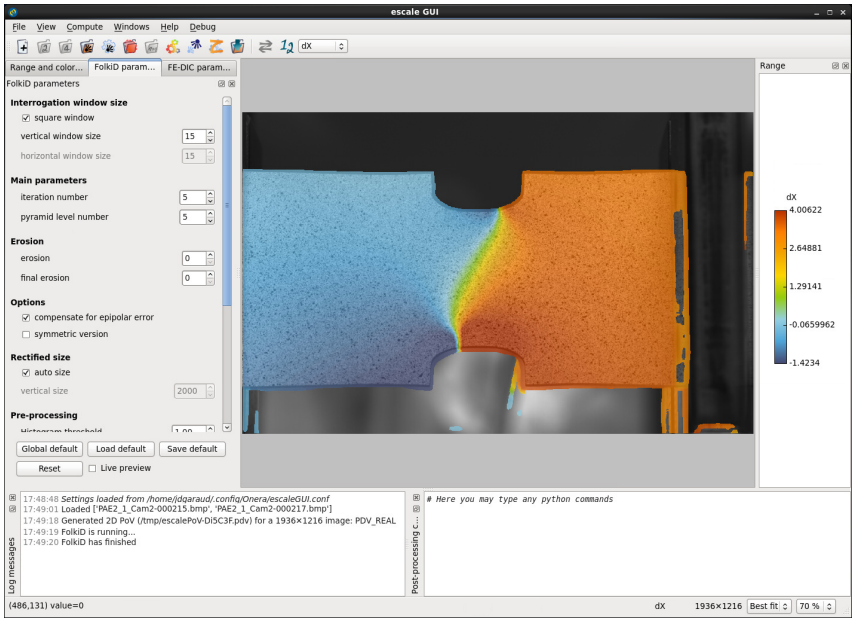

FIGURE 3 – Capture d'écran de l'interface graphique de la plateforme `escale`

accessible aux nouveaux utilisateurs, ainsi qu'à des fins pédagogiques.

Références

[1] G. Le Besnerais, Y. Le Sant, D. Lévêque *Fast and Dense 2D and 3D Displacement Field Estimation by a Highly Parallel Image Correlation Algorithm*. Strain, 2016.

[2] Z-set, *Material and structure analysis suite*, `http://www.zset-software.com/`.

[3] Eric S. Raymond. *The Art of UNIX Programming*, Addison-Wesley Professional Computing Series, 2003, ISBN-13 : 978-0-13-142901-7. Aussi disponible sur `http://www.catb.org/~esr/writings/taoup/`.

[4] PEP 20 – The Zen of Python, `https://www.python.org/dev/peps/pep-0020/`.

[5] Luciano Ramalho, *Fluent Python*, O'Reilly, 2005.

[6] Cython : C-extensions for Python, `http://cython.org/`

[7] M. Grédiac and F. Hild, *Mesures de champs et identification en mécanique des solides*, Lavoisier, 2011.

[8] G. Besnard, F. Hild and S. Roux, *"Finite-Element" Displacement Fields Analysis from Digital Images : Application to Portevin–Le Châtelier Bands*, Experimental Mechanics, 2006.

[9] J. Réthoré, F. Hild and S. Roux, *Shear-band capturing using a multiscale extended digital image correlation technique*, CMAME, 2007

[10] J.-E Pierré, J.-C. Passieux, J.-N. Périé, F. Bugarin and L. Robert, *Unstructured finite element-based digital image correlation with enhanced management of quadrature and lens distortions*, Optics and Lasers in Engineering, 2016.