Retour d'expérience sur Julia pour la recherche et l'enseignement en recherche opérationnelle

Thibaut Cuvelier[1]

[1] Orange Labs Networks
thibaut.cuvelier@orange.com

Mots-clés *: recherche opérationnelle, optimisation mathématique, Julia, JuMP, enseignement.*

Introduction

Julia est un langage de programmation récent avec une communauté très développée, notamment en recherche opérationnelle et en optimisation mathématique en particulier. Néanmoins, il dispose déjà d'un grand nombre d'avantages très pratiques, dans un contexte tant de recherche que d'enseignement. Cette présentation offre un retour d'expérience sur plusieurs années d'utilisation de Julia.

Deux mots sur Julia

Julia fait partie des derniers-nés dans les langages de programmation, avec sa première version publique en 2012 [1]. Malgré tout, il présente déjà une communauté forte : plus de 500 personnes participent à son développement et 1200 bibliothèques sont disponibles dans son gestionnaire de paquets. Il doit ce succès à un savant mélange entre une syntaxe simple (librement inspirée de MATLAB pour faciliter la prise en main) et une performance très proche des langages compilés (C, Fortran…).

Côté optimisation mathématique, la bibliothèque JuMP [2] s'est montrée très tôt comme une incontournable couche de modélisation, avec une performance équivalente ou supérieure à AMPL. Elle s'est imposée par sa syntaxe très naturelle, proche de l'écriture mathématique des programmes d'optimisation. Par une couche d'abstraction, une vingtaine de solveurs est directement accessible.

Julia dans la recherche

J'ai utilisé Julia dans trois projets de recherche différents. Le premier portait sur la programmation stochastique et robuste. À l'époque, Julia 0.3 venait de sortir et le développement n'était pas aisé ; le résultat était une série de scripts peu réutilisables. Malgré tout, la facilité d'utilisation de JuMP a permis d'obtenir des résultats probants en un temps raisonnable (nettement plus qu'avec Java, par exemple).

Ensuite, un deuxième projet traitait de la gestion de barrages hydrauliques[i]. Les différences entre deux barrages sont assez faibles : la majorité des contraintes est identique. Le code est nettement plus ordonné, avec des structures de données pour stocker les paramètres. Du côté utilisateur, cette organisation permet un code déclaratif pour les paramètres. Néanmoins, mes migrations entre versions de Julia n'ont pas été aisées : le passage de la 0.3 à la 0.4 s'est fait en quelques jours (pour à peine un millier de lignes de code), celui vers la 0.5 a été abandonné.

Finalement, le dernier projet concernait la flexibilisation de l'utilisation de l'électricité[ii]. Contrairement aux barrages, les processus modélisés font état d'une grande variété, ce qui nécessite

d'autres techniques pour la description des paramètres. Les patrons de conception se sont révélés être d'excellents alliés, pour laisser le libre choix à l'utilisateur du type de contrainte à ajouter pour chaque processus. Le système des types de Julia et plus particulièrement les multiméthodes (*multiple dispatch*) aident fortement à la réalisation d'un système aussi flexible. Ce projet a débuté avec la version 0.4 de Julia et la migration vers la 0.6 s'est faite en quelques heures.

Julia dans l'enseignement

Depuis septembre 2015, Julia et JuMP sont utilisés pour le cours d'optimisation discrète à l'université de Liège. Précédemment, les étudiants étaient incités à directement utiliser l'API Java de CPLEX, ce qui leur causait d'importantes difficultés pour la génération de leurs modèles, mais avec l'avantage de l'intégration dans un langage de programmation complet. Certaines séances d'exercices utilisaient AMPL pour la modélisation, sans toutefois utiliser ses scripts. Julia a permis d'unifier ces deux environnements, en proposant une syntaxe plus plaisante pour l'écriture du code générant les modèles, tout en étant un véritable langage de programmation avec une série de bibliothèques.

La première année utilisait la version 0.3 de Julia, la seule disponible en septembre 2015. Les conditions n'étaient pas idéales, puisqu'il n'y avait pas d'environnement de développement complet. Les problèmes d'installation étaient relativement nombreux. Les messages d'erreur n'étaient pas toujours à la hauteur des attentes (légitimes !) des étudiants. Néanmoins, quand le choix leur a été laissé, 11 groupes sur 15 ont utilisé Julia ; parmi eux, 3 ont exclusivement utilisé ce langage, les 8 autres l'ont complété avec Java. Le profil de ces 3 groupes est quelque peu atypique pour le cours, vu qu'ils n'ont pas tous suivi de cursus en informatique. Les 8 groupes qui ont utilisé Julia uniquement pour la modélisation mathématique étaient formés exclusivement d'étudiants informaticiens. Il en ressort que Julia, même dans ses versions moins abouties, est apprécié par les étudiants qui n'ont pas d'habitude de programmation.

La deuxième année, la version est passée à la 0.4. Celle-ci a marqué un tournant dans la maturité du langage et de son environnement, ce qui s'est ressenti dans les statistiques. Ainsi, l'entièreté des 20 groupes a utilisé Julia, dont seuls 4 en conjonction avec un autre langage (3 pour Java, 1 pour Python). Le public était encore plus varié qu'en 2015, avec une proportion moindre d'étudiants informaticiens. Les problèmes d'installation ont été très rares. Les étudiants ont expérimenté des fonctionnalités plus diverses de JuMP, notamment les fonctions de rappel indépendantes du solveur.

Conclusions et perspectives

Globalement, l'expérience Julia est très positive, surtout avec la maturation actuelle de l'environnement et de l'implémentation du langage : malgré des débuts chaotiques (comme pour tout nouveau langage), Julia est désormais très utilisable et recommandable pour un public technique. On peut cependant espérer quelques améliorations d'ici à la version finale, notamment un débogueur.

Références

[1] Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah. *Julia: A Fresh Approach to Numerical Computing.* SIAM Review, 59: 65–98, 2017.

[2] Iain Dunning, Joey Huchette and Miles Lubin. *JuMP: A Modeling Language for Mathematical Optimization.* SIAM Review, 59: 295–320, 2017.

[i] Le code est disponible en ligne : https://github.com/dourouc05/ReservoirManagement.jl.

[ii] Le code est disponible en ligne : https://github.com/dourouc05/IndustrialProcessFlexibilisation.jl.