Compte-rendu Distanciel
Langages de Programmation de Haut Niveau

Romain BERNARD, Loïc BOUTIN,
Ivan DOMIGNY--CHEVREUIL

Introduction :

Le but de ce projet en distanciel est de découvrir les trois langages introduits dans l'UE, le R, le Python et le Julia, et au travers d'un projet simple mais où l'attention est portée sur la performance de découvrir leurs philosophies respectives quand au style de code et d'usage des structures de données de chaque langage afin de s'en servir de façon efficace et en utilisant les fonctionnalités avancées permises par l'usage d'un langage haut niveau.

Tableau comparatif :

Le comparatif est effectué avec les algorithmes du groupe de Romain BROHAN, Samy GASCOIN et Dylan CARON. Pour faire un duel au sommet, chaque groupe prendra le nom de famille de son Romain. Les algorithmes "intentionnellement nuls" de chaque groupe seront symbolisés par des temps en gras. Tous les algorithmes sont testés sur la même machine et sont exécutés 10 fois sur lesquelles on fait la moyenne pour des valeurs de n de 1000 et 100000 afin de se faire une idée de la capacité à scale de la fonction.

	Team Bernard		Team Brohan	
	1000	100000	1000	100000
R	1e-04	0.0282	0.0022	0.3237
Julia	**0.00187347450**	**6.4406946194**	0.0018550173	0.0013854559
Python	7.32421875e-05	0.00501489639	**0.00426287651**	**24.5862233162**

Commentaire :

Un simple coup d'oeil du tableau permet de se rendre compte de quels algos scale le moins bien avec la taille de n. On va donc regarder un peu le code et comparer les différences d'usages faites entre les groupes et ainsi, malgré le peu d'expérience dans ces langages déterminer des utilisations optimisées et intelligentes des langages.

R :

Dans un premier temps, en R, nous avons utilisé une syntaxe spécifique au R applicable aux tableaux pour le filtrer, qui peut s'apparenter à une déclaration descriptive d'ensemble. Sans parler de fonction à proprement parler, on se sert donc d'implémentation d'opérateurs spécifiques au langage pour accomplir l'algorithme, et on remarque par les

chiffres que ça paye. L'algorithme scale bien, montrant que les tableaux sont la structure de base du R et qu'en tant que tel, les fonctionnalités qui y sont associées sont très optimisées, donnant ainsi un code compact mais efficace, et compréhensible pour qui connaît un peu la syntaxe de R.

Côté Team Brohan, une approche assez brute a été utilisée en reproduisant les étapes de l'algorithme sur un tableau de booléens avec deux boucles while (et non pas un break dans un for) . Après actualisation des booléens, la fonction which, donnant les indices ayant la valeur TRUE, ce qui correspond aux nombres premiers de 1 à n. Par soucis d'optimisation, la valeur 1 est passée, et à chaque passage à la seconde boucle, le compteur i est initialisé à 2*cpt, le compteur de la première boucle, réduisant ainsi grandement la quantité de valeurs testées. Hormis l'initialisation du tableau et la fonction which, le coeur de l'algorithme repose peu sur les fonctionnalités de R et est donc moins efficace que notre algorithme, mais les bons choix de valeurs à tester permettent des performances très acceptables sur les tests effectués.

Julia :

L'algorithme en Julia est notre algorithme inefficace et scale vraiment mal malgré une rapidité comparable au groupe Brohan sur la plus petite instance. Notre écriture en Julia est comparable à celle écrite en R. Une boucle dans laquelle on filtre notre tableau de nombres. Pourquoi donc une si grande différence de scaling ? On peut déjà remarquer la différence de condition d'arrêt de la boucle. D'un côté un simple Pour de 2 à n, et en R, on s'arrête une fois que le carré de la valeur testée est supérieur ou égal à n, ce qui permet d'éviter plus de valeurs plus n est grand, ce qui oblige notre algorithme en Julia à filtrer sur toute les valeurs restantes du tableau à chaque tour de boucle.

Comme précédemment, l'algorithme du groupe Brohan reprend une implémentation semblable à la description du crible avec les mêmes optimisations que sous R, à savoir la valeur de j mise à 2*i pour passer beaucoup de valeurs inutiles puis ils retournent l'ensemble des valeurs à Vrai dans leur tableau. Et comme les chiffres le montrent, là où notre algorithme est très rapide et simple à lire en utilisant les fonctions de Julia, des optimisations plus standards du code donnant un algorithme qui hormis l'initialisation du tableau pourrait s'écrire selon la même philosophie en C donnent un algorithme très rapide et qui scale tellement bien que les tests avec le plus grand n sont moins longues que ceux avec un plus petit n, étonnamment.

Même si l'optimisation de filter! sous Julia peut être amenée à être améliorée avec l'avancée du langage (qui est encore en 0.6), les résultats font transparaître la volonté de faire une sorte de mélange entre Python et C avec le langage Julia, donnant un langage où faire des algorithmes optimisés plus traditionnellement est possible tout en bénéficiant de fonctionnalités de haut niveau puissantes si bien utilisées, accélérant la production de code face aux langages bas niveau, en faisant un bon langage de prototypage, tout comme le Python.

Python :

Notre algorithme sous Python est le meilleur de nos algorithmes, et c'est aussi celui qui utilise le plus de syntaxes et fonctions spécifiques au langage. On commence par initialiser la liste avec les deux premières valeurs à Faux puis on remplit le reste de Vrai. Ensuite dans une range de 3 à racine de n (même optimisation que vue précédemment) et en allant de deux en deux, à chaque tour on va donc, si la valeur étudiée est à Vrai, l'ajouter aux résultats et mettre à Faux tout ses multiples. Ensuite, dans l'intervalle restant entre racine de n et n, on récupère les valeurs à Vrai restante. Ainsi, on s'évite des opérations lourdes (passage à Faux des multiples) sur toute une partie de la range étudiée, permettant d'avoir un très bon scaling.

Le groupe Brohan a choisi de faire son algorithme inefficace en Python, où, pour chaque valeur de 3 à n ils vérifient si la valeur testée est divisible par un nombre premier déjà trouvé, faisant beaucoup de comparaisons inutiles.

Python s'assumant totalement comme langage de haut niveau et étant installé depuis longtemps (contrairement à Julia, qui est jeune et essaie de concilier le haut niveau et la possibilité d'optimiser à l'ancienne), il n'est pas surprenant de voir beaucoup de syntaxe très spécifique au Python, mais ce focus sur ses spécificités donne un algorithme extrêmement rapide en tirant partie des fonctionnalités haut niveau de Python.

Conclusion :

A travers ce distanciel, on voit que les langages de haut niveaux ont chacun leurs spécificités. R, un langage de statistique dispose de façons très optimisées d'utiliser les tableaux de données tandis que Python se focalise sur des fonctionnalités de très haut niveau qui grâce à sa longue installation et sa grande communauté son très optimisées et permettent de prototyper des algorithmes simplement et rapidement (autant en temps de conception que d'exécution). Julia quand à lui est encore très jeune, et a une communauté naissante, ce qui couplé à son double objectif font que les fonctionnalités haut niveau sont encore à utiliser prudemment, mais les résultats avec une optimisation à l'ancienne laissent entrevoir de très bon résultats une fois le langage plus développé.