Français

- accueil
- source
- téléchargements
- documentation
- paquets
- blog
- communauté
- ecosystems
- apprendre
- enseigner
- publications
- GSoC
- juliacon

Julia est un langage de programmation de haut niveau et à haute performance pour le calcul numérique. Il offre un compilateur sophistiqué, du calcul parallèle distribué, une bonne précision numérique, et une vaste bibliothèque de fonctions mathématiques. La bibliothèque de base de Julia, écrite essentiellement en Julia, intègre aussi des bibliothèques open source C et Fortran qui sont matures et haut de gamme pour l'algèbre linéaire, la génération de nombres aléatoires, le traitement du signal, et le traitement de chaînes de caractères. De plus, la communauté de développeurs de Julia participe à l'élaboration de plusieurs paquets externes à l'aide du gestionnaire de paquets intégré de Julia à un rythme soutenu. IJulia, le fruit d'une collaboration entre les communautés Jupyter et Julia, fournit une puissante interface graphique par navigateur à Julia sous la forme de blocs-notes.

Les programmes Julia sont organisés autour de la distribution multiple (multiple dispatch) ; en définissant les fonctions et en les surchargeant pour différentes combinaisons de types d'arguments, qui peuvent également être définis par l'utilisateur. Pour une discussion plus approfondie de la justification et des avantages de Julia sur d'autres systèmes, voir les points suivants ou lire l'introduction dans le manuel en ligne.

JuliaCon 2017, the annual conference on Julia was held from June 20th to June 24th at the University of California, Berkeley. Below is a random video from our youtube playlist of the talks. Click on the playlist icon to check out the other videos.

Un résumé des fonctionnalités

- Distribution multiple (multiple dispatch) : permettre de définir le comportement des fonctions suivant de nombreuses combinaisons de types d'arguments
- Système de typage dynamique : types pour la documentation, l'optimisation et la distribution (dispatch)
- Bonne performance, proche de celle de langages à compilation statique comme C
- Gestionnaire de paquets intégré
- Macros dans le style Lisp et autres outils de méta-programmation
- Appel de fonctions Python : utilisez le paquet PyCall

Français

- Appel direct de fonctions C : pas d'API spéciale ou d'enveloppe (wrapper)
- Capacités de type shell puissantes pour la gestion d'autres processus
- Conçu pour le parallélisme et le calcul distribué
- Coroutines : fils d'exécution légers (green threading)
- Les types définis par l'utilisateur sont aussi rapides et compacts que ceux intégrés au langage
- Génération automatique de code spécialisé et efficace pour les différents types d'arguments
- Conversions et promotions élégantes et extensibles pour les types numériques et autres
- Prise en charge performante d'Unicode, dont notamment UTF-8
- Sous licence MIT : libre et open source

Compilateur JIT à haute performance

Julia's LLVM-based just-in-time (JIT) compiler combined with the language's design allow it to approach and often match the performance of C. To get a sense of relative performance of Julia compared to other languages that can or could be used for numerical and scientific computing, we've written a small set of micro-benchmarks in a variety of languages: C, Fortran, Julia, Python, Matlab/Octave, R, JavaScript, Java, Lua, Mathematica. We encourage you to skim the code to get a sense for how easy or difficult numerical programming in each language is.

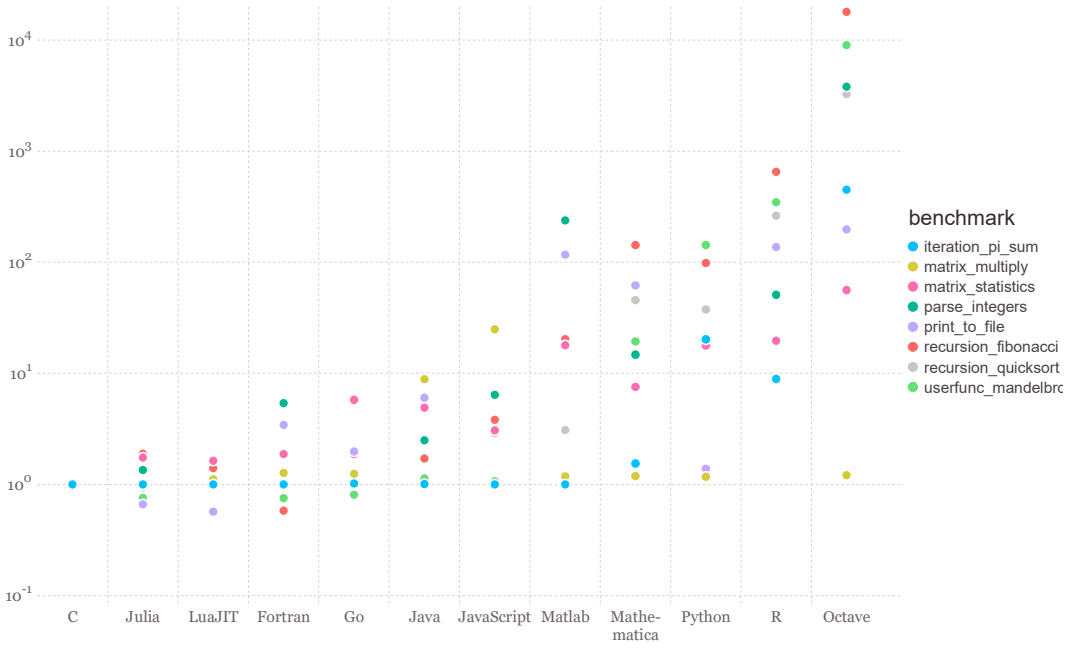


Figure: Benchmark times relative to C (smaller is better, C performance = 1.0). Plot created with Gadfly and IJulia from this notebook. See the benchmarks page for more information.

A quick taste of Julia

Pour donner un avant-goût de ce à quoi Julia ressemble, voici le code utilisé dans les benchmarks de Mandelbrot et des statistiques de matrice aléatoire :

```
function mandel(z)
    c = z
    maxiter = 80
    for n = 1:maxiter
        if abs2(z) > 4
            return n-1
        end
        z = z^2 + c
    end
    return maxiter
end

function randmatstat(t)
    n = 5
    v = zeros(t)
    w = zeros(t)
    for i = 1:t
```

Français

```
        a = randn(n,n)
        b = randn(n,n)
        c = randn(n,n)
        d = randn(n,n)
        P = [a b c d]
        Q = [a b; c d]
        v[i] = trace((P.'*P)^4)
        w[i] = trace((Q.'*Q)^4)
    end
    std(v)/mean(v), std(w)/mean(w)
end
```

Le code ci-dessus est tout à fait clair, et doit être familier à toute personne qui a programmé dans d'autres langages mathématiques. La mise en œuvre de Julia `randmatstat` est beaucoup plus simple que l'implémentation de C , équivalente, sans renoncer à beaucoup de performance. Les optimisations du compilateur sont prévues pour combler cet écart de performance à l'avenir. De par sa conception, Julia vous permet de faire de boucles serrées de bas niveau, jusqu'à un style de programmation de haut niveau, tout en sacrifiant peu de performance, mais gagne en capacité d'exprimer des algorithmes complexes facilement. Ce spectre continu des niveaux de programmation est une caractéristique de l'approche de Julia à la programmation et est une très bonne caractéristique intentionnelle de la langue.

Conçu pour le parallélisme et l'informatique en nuage

Julia n'impose aucun style particulier de parallélisme à l'utilisateur. Le langage fournit plutôt un certain nombre de fonctionnalités de base pour le calcul distribué, ce qui le rend suffisamment flexible pour prendre en charge de multiples styles de parallélisme et permet aux utilisateurs d'en ajouter d'autres. L'exemple simple suivant montre comment compter le nombre de faces dans un grand nombre de jets de pièces en parallèle.

```
nheads = @parallel (+) for i=1:100000000
  rand(Bool)
end
```

Ce calcul est automatiquement distribué à travers tous les noeuds de calcul disponibles, et le résultat, réduit par sommation (+), est renvoyée au niveau du noeud d'appel.

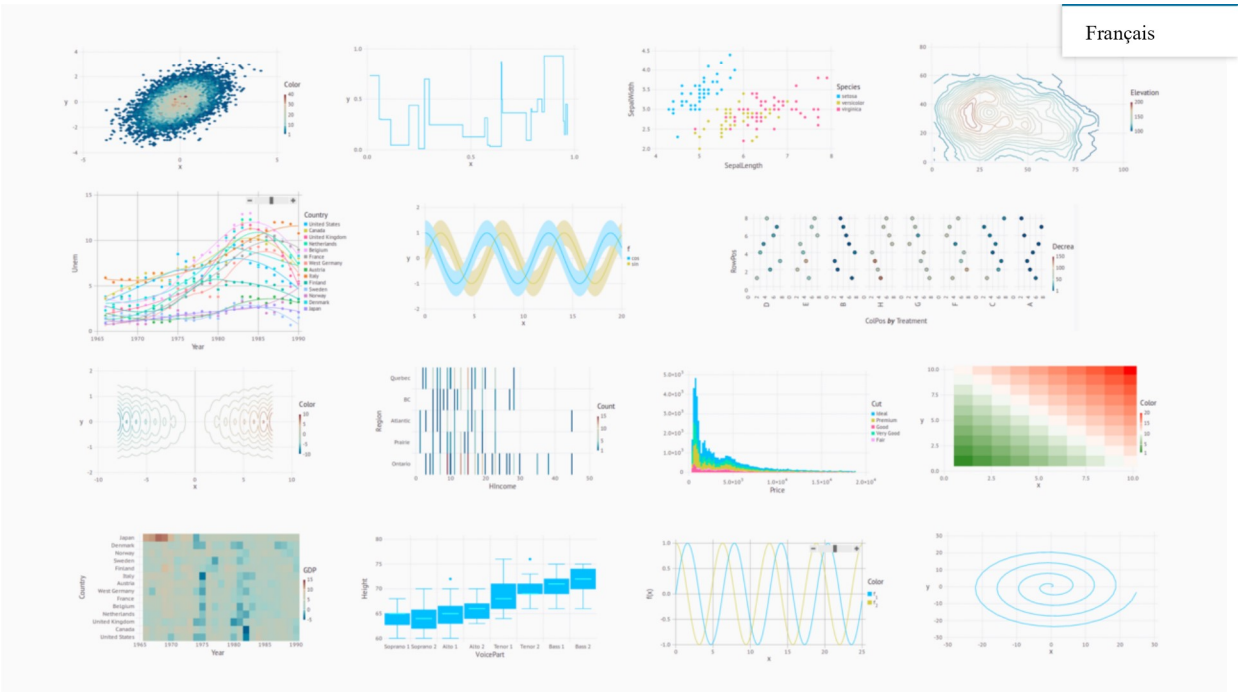
Voici une capture d'écran d'un bloc-notes IJulia interactif dans le navigateur Web, utilisant Gadfly. JuliaBox permet d'exécuter des blocs-notes IJulia dans votre navigateur sur des conteneurs Docker lancés à la demande.

Français

Cela ouvre la voie à un fonctionnement entièrement basé sur le cloud, comprenant notamment la gestion de données, l'édition et le partage de code, l'exécution, le débogage, la collaboration, l'analyse, l'exploration de données et la visualisation. L'objectif final est de faire en sorte que les gens cessent de se soucier de l'administration des machines et de la gestion des données et s'attaquent directement au problème réel.

Gadfly peut produire différents graphiques avec de multiples moteurs de rendu dans le navigateur (SVG, PDF, PNG et divers autres moteurs sont également pris en charge). L'interactivité peut être ajoutée à des graphiques avec le paquet Interact.jl. Une illustration des possibilités offertes par Gadfly est présentée ci-dessous.

Français



Libre, open source et ouvert aux bibliothèques

Le cœur de l'implémentation de Julia est sous licence MIT. Diverses bibliothèques utilisées par l'environnement Julia comprennent leurs propres licences telles que GPL, LGPL, et BSD (par conséquent, l'environnement, qui se compose du langage, des interfaces utilisateur et des bibliothèques, est sous licence GPL). Le langage peut être compilé comme une bibliothèque partagée, afin que les utilisateurs puissent combiner Julia avec leur propre code C/Fortran ou des bibliothèques propriétaires tierces. En outre, Julia simplifie l'appel des fonctions externes vers des bibliothèques partagées C et Fortran, sans écrire de code d'emballage (wrapper) et même sans recompiler le code existant. Vous pouvez essayer d'appeler des fonctions de bibliothèque externes directement à partir de l'invite interactive de Julia, obtenant ainsi immédiatement le résultat. Voir le fichier LICENSE pour les conditions complètes de la licence de Julia.

Julia est un projet NumFocus. Nous remercions Fastly pour leur généreux soutien en infrastructure. Éditer cette page sur GitHub.

Faire un don