introJulia

September 1, 2021

Introduction à Julia

Fabian Bastin (fabian.bastin@umontreal.ca), 2017-2021

0.1 Premiers pas en Julia

La page officielle du langage de programmation Julia est http://julialang.org

- Conçu pour répondre au problème de double langage (un pour le développement rapide, un pour la production), en alliant efficacité avec simplicité de code.
- Concerne avant tout le calcul scientifique.
- JIT (just-in-time): le code Julia est compilé au moment de son exécution, un peu à l'image de Java.
- Version actuelle: 1.6.2.
- Croissance exponentielle des librairies disponibles.
- Peut s'interfacer avec des langages tels que Python, C++, Fortran, R,…
- Versions commerciales pour l'infonuagique: JuliaHub

0.1.1 Pourquoi pas Python?

- Python est idéal pour le prototypage, mais souffre de son côté interprété pour les boucles.
- Julia est un langage fortement typé, permettant des performances proches de C/C++ (à condition de veiller à la qualité de son implémentation).

0.2 Installation

L'interpréteur et compilateur de Julia peut être téléchargé à l'adresse http://julialang.org/downloads/

Julia peut être installé sur les OS majeurs: Windows, MacOS X, Linux. Sous Windows, il est possible de lancer Julia via le menu démarrer ou, si présente, en cliquant sur l'icône adéquate. Sous Linux, pour lancer l'interpréteur Julia en mode interactif, il suffit d'entrer au niveau du terminal

julia

Il est également possible d'exécuter un code julia en indiquant le nom du fichier, par exemple

julia hello.jl

où le fichier "hello.jl" contient la commande

[3]:
```
println("Hello World!")
```

```
Hello World!
```

0.3 Tester Julia

Il est également possible de programmation en Julia directement en ligne, notamment à l'adresse https://repl.it

0.4 Environnements de développement

Plusieurs environnements de développement sont disponibles, en particulier - Juno: https://junolab.org - Atom: https://ide.atom.io/ - Visual Studio: https://code.visualstudio.com/

0.5 Librairies

Un nombre important de librairies officielles sont disponibles, mais ne sont pas installées par défaut avec Julia. Il existe plusieurs méthodes pour utiliser le gestionnaire de librairies. Nous allons travailler ici avec les commandes du gestionnaire Pkg, qui nous devons au préalable importer avec la commande

[ ]:
```
import Pkg
```

Nous pouvons à présent ajouter une librairie avec la commande `Pkg.add("Nom de la librairies")`. Par exemple, pour pouvoir réaliser des graphes des fonctions qui nous intéressent, nous utiliserons la librairie Plots. Pour l'ajouter, entrons

[ ]:
```
Pkg.add("Plots")
```

0.5.1 Où se trouvent localement les librairies?

Sous Linux et MacOS, les sources des librairies installées sont disponibles par défaut dans le sous-répertoire `$HOME/.julia`

Sous Windows, il suffit de remplacer $HOME par le répertoire utilisateur.

La liste des librairies officielles ("registered packages") est disponible à l'adresse https://julialang.org/packages/.

Il est conseillé de mettre à jour régulièrement les librairies installées avec la commande

[ ]:
```
Pkg.update()
```

Il est également possible d'importer une librairie non officielle disponible sur GitHub en replaçant le nom de la libraire pas son URL, par exemple

[ ]:
```
Pkg.add(url="https://github.com/fbastin/Jasmine")
```

0.6 Bloc-notes

Julia s'intègre aussi à l'environnement de bloc-notes, permettant de mélanger texte et code, tout en offrant un environnement interactif. Un fichier de bloc-notes se reconnaît à l'extension de fichier `ipynb`.

Pour utiliser Julia avec les bloc-notes, il est nécessaire d'ajouter la librairie IJulia.

[ ]:
```
Pkg.add("IJulia")
```

Pour lire, écrire et modifier des bloc-notes, il est possible d'utiliser `Jupyter` ou `Nteract`. Le premier s'exécute à l'intérieur du navigateur internet tandis que le second est une application indépendante.

0.6.1 Jupyter

L'environnement Jupyter (http://www.jupyter.org) est une application web permettant de créer des documents combinant texte, codes, équations, et graphiques. Il est également possible d'exporter le code source ainsi que de créer des documents pdf et même des présentations interactives.

Le nom vient d'une origine double. Il évoque la planète Jupiter ainsi que les languages majeurs supportés par Jupyter: Julia, Python et R. Sa documentation peut être consultée à l'adresse https://jupyter.org/documentation.

L'installation directe de Jupyter est dépendante de l'OS, mais peut être également réalisée à travers la suite Anaconda, téléchargeable à l'adresse: https://anaconda.com, laquelle offre un environnement intégré pour Python.

Une fois la librairie installée, nous la chargerons en mémoire et utiliserons la fonction 'notebook' pour démarrer Jupyter.

[ ]:
```
using IJulia
notebook()
```

Si aucune version de Jupyter n'est détecté, Julia installera une version minimale de Jupyter. Pour démarrer Jupyter dans le navigateur internet, depuis la ligne de commande, entrez

jupyter notebook

0.7 Nteract

Une alternative à Jupyter est `Nteract`, disponible à l'adresse https://nteract.io/ Son avantage est de fonctionner en dehors du navigateur internet. Par contre, son support de formattage de texte est moins avancé.

`Nteract` étant une interface, le support des bloc-notes doit être installé au sein du langage de programmation utilisé.

Comme indiqué à la page https://nteract.io/kernels/julia, il est nécessaire d'installer le noyau Julia pour pouvoir exécuter du code.

[ ]:
```
# From a Julia prompt
using Pkg
Pkg.update("IJulia")
IJulia.installkernel("Julia nteract")
```

0.8 Boîtes à outil spécifiques

Diverses orientations spécifiques existent, permettant de spécialiser l'usage de Julia pour des besoins spécifiques. Il serait difficile d'être exhaustif, et nous ne citerons ici qu'un cas particulier, mais il existe actuellement des ensembles d'outils pour les principaux sujets de recherche scientifique actifs.

0.8.1 Optimisation

Julia supporte de nombreux algorithmes d'optimisation et propose un langage de modélisation algébrique. Les détails peuvent être trouvés à l'adresse http://www.juliaopt.org

En outre, le langage JuMP offre de modéliser les programmes mathématiques de manière intuitive.

0.8.2 GPU

Julia offre un support des GPU grâce aux librairies reprises à dans le projet JuliaGPU: https://juliagpu.org/

0.8.3 Exporter vers d'autres formats

Il est également possible d'exporter les bloc-notes, directement depuis l'interface de Jupyter, ou en ligne de commandes. Par exemples, pour créer des slides, on pourra utiliser

jupyter nbconvert your_slides.ipynb –to slides

Pour appeler directement le navigateur internet, entrez

jupyter nbconvert your_slides.ipynb –to slides –post serve

Il est également possible d'exporter vers des formats tels que PDF, HTML, etc.

Le principe général reste d'appeler jupyter avec la commande nbconvert, et d'indiquer le format de conversion, par exemple

jupyter nbconvert your_slides.ipynb –to PDF

0.9 Tutoriaux

De nombreux tutoriaux existent, mais il faut veiller à consulter un tutorial à jour pour la dernière version de Julia, comme le langage est encore en cours d'évolution. Le plus simple est de consulter la page https://julialang.org/learning/

Dans l'interprétation, il est possible d'accéder à de la documentation en ligne en introduisant le caractère '?' dans la ligne de commande.

Il est possible de suivre des cours d'introduction sur https://juliaacademy.com/ En particulier, un cours d'introduction est disponible à l'adresse https://juliaacademy.com/p/intro-to-julia

1 Support

Il existe différents canaux de communication avec la communauté Julia. Mentionnons ici https://discourse.julialang.org/

2 Débuter avec Julia

Des bloc-notes interactifs sont disponibles sur https://github.com/JuliaAcademy/JuliaTutorials/blob/master/intro
tutorials/intro-to-julia, en particulier https://github.com/JuliaAcademy/JuliaTutorials/blob/master/introductory
tutorials/intro-to-julia/01.%20Getting%20started.ipynb

2.0.1 Affectation de variables

La syntaxe de base consiste à écrire le nom de variable suivi du signe d'égalité et de la valeur à affecter à la variable, par exemple

[ ]:
```
the_answer_to_life_the_universe_and_everything = 42
```

Note: Google est d'accord https://www.google.ca/search?sxsrf=ALeKk00YlrkNlc4nJGBesWkETj6KN4G3yw%3A
ab&ved=0ahUKEwilt_rr6LHrAhVqTd8KHTBOAREQ4dUDCAw&uact=5

Julia est fortement typé, pourtant nous n'avons pas précisé le type! Par défaut, Julia infère le type de la valeur assignée, et nous pouvons le connaître comme suit:

[ ]:
```
typeof(the_answer_to_life_the_universe_and_everything)
```

Simplifions le nom de la variable, en la transformant au passage en réel:

[ ]:
```
life = convert(Float64,the_answer_to_life_the_universe_and_everything)
```

[ ]:
```
typeof(life)
```

Jupyter (et Julia) supporte le code LaTeX et même les émoticônes! Pour le code LaTeX, il suffit d'écrire la commande LaTeX puis d'appuyer sur tabulation. Les émoticônes s'obtiennent en écrivant en entourant le nom de l'émoticône de `\:` et `:`, par exemple `\:alien:`, suivi de tabulation. Si le nom n'est pas connu en entier, après `\:` et les premières lettres de l'émoticône entrée, l'appui sur tabulation permet d'afficher une liste déroulante où vous pouvez choisir l'émoticône voulue.

[ ]:
```
 = 1.0
 = "alien"
println( *length( ))
```

[ ]:
```
 = 1.0
 = -1.0
 = 0.0
+==
```

2.1 Types

Les types sont organisés hiérachiquement, avec une structure aborescente. Il est possible de connaître le type dont dépend directement un type avec la commande `supertype`:

[ ]:
```
println(supertype(Int64), " ", supertype(Float64))
```

À la racine se trouve le type `Any`, dont dépendent directement un grand nombre de types, qui peuvent être obtenus avec la commande `subtypes`:

[ ]:
```
subtypes(Any)
```

Nous pouvons exposer l'arborescence à l'aide de la function récusive suivante, que nous appliquons au type `Number`, qui a comme prédécesseur direct le type `Any`.

[ ]:
```
# Tiré de https://en.wikibooks.org/wiki/Introducing_Julia/Types
function showtypetree(T, level=0)
     println("\t" ^ level, T)
     for t in subtypes(T)
         showtypetree(t, level+1)
     end
end

showtypetree(Number)
```

3 Tableaux

Les variables peuvent être groupés en tableaux, à une ou plusieurs dimensions. Il est à noter que les indices commencent à 1 en Julia.

[ ]:
```
x = [i for i = 1:10]
```

[ ]:
```
x[1]
```