

Ahmet-kadir ONSEKİZ

Veille technologique



Table des matières

I- Histoire et définition du fonctionnement de Julia	2
A. Histoire de Julia et définition	2
B. JuliaPro et JuliaFin	3
II- Exploitation de Julia : Avantages et inconvénients	9
A. Les avantages du langage Julia	10
1. Le monde de la Science et Julia : facilitation de la collaboration	10
2. Julia et Python : problème d'expression.....	10
3. Julia et machine learning.....	12
B. Inconvénients et Julia	13
Conclusion	14
Bibliographie :	15

Langage informatique ou langage de programmation correspond un langage formel utilisable lors de la conception, lors la mise en œuvre et/ou lors de l'exploitation d'un système d'information. Il existe différents types de langages informatiques comme Fortran, MATLAB, etc. À l'époque, on utilisait davantage de Fortran ou de MATLAB que de Python contrairement à aujourd'hui où ce dernier domine actuellement le marché. Cependant, Python n'est pas assez fonctionnel au niveau de vitesse de calcul. Afin de ressoudre cette anomalie, il y a eu l'apparition de Cython et PyPy pour l'accélération de code en 2007 puis Numba en 2012. Ainsi, à la suite des défauts de ces langages, des chercheurs du Massachusetts Institute of Technology ont développer un nouveau langage le 23 août 2009 : Julia.

Dans cette veille technologique, nous allons nous intéresser à Julia. Dans un premier temps, nous étudierons l'histoire de Julia et son fonctionnement. Puis dans un dernier temps, nous verrons les avantages et les inconvénients de ce langage informatique.

I- Histoire et définition du fonctionnement de Julia

A. Histoire de Julia et définition

Les concepteurs de Julia sont Jeff Bezanson, Stefan Karpinski, Viral Shah, et Alan Edelman. Ces créateurs souhaitaient créer un langage open source avec une licence libre, avec de nombreux avantages, en particulier scientifique.

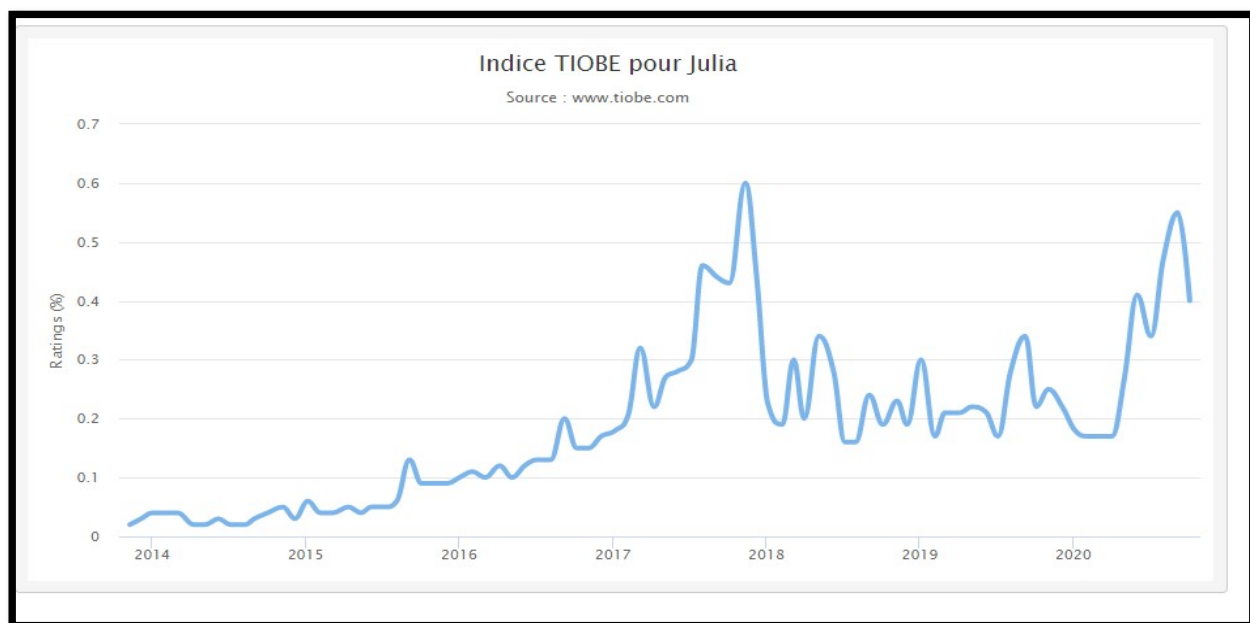
« Nous voulons un langage open source, avec une licence libre. Nous voulons un langage qui associe la rapidité de C et le dynamisme de Ruby. En fait, nous voulons un langage homoiconic, avec de vraies macros comme Lisp et avec une notation mathématique évidente et familière comme Matlab. Nous voulons quelque chose d'aussi utilisable pour la programmation générale que Python, aussi facile pour les statistiques que R, aussi naturel pour la gestion de chaîne de caractères que Perl, aussi puissant pour l'algèbre linéaire que Matlab et aussi bien pour lier des programmes que le Shell. Nous voulons qu'il soit à la fois interactif et compilé. »

(PierreB, 17 novembre 2020, dans revue siltkom.com [1])

Ils élaborent ainsi Julia sous ces objectifs. En effet, Julia est un compilateur, avec un système de types dynamiques, polymorphismes paramétrés avec une exécution parallèle distribuée, des fonctions de Fortran, Python et C en appel direct. Julia est donc rapide, car c'est

un langage compilé qui utilise le compilateur JIT de LLVM. Ainsi cela permet à Julia de se comporter comme un interpréteur comme R ou Python. De plus, sa syntaxe est similaire à celle de MATLAB et de Basic de Microsoft, ce qui garantit que la transition vers Julia est plus facile.

L'intérêt pour Julia s'accroît de plus en plus, laissant apercevoir cet intérêt au travers de plusieurs ouvrages sur le sujet de Julia qui fournissent la fonctionnalité, l'utilisation et la syntaxe de R, Python, etc. De plus, il utilise les principes de POO, ainsi que des mathématiques de base avec Julia. L'un des ouvrages les plus connus est « Beginning Julia Programming for Engineers and Scientists » ou encore « Julia Quick Syntax Reference: A Pocket Guide for Data Science Programming » ou bien « Julia 1.0 Programming Complete Reference Guide: Discover Julia, a high-performance language for technical computing ». Tous ces livres sont des incontournables pour apprendre le langage de Julia plus précisément les bases de Julia et montrent l'intérêt du monde de l'informatique pour ce langage. En effet Julia n'est pas encore beaucoup utilisée et a repris de l'intérêt que récemment comme on peut le voir via le graphique ci-dessous :



B. JuliaPro et JuliaFin

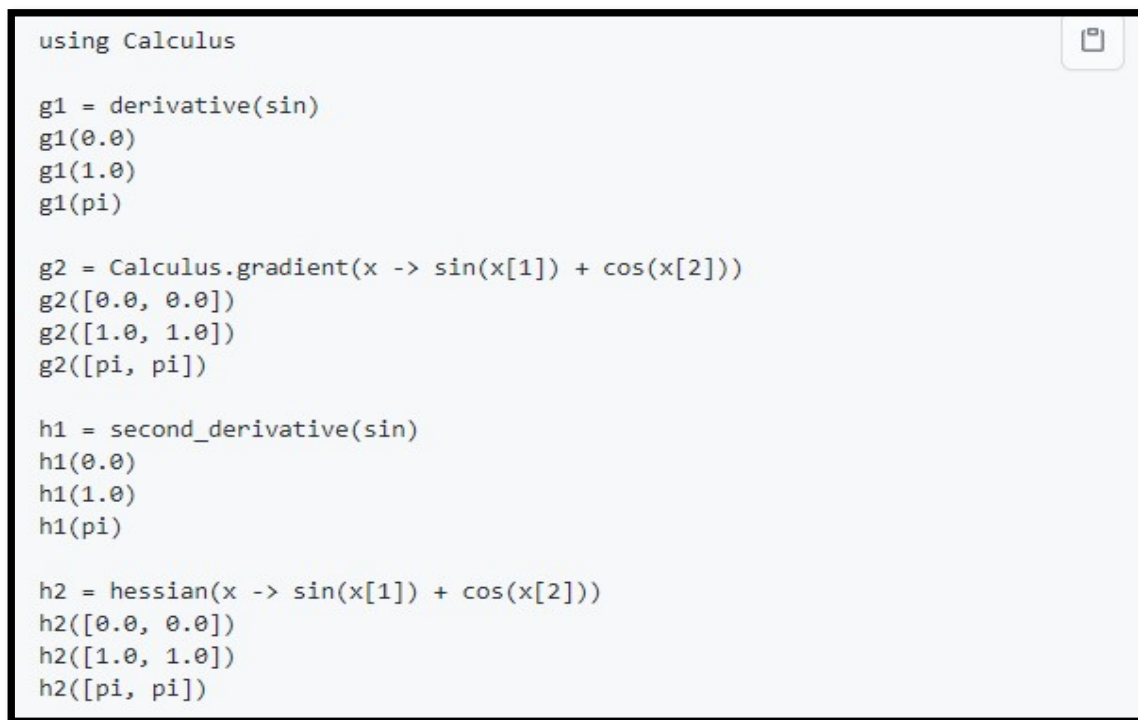
JuliaFin est un produit Julia Computing qui est utilisé en particulier dans la gestion d'actifs, des risques, le trading algorithmique, le backtesting et plein d'autre en rapport du domaine de la finance informatique, y compris la modélisation de contrats financiers.

JuliaPro peut être utilisé sur Mac, Linux, Linux(GPG) et Windows. JuliaPro utilise 2600 packages open source ou des packages répertoriés de 250 packages JuliaPro. Les packages

choisie sont alors testés, documentés, vérifiés et pris en compte par Julia Computing. Voici quelque exemple de packages :

- **Maths generals**

- Calculus : fournit des outils pour travailler avec les opérations de calcul de base de différenciation et d'intégration. Vous pouvez utiliser le progiciel Calculus pour produire des dérivées approximatives par plusieurs formes de différenciation finie ou pour produire une dérivée exacte à l'aide de la différenciation symbolique. Vous pouvez également calculer des intégrales définies par différentes méthodes numériques.

A screenshot of a Julia REPL window showing code that uses the 'Calculus' package. The code defines functions for first and second derivatives of a sine function and a 2D function. The first part uses the 'derivative' function to compute the first derivative of sin(x) at points 0.0, 1.0, and pi. The second part uses 'Calculus.gradient' to compute the gradient of a 2D function at points [0.0, 0.0], [1.0, 1.0], and [pi, pi]. The third part uses 'second_derivative' to compute the second derivative of sin(x) at the same points. The fourth part uses 'hessian' to compute the Hessian of the 2D function at the same points. The code is as follows:

```
using Calculus

g1 = derivative(sin)
g1(0.0)
g1(1.0)
g1(pi)

g2 = Calculus.gradient(x -> sin(x[1]) + cos(x[2]))
g2([0.0, 0.0])
g2([1.0, 1.0])
g2([pi, pi])

h1 = second_derivative(sin)
h1(0.0)
h1(1.0)
h1(pi)

h2 = hessian(x -> sin(x[1]) + cos(x[2]))
h2([0.0, 0.0])
h2([1.0, 1.0])
h2([pi, pi])
```

Fig. 2 : Une opération en utilisant le packages Calculus [3]

- **Apprentissage profond et apprentissage automatique**

- TensorFlow : est une bibliothèque de Machine Learning, Deep Learning, il s'agit d'une boîte à outils qui permet de résoudre des problèmes mathématiques extrêmement complexes (calcul de tenseur). Elle permet aux chercheurs de développer des architectures d'apprentissage expérimentales et de les

transformer en logiciels. Ce package est aussi très utilisé pour des réseaux neuronaux

```
using TensorFlow
using Test

sess = TensorFlow.Session()

x = TensorFlow.constant(Float64[1,2])
y = TensorFlow.Variable(Float64[3,4])
z = TensorFlow.placeholder(Float64)

w = exp(x + z + -y)

run(sess, TensorFlow.global_variables_initializer())
res = run(sess, w, Dict{z=>Float64[1,2]})
@test res[1] ≈ exp(-1)
```

Fig. 3 : Une opération en utilisant le package TensorFlow [4]

- Metalhead : Ce package permet d'avoir des modèles de vision par ordinateur qui s'exécutent sur la bibliothèque d'apprentissage automatique Flux. Chaque modèle est une couche Flux, vous pouvez le déplacer vers le GPU, entraîner ou

```
In [1]: using Metalhead
        using Metalhead: classify

In [2]: vgg = VGG19()
        img = load("Elephant.jpg")

Out[2]: 

In [3]: classify(vgg, img)

Out[3]: "African elephant, Loxodonta africana"

In [ ]: 
```

geler des composants et l'étendre pour effectuer d'autres tâches.

Fig. 4 : Une opération en utilisant le package Metalhead [5]

- **Bases de données**

- JDBC : Ce package permet l'utilisation de pilotes Java JDBC pour accéder aux bases de données depuis Julia. Il utilise le package JavaCall.jl pour

appeler Java afin d'utiliser les pilotes JDBC. L'API utiliser par ce package se compose essentiellement de deux composants : une interface "directe" depuis Java JDBC et une interface julienne minimale avec prise en charge de Tables.jl

```
cnxn = JDBC . Connexion ( " jdbc:derby:test/juliatest " ) # créer une connexion
csr = curseur (cnxn) # créer un curseur à partir de la connexion

# si vous n'avez pas besoin d'accéder à la connexion vous pouvez créer le curseur directement
csr = curseur ( " jdbc:derby:test/juliatest " )

# exécuter du SQL
exécuter ! (csr, " insérer dans pi_table (pi_value) valeurs (3.14); " )
exécuter ! (csr, " select * from my_table; " )

# pour parcourir les lignes
pour la ligne ∈ lignes (csr)
    # faire des trucs avec la
fin de la ligne

close (csr) # ferme la connexion, peut être appelé sur la connexion ou le curseur
```

Fig. 5 : Une opération en utilisant le packages JDBC [6]

- **Interopérabilité avec d'autres langues**

- RCall : Ce package permet d'interagir avec le langage R Peut être utiliser dans Linux & OSX, Windows

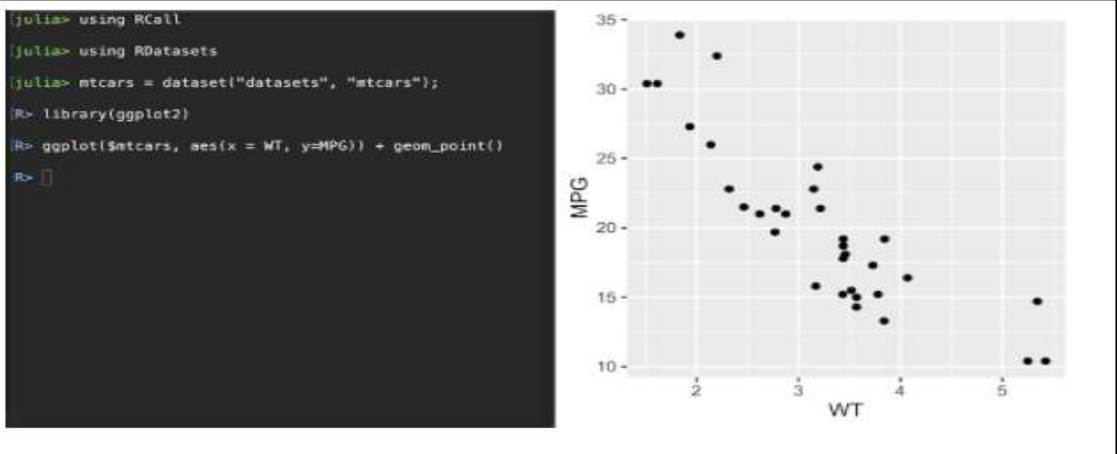


Fig. 5 : Une opération en utilisant le packages RCall [7]

- [JavaCall](#) (Java) : Ce package permet d'interagir avec le langage Java

```
julia> using JavaCall

julia> JavaCall.init(["-Xmx128M"])

julia> jlm = @jimport java.lang.Math

JavaObject{:java.lang.Math} (constructor with 2 methods)

julia> jcall(jlm, "sin", jdouble, (jdouble,), pi/2)

1.0

julia> jnu = @jimport java.net.URL

JavaObject{:java.net.URL} (constructor with 2 methods)

julia> gurl = jnu((JString,), "http://www.google.com")

JavaObject{:java.net.URL}{Ptr{Void} @0x0000000108ae2aa8}

julia> jcall(gurl, "getHost", JString,())

"www.google.com"

julia> j_u_arrays = @jimport java.util.Arrays

JavaObject{:java.util.Arrays} (constructor with 2 methods)

julia> jcall(j_u_arrays, "binarySearch", jint, (Array{jint,1}, jint),
[10,20,30,40,50,60], 40)

3
```


Fig. 6 : Une opération en utilisant le packages JavaCall [8]

- PyCall (Python) : Ce package permet d'appeler et d'interagir avec Python à partir du langage Julia. Vous pouvez importer des modules Python à partir de Julia, utiliser des fonctions Python, définir des classes Python à partir des méthodes Julia et partager de grandes structures de données entre Julia et Python sans les copier.

```
using PyCall
math = pyimport("math")
math.sin(math.pi / 4) # returns  $\approx 1/\sqrt{2} = 0.70710678...$ 
```

Fig. 7 : Une opération en utilisant le packages PyCall [9]

- **Conda (dépendances Python)** : Ce package permet d'utiliser conda en tant que fournisseur binaire multiplateforme pour Julia pour d'autres packages Julia. Conda est un gestionnaire de packages qui a commencé en tant que gestionnaire de packages binaires pour la distribution Python d'Anaconda, mais il fournit également des packages arbitraires.

```
using Conda
Conda.add("libnetcdf", :my_env)
Conda.add("libnetcdf", "/path/to/directory")
Conda.add("libnetcdf", "/path/to/directory"; channel="anaconda")
```

Fig. 8 : Une opération en utilisant le packages Conda [10]

Et encore plein d'autre package comme DataFrames, StatsBase, Distributions, etc...

Miletus.jl est un des piliers qui compose JuliaFin. Il est souvent utilisé pour définir ou modéliser des contrats financiers ou encore utiliser pour un cadre d'évaluation. En effet les contrats financiers sont modélisés la plupart du temps par des langages comme Haskell ou encore OCaml, puis viens la mise en œuvre des processus d'évaluation sont fait par un autre langage comme C++ ou Java. Miletus s'appuie sur le système de type fort de Julia ainsi elle génère un code d'évaluation efficace, résolvant le problème d'utiliser deux langages différents. Ainsi Miletus permet de faire des contrats complexes avec seulement un assemblage de composant et d'opération élémentaire

Voici un exemple d'utilisation de Miletus.jl avec JuliaFin, cet exemple a pour but de définir et de valoriser un contrat pour l'achat européenne :

```
# Receive an amount of 100 USD
In[2]: x=Receive(100USD)
Out[2]: Amount
        ↳100USD

# Pay is the opposite of Receive
In[3]: x=Pay(100USD)
Out[3]: Give
        ↳Amount
        ↳100USD

# Models are constructed and valued on a generic SingleStock type
In[4]: s=SingleStock()
Out[4]: SingleStock
```

Fig. 9 : Une opération en utilisant le packages Miletus [11]

Ainsi, Julia semble être un langage informatique novateur qui pourrait permettre de résoudre les obstacles actuels, notamment dans le monde de l'intelligence artificielle.

II- Exploitation de Julia : Avantages et inconvénients

Aujourd'hui, la communauté se développe autour de Julia. JuliaCon a lieu chaque année, c'est la conférence annuelle de Julia, où un grand nombre de disciplines informatiques (programmes, algorithmes, compilateurs, optimisations, etc.) et de disciplines scientifiques d'ailleurs, la dynamique des fluides et même l'imagerie cérébrale. Julia commence ainsi à jouer un rôle important dans plusieurs secteurs en raison de ces nombreux avantages.

A. Les avantages du langage Julia

1. Le monde de la Science et Julia : facilitation de la collaboration

Le domaine scientifique semble véritablement embrasser le mouvement open source et le partage du travail scientifique, ce qui en soi est un bouleversement. Le problème est que l'histoire montre que l'organisation de la recherche à l'époque était différente. Dans le passé, le code quittait rarement le laboratoire. Aujourd'hui, grâce à des langages comme Julia par exemple, un algorithme scientifique peut être passé d'un chercheur à un autre, tout comme un développeur accède au code d'un autre développeur sur Github.

La recherche scientifique est désormais ouverte à tous et progresse ensemble. De nombreux scientifiques semblent convenir que Julia peut facilement réaliser une collaboration scientifique et une réutilisation du code. Ces scientifiques ont tous découvert que Julia augmente les possibilités de collaboration, rendant l'intégration du travail de chacun plus facile que jamais et leur permettant d'écrire du code pouvant être utilisé de manière inattendue. En raison de son interopérabilité et de son ouverture, Julia nous permet d'emprunter des idées à d'autres scientifiques et de les appliquer à ses propres recherches. Cela facilite la collaboration sur des questions spécifiques, favorisant ainsi le progrès scientifique général.

2. Julia et Python : problème d'expression

Julia est souvent comparée à Python en effet les deux visent à être facile à comprendre et facile d'utilisation, les points où il se ressemble sont les tuples, les variables dynamiques et des fonction imbriquées. Mais ce qui nous intéressent est leurs différences :

- Python utilise des accolades pour dénoter la hiérarchie des blocs alors que Julia utilise des end, c'est plus dur de différencier et de savoir quel end fini quel bloc
- Python est interprété, alors que Julia joue un rôle d'interpréteur qui se sert comme compilateur JIT de LLVM
- Julia se sert de fonction alors que Python lui se sert de def, def étant moins précis que fonction
- Homoiconicité, un programme de Julia est utilisé comme une représentation de données. Ainsi le programme peut se manipuler lui-même, s'étendre, voir se transformer. Ou bien encore crée un autre programme et l'exécuter, qui peut être assez pratique dans le domaine de la robotique

- On peut utiliser les bibliothèques écrites en C ou en Python directement avec Julia
- Julia a été conçu afin de faire un traitement distribué, des fonctions exécutées en parallèle. Ainsi on peut exécuter des traitements avec le même programme sur cloud

Voici un exemple du résultat d'un test basic afin de fusionner des opérations diffusées :

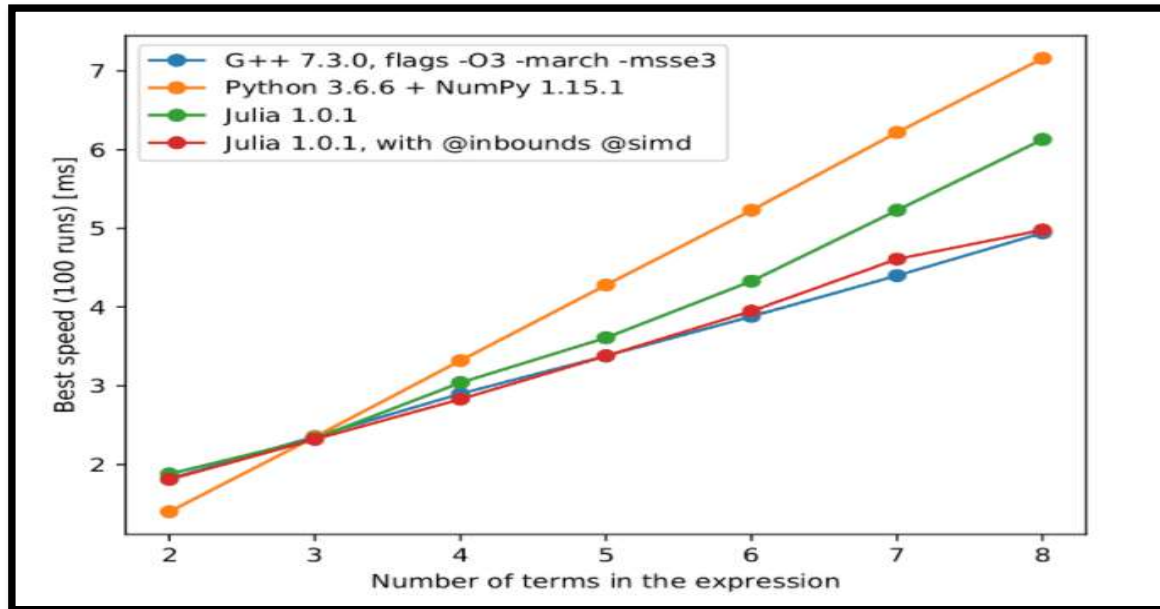


Fig. 10 : Résultat de la vitesse de résolution des opérations diffusée [12]

On peut voir que pour $n(2)$ python est le meilleur et puis le temps passe moins il est performant que Julia a presque la même vitesse de calcul que du C++. La puissance de Julia réside dans la solution du langage à un problème ancien bien connu en informatique : le problème d'expression. En comparant la création d'un ordinateur avec une recette de cuisine, Lee Phillips illustre bien la source du problème. Il a expliqué qu'il y a deux façons d'organiser un livre de cuisine.

Un livre de recettes peut être organisé autour des procédures de préparation des plats, mais il peut également se concentrer sur les ingrédients utilisés pour réaliser ces plats. Par analogie, il fait directement référence aux deux types de langages informatiques qui divisent le monde de la programmation informatique. Les recettes organisées autour du processus sont écrites dans un langage fonctionnel, tandis que les recettes organisées autour des ingrédients sont écrites dans un langage orienté objet.

De plus, Julia a un aspect compilé à la volée, ce qui permet d'avoir des meilleures performances par rapport à du Python ou encore du R classique. En terme de nombre de packages qui peuvent être utilisés, Julia est clairement en dessous de Python et de R. De plus, on peut remarquer que Julia intéresse de plus en plus de monde récemment comme nous montre ce graphe :



Fig. 11 : Graphe qui montre la recherche fait pour Julia sur Google [13]

3. Julia et machine learning

En raison de ses capacités de calcul, Julia est évolutive et plus rapide que Python et R. Si l'on travaille sur des données volumineuses dans un système distribué, il peut être déployé rapidement pour de grands clusters. Julia s'embles donc important pour les professionnels du Machine Learning (ML) travaillant dans la modélisation prédictive, les statistiques, la visualisation et l'apprentissage automatique.

Sans aucun doute, Julia alimente les outils d'analyse de nombreuses organisations de premier plan. Les plus grandes entreprises du monde, dont plusieurs entreprises bien connues utilisent le langage Julia depuis de nombreuses années. Parmi eux, le groupe multinational d'assurance Aviva, qui utilise Julia pour les calculs de risque. BlackRock, une autre société multinationale de gestion d'actifs, a parlé de sa réforme des retraites plus tôt cette année et utilise également les calculs de séries chronologiques de Julia pour les investissements. La

Federal Reserve Bank de New York (l'une des banques les plus célèbres des États-Unis) utilise également le langage Julia pour la modélisation économique.

De nombreuses autres entreprises opérant dans le domaine du big data utilisent également ce langage très utile (visualisation, data science, machine learning, calcul parallèle, etc.). Il y a aussi d'autre grande entreprise comme Amazon, Apple, Disney, Facebook, Google, IBM, Microsoft, Oracle ou encore NASA qui utilisent Julia et embauchent des spécialistes du ML avec son savoir-faire. Il y a aussi des universités qui enseignent Julia comme l'université de Nantes à L'ENSTA, ParisTech à l'Université Lyon I. Apprendre à Julia à programmer des applications d'IA est productif car :

- Il est mieux armé pour aborder le Deep Learning (après Python).
- Il a été spécialement conçu pour implémenter rapidement des mathématiques de bases et des requêtes scientifiques.

Tout dans Julia a été conçu pour les professionnels du ML, de sorte qu'il est centré sur une analyse numérique haute performance. Avec la base d'apprentissage automatique de Python, Julia est simple comme Python, et possède en même temps les pouvoirs de manipulation de Big Data de Spark. Pour les data scientists qui débutent leur carrière, cette caractéristique de Julia doit être prise en compte.

B. Inconvénients et Julia

Il existe néanmoins plusieurs inconvénients de ce langage qui peuvent pour certains se résoudre rapidement et sont surtout la conséquence de la jeunesse du langage.

En effet, Julia est un langage jeune et nouveau ce qui conduit à avoir une syntaxe en constante évolution. Néanmoins, les scientifiques supposent qu'elle serait assez complète pour ne plus être dans cette modification continue.

Associée également à cette jeunesse, le peu de paquets présents comparés à R ou Python semble être aussi un inconvénient, même s'il semblerait qu'il puisse utiliser la bibliothèque d'autres langages.

Un autre inconvénient qui est la conséquence de la jeunesse de ce langage est la communauté en cours de création. Contrairement à Python où la communauté est en belle croissance, Julia est assez jeune pour avoir des utilisateurs aussi importants.

En dehors de cet aspect jeune du langage, un deuxième inconvénient notable est l'objectif de ce langage. En effet, ce langage de programmation a été mis en place spécifiquement pour les opérations scientifiques et pour le machine learning. Et la concurrence reste rude avec Python. Même si Julia le surpasse en termes de vitesse, et qu'il est facile et pratique dans son utilisation, Python reste un excellent langage. Le choix entre ces deux langages s'effectue donc selon nos objectifs.

Julia est ainsi un langage assez jeune composé de peu d'inconvénients, mais assez pour rester en concurrence et derrière les piliers du marché.

Conclusion

Pour conclure Julia est une belle surprise via ses nombreuses possibilités de langage. Julia a toutes les fonctionnalités de Python voir plus, le code est simple à écrire. De plus, il a beaucoup de bibliothèque similaire dans le domaine scientifique, il utiliserait même des bibliothèques de Python. Cependant, ce n'est encore que le début de Julia. En effet Julia n'a qu'une dizaine d'années et n'est pas très connue encore même si elle a un énorme potentiel. Cela nous emmène à une question intéressante faut-il apprendre Julia ? Julia est-elle réellement un nouveau langage qui pourrait surpasser le géant Python ? Ce sont des questions qui seront sûrement étudiées avec plus de profondeurs dans les années à venir.

Bibliographie

- [1] <https://www.silkhom.com/julia-le-langage-de-programmation-qui-accelere-la-recherche-scientifique/>
- [2] <https://www.tiobe.com/tiobe-index/julia/>
- [3] <https://github.com/JuliaMath/Calculus.jl>
- [4] <https://github.com/malmaud/TensorFlow.jl>
- [5] <https://github.com/FluxML/Metalhead.jl>
- [6] <https://github.com/JuliaDatabases/JDBC.jl>
- [7] <https://github.com/JuliaInterop/RCall.jl>
- [8] <https://github.com/JuliaInterop/JavaCall.jl>
- [9] <https://github.com/JuliaPy/PyCall.jl>
- [10] <https://github.com/JuliaPy/Conda.jl>
- [11] <https://www.juliabloggers.com/introduction-to-the-suite-of-juliafin-packages-part-1-miletus-jl/>
- [12] <https://discourse.julialang.org/t/comparing-python-julia-and-c/17019/9>
- [13] <https://www.stat4decision.com/fr/langage-julia-replace-r-python/>
- [14] <https://juliacomputing.com/products/juliapro/>
- [15] <https://julialang.univ-nantes.fr/julianantes/documents/>
- [16] [https://fr.wikipedia.org/wiki/Julia_\(langage\)](https://fr.wikipedia.org/wiki/Julia_(langage))
- [17] <https://1lib.fr/book/3400272/dafdfd>
- [18] <https://1lib.fr/book/5296615/e373d6>
- [19] <https://1lib.fr/book/5304067/551f7a>
- [20] <https://blog.usejournal.com/why-julia-is-beneficial-for-machine-learning-13332fce2bf6>
- [21] <https://www.scriptol.fr/programmation/julia.php>