動的プログラミング言語 Julia の紹介

my recommendations on research & development tools.

佐藤 純一[†], 鷲沢 嘉一^{††}

キーワード: Julia, プログラミング言語, 科学技術計算, 数値計算

1. まえがき

Julia はフリーの科学技術計算用言語として近年注目されている。活発なオープンソースコミュニティを持ち、ライブラリーが盛んに開発されており、この先、科学技術計算における主要言語に位置付けられる可能性が高いと考えている。本稿では基本的な文法や使い方を紹介していく。Julia は公式にとっても充実したドキュメント¹⁾があるため、興味を持った方は詳細をそちらで確認していただきたい。

2. プログラミング言語 Julia

行列計算などの数値計算を高速に行うプログラムを作成するには、C/C++ を用いることが一つの方法であるが、C/C++ は学習コストが高く、バグを作り込みやすいため扱いが難しい。科学技術計算にはそのフィールドに適したプログラミング言語を用いると、開発時間を短縮することができる。Julia は科学技術計算のために設計されており、簡単に数値計算プログラムを作成することができる。また LLVM (Low Level Virtual Machine) による JIT (just in time) コンパイルにより非常に高速に動作する。科学技術計算には、シンプルでアルゴリズムの実装が行いやすい MATLAB、統計解析に強い R、汎用言語の Python (NumPy, SciPy) などがすでに存在している。それにもかかわらず Julia を開発した理由を開発者は、「われわれは貪欲で、C 言語の速度、MATLAB のような数式になじんだ記述、Python のような汎用性を併せ持つ言語を必要としたから」と述べている²⁾。

Julia は公開されたのが 2012 年と、新しい言語である。執筆時における Julia の最新版は v0.5.0 であるため、本稿のサンプルコードでは v0.5.0 で動作させている。他のバージョンでの動作は注意が必要である。

3. 起動と実行

Julia をインストール [web 原稿参照] した後に、コマンドラインで julia とコマンドを実行すると、対話環境の

図1 Juliaの起動

REPL (read-eval-print-loop) が開かれる。REPL はインタラクティブにコードを実行することが可能で、高度な関数電卓としての利用や、開発の際、動作確認などの役に立つ。実行を終えるには Ctrl-D を押すか quit () と入力する。例えば、 2^{20} を計算したい場合は

```
julia> 2^20
1048576
```

とすると結果が表示される。文末に「;」をつけると結果を表示しない。直前に評価した結果は変数 ans に格納される。

```
julia> ans
1048576
```

Julia の REPL は julia > が表示される通常の Julian プロンプトの他に、Help モード、Shell モード、Search モードがある。「?」を入力することで Help モードに入り、関数やマクロ、型などの情報を得ることができる。「;」を入力すると Shell モードに入り、システムのシェルコマンドを実行することが可能である。「Ctrl-R」は前方検索、「Ctrl-S」で後方検索を行う Search モードに入る。

もちろんコードをテキストファイルに保存して、スクリプトファイルとして実行することもできる。拡張子は .jl が使われる。ファイルを実行するには julia 起動時にファイル名を指定する。

```
$ julia file_name.jl
```

REPL 上でソースファイルの読み込み、実行をしたいときは include を用いて

```
julia> include ("file_name.jl")
```

とする。

[†] 電気通信大学 大学院情報理工学研究科 情報・通信工学専攻

^{††} 電気通信大学 大学院情報理工学研究科 情報・ネットワーク工学専攻
"An Introduction of Julia language" by Junichi Sato and Yoshikazu Washizawa (Graduate School of Informatics and Engineering, the University of Electro-Communications, Tokyo)

4. 文 法

4.1 変数

Julia は動的型付言語であり、値を変数に格納する際に型を宣言する必要がない。例えば、

```
julia> x = 1
```

とおくと整数型の変数となる。変数の型は関数 `typeof` で確認することができる。

```
julia> typeof(x)
```

```
Int64
```

Int64 は 64 ビットの整数型である。浮動小数点、文字列、Bool、タプル、連想配列、集合など通常のプログラミング言語に備えられている型などはもちろん備えられている。数値計算用の言語であるため複素数型、有理数型などはデフォルトで利用できる。

はコメントを表す

```
julia> typeof(1 + im) # 虚数 im を使うと複素数型となる
```

```
Complex{Int64}
```

```
julia> typeof(1//3) # 有理数は // 演算子で表すことができる
```

```
Rational{Int64}
```

また数値計算によく使う定数がグローバルで定義されている。

```
julia> e
```

```
e = 2.7182818284590...
```

```
julia> pi
```

```
π = 3.1415926535897...
```

変数名は unicode 文字を使うことができる。バックスラッシュと LaTeX のシンボル名を入力し Tab を押すと LaTeX シンボルを入力することができる。

```
julia> \delta [tab]
```

```
julia> δ
```

4.2 行列とベクトル

Julia では行列、ベクトルを簡単に作成することができる。

例えば、2x2 の行列

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

を作成するには、次のように入力する。

```
julia> A = [1 2; 3 4]
```

2x1 の縦ベクトル

$$b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

を作成するには

```
julia> b = [5, 6]
```

とする。転置は `'` で求められ、例えば $c = b^T A b$ は

```
julia> c = b' * A * b
```

となる。このとき、 c は要素 1 のベクトル (1 次元配列) となる。 c をスカラーにするには内積 `dot` を用いて、

```
julia> c = dot(b, A*b)
```

```
B = rand(10, 10)
for i = 1:10, j = 1:10
    if B[i, j] < 0.5
        B[i, j] = 0
    end
end
```

図2 サンプルプログラム 1

と書く。

このように Julia では行列やベクトルを簡単に扱うことができる。これは MATLAB の記法とほぼ同じであり、もともと MATLAB に慣れ親しんでいた方には扱いやすいかもしれない。

MATLAB や Python などの数値計算のプログラミングでは、for や if などの制御構文をベクトルや行列の演算に置き換えるのがコツである。これによりコードが簡潔になり、高速化も行える。Julia でも同様の変換を行うことが可能である。例えば、乱数行列の要素が 0.5 より小さいときはその要素を 0 にする処理は、素直に制御構文を用いて書くと図 2 となる。

For 文で使われている `1:10` は Range オブジェクトでループ処理やベクトルや行列の範囲指定に用いられる。例えば、`X[1:2, 3:4]` とすると X の 1 行から 2 行、3 列から 4 列までの部分行列を取出す。この for と if は以下のコードに書き換えることができる。

```
B[B.<0.5]=0
```

$B.<0.5$ で要素が真偽を持つ行列が返され、真となる要素に 0 を代入している。

4.3 関数

Julia では以下のように関数を定義する。

```
function f(x, y)
```

```
    2x + y
```

```
end
```

関数の最後に評価された値を返り値とするため、`return` はこの場合書く必要がないが、明示することもできる。また係数と変数の積 `*` を数式のように省略することができる。単純な関数であれば数式のように

```
f(x, y) = 2x + y
```

と、よりシンプルに書くこともでき、上述した LaTeX シンボルを使うとほとんど数式と同じ表記で関数を定義することが可能である。

Julia は多重ディスパッチを採用しており、引数の型を指定して関数を定義すると、型によって異なる処理を行う “メソッド” を定義できる。例えば、型の引数を Int64 と Float64 と変え、異なる処理を定義すると、

```
g(x::Int64, y::Int64) = x + y
```

```
g(x::Float64, y::Float64) = x - y
```

```
julia> g(1, 1)
```

```
2
```

```
function bubble()
    N = 1000
    a = rand(N)
    for i = 1:N-1, j = N:-1:i+1
        if a[j] < a[j-1]
            a[j], a[j-1] = a[j-1], a[j]
        end
    end
end

for n = 1:100
    bubble()
end
```

図3 サンプルプログラム2 バブルソートJulia版

```
import numpy as np

def bubble():
    N = 1000
    a = np.random.rand(N)
    for i in range(N-1):
        for j in range(N-1, i, -1):
            if a[j] < a[j-1]:
                a[j], a[j-1] = a[j-1], a[j]

for n in range(100):
    bubble()
```

図4 サンプルプログラム3 バブルソートPython版

```
using PyPlot

x = linspace(0, 2pi, 1000)
f(x) = sin(x) + cos(x)

plot(x, f(x))
title("\$f(x) = \sin(x) + \cos(x)\$")
xlabel("\$x\$")
ylabel("\$f(x)\$")
show()
```

図5 サンプルプログラム4

```
julia > g(1.0, 1.0)
0.0
```

のように異なる結果となる。関数に割当てられているメソッドの型とその定義箇所を調べるにはmethods関数を用いる。

```
julia > methods(g)
# 2 methods for generic function "g":
g(x::Float64, y::Float64) at REPL[2]:1
g(x::Int64, y::Int64) at REPL[1]:1
```

例えば、演算子+は関数+()を呼びが、methods(+)を実行すると+に多くのmethodが定義されていることがわかる。

5. 数値計算プログラム

科学技術計算のプログラムでは線形代数の計算を多用するが、そのバックエンドにBLAS (Basic Linear Algebra Subprograms) やLAPACK (Liner Algebra Package) を用いることが多い。BLASは行列計算などの基本的な線形代数計算、LAPACKは連立方程式や固有値問題などを解くための関数を提供している。これらはさまざまな実装が存在しており、FortranやC言語で作成されている。MATLABなどのインタプリタ言語でも高速な計算が可能なのは、遅いインタプリタのループは使用せず、BLAS/LAPACKに処理を行わせているからである。Juliaは基本的な線形代数計算はデフォルトでOpenBLASを用いており、その動作は高速である。

アルゴリズムをテクニックによって行列演算に落とし込める場合は計算の高速化が可能だが、forループなどの反復文を書かなくてはならない場合は処理が遅くなってしまふ。このような場合に、MATLABでは処理の一部をC言語(Mex)で記述することで、ボトルネックの解消を行うこともできるが開発コストが高くなってしまふ。PythonではCythonを用いることで、直接Cを書かずにPythonコードをCコードに変換することが可能だが、それでも手軽さは失われる。

一方でJuliaは単にforループを記述しても十分に高速である。この点が従来のMATLABなどの言語に対して優れている。

forループの速度を比較するためにバブルソートを繰り返すプログラムで実行時間を比較した。Linux(64ビット)上で実行したところ、Python(3.5.1 | Anaconda 2.5.0)では実行終了まで17.6秒かかったが、Julia(0.5.0)は0.35秒で実行が終了した。

Juliaが高速な理由は、LLVMによるJITコンパイラを用いており、実行時コンパイルを行っているためである。そのため、コンパイルを終えるまでに多少の時間がかかるが、それが終了した後は高速となる。公式Webページのベンチマーク³⁾ではCやFortranなどの静的型付言語と同等の速度が出ている。

6. ライブラリー

6.1 Pythonとの連携

Juliaはさまざまなアルゴリズムや便利な機能を持つパッケージが数多く公開されている⁴⁾。先に述べた通り、Juliaは新しい言語であるため、まだPythonほど多種多様なパッケージはない。だが、PyCall⁵⁾を用いることでPythonのモジュールや関数を呼び出すことができ、Juliaの適用範囲を広げることができる。Juliaでパッケージを追加するにはREPL上で

```
julia > Pkg.add("PyCall")
```

のように入力する。例えば、NumPyを使用したいときは、

図6 PyPlotによるプロットの例

```
label, size
1, 3.2
2, 1.2
2, 1.8
1, 2.1
```

図7 test.csv

```
julia > using PyCall
```

```
julia > @pyimport numpy as np
```

と入力する。JuliaのREPL上から自由にNumPyの変数や関数を扱うことができるようになる。

6.2 プロット

グラフをプロットしたい場合には、PyCallからPythonのmatplotlibを読み込んで使えば良いが、プロットだけしたい場合にはPyPlot⁶⁾を用いると良い。他にもコンソール上で可視化したい時に便利なTextPlot⁷⁾、Rのggplot2のように美しい描画が可能なGadfly⁸⁾などがある。

6.3 データフレーム

Rのデータフレームのような表形式のデータ構造はDataFrames⁹⁾で利用できる。これを用いるとcsvファイルからデータの読取り、データフレームの集約、集計などができる。例えば、図7のcsvファイルが与えられたとき、labelごとにsizeの平均を出すには図8のように書く。

7. むすび

本稿では、科学技術計算のためのプログラミング言語Juliaを紹介した。Juliaは新しい言語であるため、言語仕様の更新が頻繁で、ライブラリーの数が少ないなど、現時

```
using DataFrames
# csv ファイル読み込み
df = readtable("test.csv")
# labelで集約, meanを適用
df_m = aggregate(df, :label, mean)
```

図8 サンプルプログラム5

```
julia> df
4×2 DataFrames.DataFrame
  Row  label  size
----
  1     1    3.2
  2     2    1.2
  3     2    1.8
  4     1    2.1

julia> df_m
2×2 DataFrames.DataFrame
  Row  label  size_mean
----
  1     1    2.65
  2     2    1.5

julia> 
```

図9 データフレームの表示

点では業務での利用は難しいかもしれない。だが、シンプルな文法を持ち高速に動作するため、アルゴリズムの作成や研究、教育目的の利用に向いていると言える。

(2016年9月30日受付)

〔文 献〕

- 1) <http://docs.julialang.org/>
- 2) <http://julialang.org/blog/2012/02/why-we-created-julia>
- 3) <http://julialang.org/benchmarks>
- 4) <http://pkg.julialang.org/>
- 5) <https://github.com/JuliaPy/PyCall.jl>
- 6) <https://github.com/JuliaPy/PyPlot.jl>
- 7) <https://github.com/sunetos/TextPlots.jl>
- 8) <https://github.com/dcjonas/Gadfly.jl>
- 9) <https://github.com/JuliaStats/DataFrames.jl>

佐藤 純一 2015年、電気通信大学卒業。現在、同大学院修士課程在学中。脳コンピュータインタフェースに関する研究に従事。

鷲沢 嘉一 2008年、東京工業大学大学院博士課程修了。(独)理化学研究所脳科学総合研究センター研究員を経て、現在、電気通信大学大学院情報理工学研究科准教授。パターン識別、脳信号処理の研究に従事。