Gradus Vol 3, No 2 (2016) 49-54
ISSN 2064-8014

A JULIA PROGRAMOZÁSI NYELVRŐL

ABOUT JULIA PROGRAMMING LANGUAGE

Csizmás Edit [1*]

[1]Informatika Tanszék, GAMF Műszaki és Informatikai Kar, Pallasz Athéné Egyetem, Magyarország

Kulcsszavak:
Julia
programozási nyelv
operációkutatás
tudományos számítások

Keywords:
Julia
programming language
operations research
scientific computing

Cikktörténet:
Beérkezett 2016. szeptember 21
Átdolgozva 2016. október 27.
Elfogadva 2016. november 2.

Összefoglalás

A Julia programozási nyelvet az MIT-n fejlesztették ki, az első változata 2012-ben jelent meg. Ez az új, ingyenes, nyílt forráskódú programozási nyelv hatékonyan használható többek között lineáris programozási feladatok megoldására is.

Abstract

Julia programming language was developed on MIT and the first version appeared in 2012. This new, free and open source language can be used effectively for solving linear programming problems.

1. Bevezetés

A Julia ingyenes és nyílt forráskódú új programozási nyelvet 2009-ben kezdték el fejleszteni az MIT munkatársai. Az első megjelenés 2012 februárjában volt. Ezt a magas szintű, nagyteljesítményű dinamikus nyelvet műszaki számításokra fejlesztették ki, azzal a céllal, hogy olyan szintaxissal rendelkezzen, amely megszokott a más műszaki számítási környezetek felhasználói számára is. A Julia szolgáltatásként egy bonyolult fordítót, elosztott párhuzamos végrehajtást, numerikus precizitást és egy átfogó matematikai függvény könyvtárat bocsájt a felhasználó rendelkezésére. A Julia Alapkönyvtár, amely nagyrészt magában a Juliában íródott, integrálja a kifejlett, legjobbfajta nyílt forráskódú C és Fortran könyvtárakat a lineáris algebra, véletlenszám generálás, jelfeldolgozás és szövegfeldolgozás számára. Ezen kívül a Julia fejlesztői közösség is számos külső csomaggal járul hozzá ahhoz, hogy a Julia beépített csomagkezelője gyors ütemben fejlődjön. Az IJulia, egy olyan együttműködés az IPython és a Julia közösségek között, amely egy hatékony böngésző alapú grafikus notebook felületet szolgáltat a Juliához. [1,2]

A nyelv megalkotói, Jeff Bezanson, Stefan Karpinski, Viral B. Shah és Alan Edelman a *Why We Created Julia* című írásukban a nyelv létrehozásának céljairól így írnak:

„Egy nyelvet akartunk, amely nyílt forráskódú, szabad licenccel. Akartuk a C sebességét a Ruby dinamizmusával. Akartunk egy nyelvet, amely homoikonikus[†], olyan megbízható makrókkal, mint a Lisp, de olyan világos, megszokott matematikai jelölésmóddal, mint a Matlab. Akartunk valamit, ami úgy használható általános programozásra, mint a Python, olyan egyszerű a statisztikák számára, mint az R, olyan természetes, a szövegfeldolgozásra, mint a Perl, olyan hatékony a lineáris algebra számára, mint a Matlab, olyan jó a programok összeillesztésében, mint a shell. Valami olyat akartunk, amit nagyon egyszerű megtanulni, mégis megtartja a legkomolyabb programozók boldogságát. Azt akartuk, hogy legyen interaktív és azt akartuk, hogy legyen összeszerkesztett." [4]

* Kapcsolattartó szerző. Tel.: +36 76 516 412; fax: +36 76 516 399
E-mail cím: csizmas.edit@gamf.kefo.hu

† Azok a nyelvek, amelyekben a programkód és az adat nem különül el egymástól. Ilyen nyelv például a Lisp.

A Julia nyelv JuliaOpt programcsomagja hatékonyan használható az operációkutatásban is. Jelen cikk egyrészt a szakirodalomban található hatékonyságra vonatkozó eredményeket ismerteti, másrészt saját méréseket, amelyek kapcsán a sudoku feladat Juliában, illetve Matlabban implementált megoldások futási idői kerültek összehasonlításra.

2. A nagy teljesítményű Julia

A Julia nyelvet az alapoktól kezdve úgy tervezték, hogy kihasználják a dinamikus nyelvek modern technikáit. Ennek eredményeként a Julia egy statikusan fordított nyelv teljesítményével rendelkezik, miközben interaktív dinamikus viselkedést nyújt, és olyan produktív, mint a Python, a Lisp vagy a Ruby. A teljesítmény fő alkotórészei a következők: [5]

- Gazdag típus információ, amelyet a többszörös polimorfizmus (multiple dispatch) tesz lehetővé;
- Agresszív kód specializáció, a futásidejű típussal szemben;
- Just-in-time (JIT) fordítás, felhasználva az LLVM fordító keretrendszert;

A Julia LLVM alapú JIT fordítója és a nyelv kialakítása lehetővé teszi azt, hogy a teljesítménye megközelíti és gyakran el is éri a C teljesítményét. Összehasonlításként más nyelvi megvalósításokkal, a szerzők közölnek egy táblázatot, melyben a C nyelvbeli futási időkhöz hasonlítják a többi nyelvbeli futási időket. [1]

	Fortran gcc 5.1.1	Julia 0.4.0	Python 3.4.3	R 3.2.2	Matlab R2015b	Octave 4.0.0	Mathematica 10.2.0	JavaScript V8 3.28.71.19	Go go1.5	LuaJIT gsl-shell 2.3.1	Java 1.8.0_45
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36	1.86	1.71	1.21
parse_int	5.05	1.45	17.02	45.73	802.52	9581.44	15.02	6.06	1.20	5.77	3.35
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70	1.29	2.03	2.60
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66	1.11	0.67	1.35
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01	1.00	1.00	1.00
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30	2.96	3.27	3.92
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07	1.42	1.16	2.36

1. ábra. Benchmark idők a C-hez viszonyítva, a kisebb a jobb, a C teljesítménye = 1.0 (az ábra forrása [1])

A Julia számos lehetőséget nyújt különböző párhuzamos számításokra, mellyel tovább növelhető a futási teljesítmény. Az osztott memória programozása Juliában két primitívre épül, ezek a távoli hívások és a távoli referenciák, melyek Julia nyelven lettek implementálva. Ezen kívül rendelkezésre áll az osztott tömb, a párhuzamos térkép (parallel map) és párhuzamos for ciklus. [6]

3. Julia használata az operációkutatásban

Optimalizálási feladatok megoldásához a JuliaOpt programcsomagok gyűjteménye áll rendelkezésünkre. Ahhoz hogy használni tudjuk, először frissítenünk kell a Julia alkalmazásunkat, utána a programcsomagok telepítése következik: [3,7]

```
julia> Pkg.update()
julia> Pkg.add("Optim")
julia> Pkg.add("JuMP")
julia> Pkg.add("Cbc")
```

Majd a programcsomagokat használatba kell vennünk:

```
using JuMP, Cbc
```

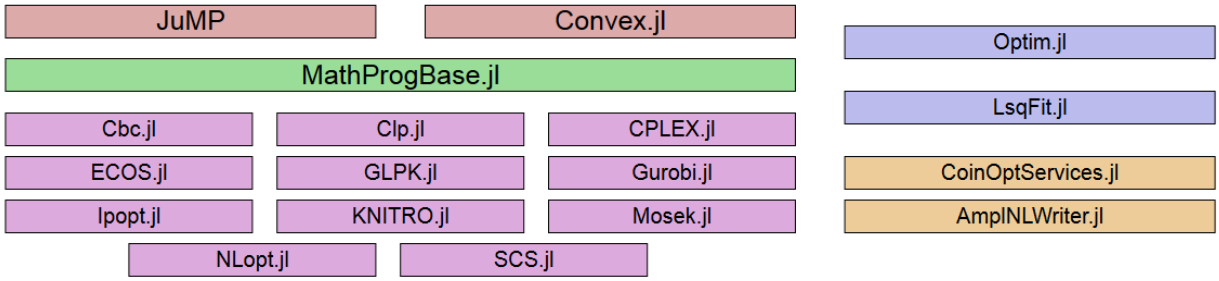
A 2. ábra a JuliaOpt programcsomagjait mutatja be. A JuliaOpt programcsomagjait két csoportra oszthatjuk. Az első csoportba tartoznak az önálló programcsomagok, ezek az Optim.jl és a LsqFit.jl. A második csoportba tartoznak a modellező nyelvek (JuMP, Convex.jl), a külső solver interfészek (Cbc.jl, Clp.jl, CPLEX.jl, stb., a lilával jelölt programcsomagok) és egy a megoldók fölötti

absztrakciós réteg (MathProgBase.jl). Az egyes programcsomagok részletes leírása megtalálható a JuliaOpt hivatalos honlapján. [9]

A következő példában a JuMP modellező és a CPLEX külső megoldó programcsomagok segítségével egy egyszerű lineáris programozási feladatot oldunk meg. A [11] lineáris programozási bevezető könyv 2.9-es feladata a következő:

$$
\begin{aligned}
-2\,x2 - 3\,x3 &>= -5 \\
x1 + x2 + 2\,x3 &<= 4 \\
x1 + 2\,x2 + 3\,x3 &<= 7 \\
x1 &>= 0 \\
x2 &>= 0 \\
x3 &>= 0 \\
Max \quad 2\,x1 + 3\,x2 + 4\,x3 &
\end{aligned}
$$

A feladat megoldásához először létre kell hoznunk egy modellt, amelyben megadjuk a változókat a korlátozásaikkal (`@variable()`), majd a feltételeket meghatározó egyenleteket,



2. ábra. A JuliaOpt programcsomagjai (az ábra forrása [9])

egyenlőtlenségeket (`@constraint()`), illetve a célfüggvényt (`@objective()`). A modell megalkotása után valamelyik megoldóval (`solve()`) megoldjuk a feladatot. A feladat megoldásának implementációja Julia nyelven a következő: [10]

```
using JuMP, CPLEX

m = Model()
@variable(m, x1 >= 0 )
@variable(m, x2 >= 0 )
@variable(m, x3 >= 0 )

@objective(m, Max, 2x1 + 3x2 + 4x3 )
@constraint(m, -2x2 - 3x3 >= -5.0 )
@constraint(m, 1x1 + 1x2 + 2x3 <= 4.0 )
@constraint(m, 1x1 + 2x2 + 3x3 <= 7.0 )

print(m)

status = solve(m)

println("Objective value: ", getobjectivevalue(m))
println("x1 = ", getvalue(x1))
println("x2 = ", getvalue(x2))
println("x3 = ", getvalue(x3))
```

Amit észrevehetünk az implementációban az, hogy a feladatot nem szükséges mátrixokkal megfogalmazni, mint például a Matlabban, hanem az egyenlőtlenségeket a feladatban megadott módon írhatjuk be a modellbe.

Egy másik érdekes példa: a Sudoku megoldása JuMP segítségével, melyet a programcsomag egyik megalkotója, Iain Dunning ismertet. A példa megoldása megtalálható JuliaOpt példái között [12].

Egy sudoku feladvány tulajdonképpen nem optimalizálási probléma, hanem felfoghatjuk úgy, mint egy lehetséges megoldás keresési feladatot. Ha a 9x9x9-es háromdimenziós változónkat, úgy vesszük fel, hogy az $x(i,j,k)$ változó értéke akkor és csak akkor *1*, ha az *i,j*-edik cella értéke *k*, minden más esetben *0*, akkor lineáris egészértékű programozási feladatként is megoldhatjuk, ahol a célfüggvény *0*:

A feltételek:

$$\sum_{j\in\{1,..,9\}} x_{i,j,k} = 1 \qquad \forall i \in \{1,\dots,9\}, k \in \{1,\dots,9\}$$

$$\sum_{i\in\{1,..,9\}} x_{i,j,k} = 1 \qquad \forall j \in \{1,\dots,9\}, k \in \{1,\dots,9\}$$

$$\sum_{k\in\{1,..,9\}} x_{i,j,k} = 1 \qquad \forall i \in \{1,\dots,9\}, j \in \{1,\dots,9\}$$

$$\sum_{a\in\{1,2,3\},b\in\{1,2,3\}} x_{(3i+a),(3j+b),k} = 1 \qquad \forall i \in \{1,2,3\}, j \in \{1,2,3\}, k \in \{1,\dots,9\}$$

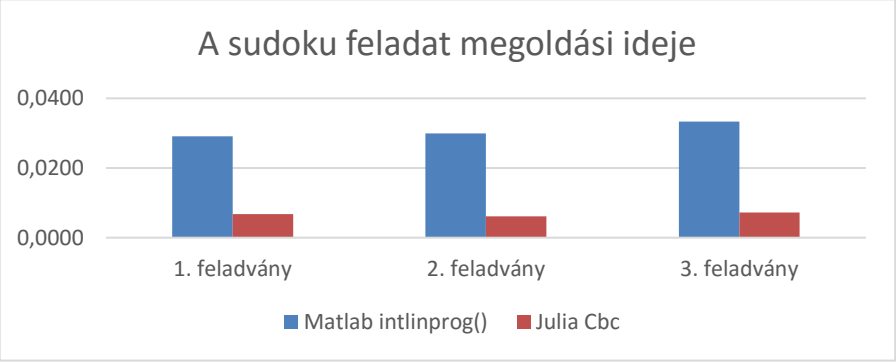
Az implementációban a Cbc megoldót használjuk. Először definiáljuk a változót bináris változóként, majd a fenti feltételeket adjuk meg for ciklusok segítségével. A kezdő feladat mátrixa segítségével megadjuk az ismert *x*-eket. A solver meghívása (`solve()`) után az eredménymátrixot elkészítjük, majd kiíratjuk az eredményt.

A sudoku feladat megoldására a Matlab dokumentációjában is találhatunk egy példát, amelyben az intlinprog() függvénnyel történik az egészértékű programozási feladat megoldása. [13]

A Matlab dokumentációjában található egy másik sudoku megoldó példaprogram is, amelyben

1. táblázat. A sudoku feladat megoldása

A megoldás módja	*1. feladvány*	*2. feladvány*	*3. feladvány*
Matlab intlinprog()	0,0291	0,0300	0,0333
Julia Cbc	0,0067	0,0062	0,0072
Matlab opt nélkül	0,0438	0,1053	16,2937

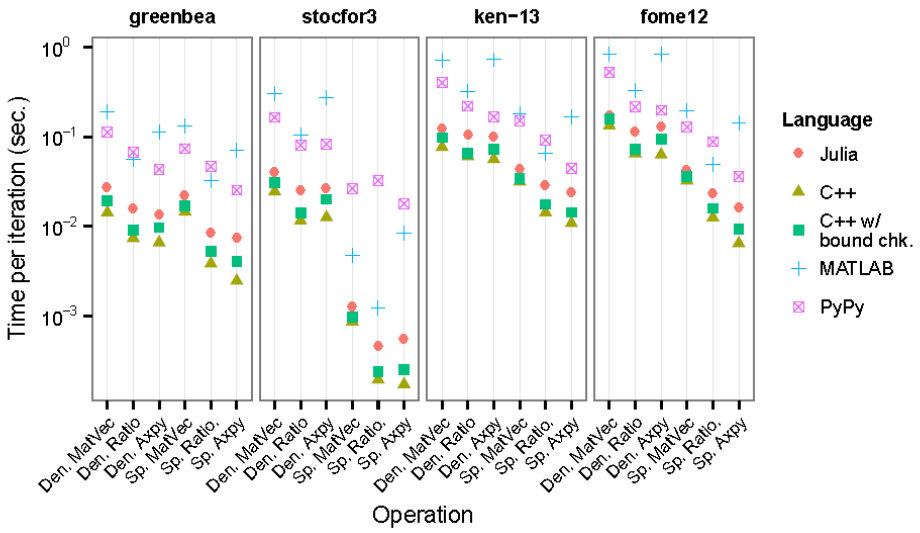


3. ábra. A sudoku 3 nehézségi fokozatú feladat megoldásának futási idői a Julia és a Matlab programozási nyelvek esetében

az optimalizálás felhasználása nélkül történik a megoldás. [14]

A három program (Julia-ban JuMP modellezővel, Matlab intlinprog() függvénnyel és Matlab optimalizálás nélkül) futásidejét hasonlítottuk össze. Az első két esetben a megoldó függvény futásidejét mértük le (ami nem lényegesen tér el a teljes program futási idejétől), a harmadik esetben pedig a teljes futási időt. Mindhárom módszerrel három, egyre nehezebb feladványt oldottunk meg. Az optimalizáló nélküli megoldások esetében a futási idők jól mutatják a nehézségi fokozatokat. Az 1. táblázatban találhatók a futási idők, a 3. ábrán pedig a kétféle programozási nyelv megoldóinak futási idejét hasonlítottuk össze.

Miles Lubin és Iain Dunning, a JuMP modellező programcsomag megalkotói a [8] cikkükben különböző optimalizálási módszereket vizsgálnak, 5 féle programozási nyelven implementálva, és ezek futási idejét hasonlítják össze. A 4. ábráról leolvasható, hogy a Julia mindegyik megvalósításban a C++-hoz közeli teljesítményt nyújt.



4. ábra. Átlagos végrehajtási idők. A Matlab és a PyPy nyelvekkel összehasonlítva a Julia futási idői szignifikánsan közelebb vannak a C++-hoz (az ábra forrása [8])

4. Összegzés

A Julia programozási nyelv és a hozzá kapcsolódó keretrendszer igen hatékony eszköznek tűnik többek között optimalizálási feladatok megoldására. A népszerű, de igen drága és sok esetben lassú Matlabbal összevetve lényegesen jobb futási időket lehet elérni. A másik népszerű, de ingyenes R programozási nyelvvel összehasonlítva is úgy tűnik, hogy bizonyos számítási feladatok esetén az R-nél is jobb futási időket tud elérni a Julia.

Irodalomjegyzék

[1] “The Julia Language (official website),” [Online]. Available: http://julialang.org/. [Megtekintés: 30-Aug-2016].
[2] “Julia (programming language),” [Online]. Available: https://en.wikipedia.org/wiki/Julia_%28programming_language%29. [Megtekintés: 30-Aug-2016].
[3] Thomas J. Sargent, John Stachurski. “Quantitative Economics,” [Online]. Available: http://quant-econ.net/jl/getting_started.html. [Megtekintés: 30-Aug-2016].
[4] Jeff Bezanson, Stefan Karpinski, Viral Shah, Alan Edelman. “Why We Created Julia,” [Online]. Available: http://julialang.org/blog/2012/02/why-we-created-julia. [Megtekintés: 30-Aug-2016].
[5] Bezanson, Jeff, et al. “Julia: A fast dynamic language for technical computing,” arXiv preprint arXiv:1209.5145 (2012). [Online]. Available: http://arxiv.org/abs/1209.5145. [Megtekintés: 30-Aug-2016].
[6] Bezanson, Jeff, et al. “Julia: A fresh approach to numerical computing,” arXiv preprint arXiv:1411.1607 (2014). [Online]. Available: http://arxiv.org/abs/1411.1607. [Megtekintés: 30-Aug-2016].
[7] Bogomił Kamiński. “The Julia Express,” 2015. [Online]. Available: http://bogumilkaminski.pl/files/julia_express.pdf. [Megtekintés: 30-Aug-2016].

[8] Lubin, Miles, and Iain Dunning. “Computing in operations research using Julia,” INFORMS Journal on Computing 27.2 (2015): 238-248. [Online]. Available: http://arxiv.org/abs/1312.1431. [Megtekintés: 30-Aug-2016].
[9] “JuliaOpt. Optimization packages for the Julia language,” [Online]. Available: http://www.juliaopt.org/. [Megtekintés: 30-Aug-2016].
[10] “JuliaOpt. Using Julia+JuMP for optimization - getting started,” [Online]. Available: http://www.juliaopt.org/notebooks/Shuvomoy%20-%20Getting%20started%20with%20JuMP.html. [Megtekintés: 30-Aug-2016].
[11] Vanderbei, Robert J. “Linear programming,” Springer, 2008.
[12] Iain Dunning. “Solving Sudoku with JuMP,” [Online]. Available: https://github.com/JuliaOpt/juliaopt-notebooks/blob/master/notebooks/JuMP-Sudoku.ipynb. [Megtekintés: 30-Aug-2016].
[13] “Solve Sudoku Puzzles Via Integer Programming,” [Online]. Available: http://www.mathworks.com/help/optim/ug/solve-sudoku-puzzles-via-integer-programming.html. [Megtekintés: 30-Aug-2016].
[14] Cleve Moler. “Solving Sudoku with MATLAB. MathWorks,” [Online]. Available: http://www.mathworks.com/company/newsletters/articles/solving-sudoku-with-matlab.html. [Megtekintés: 30-Aug-2016].