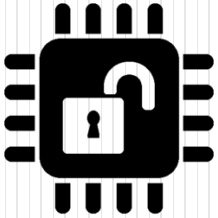
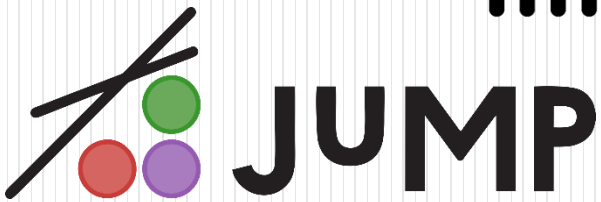
Tópico

Julia Language for Mathematical Programming

Mar 2019





Autores:
Erik Alvarez
Jefferson Chávez

Universidade **E**stadual de **C**ampinas
DSEE – **D**epartamento de **S**istemas de **E**nergia **E**létrica

ASME
SETTING THE STANDARD

UNICAMP

JuliaLang?

- ***Open-source*** & lenguaje de programación libre (MIT license).
  - Desarrollado a partir del *2012* (por: MIT researchers)
  - Creciente popularidad a nivel mundial, en investigación, data science, finanzas, etc.
  - Multi-platform: Windows, Mac OS X, GNU/Linux…
- Diseño para la ***performance:***
  - Interpretador & compilador, ***muy eficiente***
  - Fácil de ***run codes*** en paralelo (multi-core & cluster)
- Diseño para ser ***simple de aprender y usar***:
  - Easy sintax, dynamic typing (como MATLAB & Python)

Referencias

Link: Home · The Julia Language

Link: Introduction · JuMP

Link: GLPK - GNU Project - Free Software Foundation (FSF)

Link: Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT

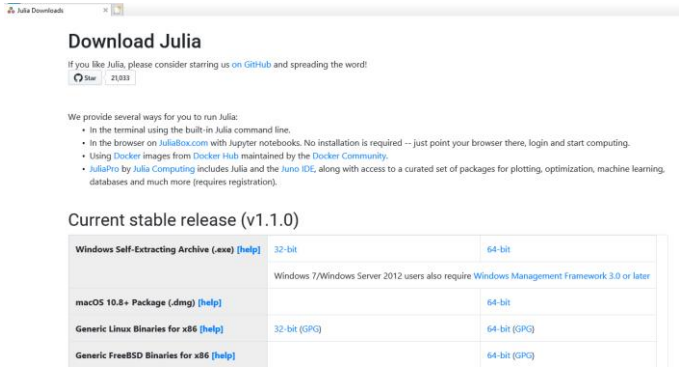
Julia vs. MATLAB: General

	Julia ☺	MATLAB ☹
Cost	Free	Hundreds of €/$ per year
License	Open-source	1 year user license
Comes from	A non-profit foundation, and the community	Mathworks Company
Editor/IDE	***Jupyter*** and ***Atom/Juno*** are recommend	Good IDE already included
Parallel computations	Very esasy, low overhead cost	Possible, high overhead
Usage	Generic, worldwide	Research in academy and industry
Fame	Young but starts to be known	Old and known

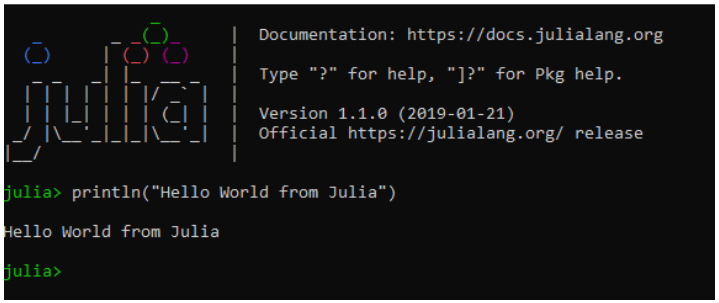
Instalación (1/3)

- Para Linux, Mac OS o Windows.
  - Se puede descargar el instalador JuliaLang en: Julia Downloads
  - Atom se puede descargar de: Atom

1. Descarga e instalar Julia con ***default settings.***



2. Abrir Julia



Instalación (2/3)

1. En Julia: Shift + tecla "}" para establecer el **(v1.1) pkg>**
2. En: **(v1.1) pkg> add JuMP**
3. En: **(v1.1) pkg> add GLPK**
4. En: **(v1.1) pkg> build GLPK**
5. Apretar tecla *"backspace"* : **julia>**
6. En: **julia> using JuMP**
7. En: **julia> using GLPK**
8. En: **julia> using GLPK**
9. En: **(v1.1) pkg> add Ipopt**
10. En: **(v1.1) pkg> build Ipopt**
11. En: **julia> using Ipopt**
12. En: **(v1.1) pkg> status**

Instalación (3/3)

1. Descarga e instalar Julia con ***default settings.***
2. In Atom, ir a ***Settings*** (Ctrl+,, or Cmd+, on macOS) e ir al "Install" panel.

3. Escribir **uber-juno** en la sección de busqueda y apretar tecla **Enter**. Dar ***click*** en el boton ***install*** de la librería.

4. **Atom** instalará y configurará **Juno** por ti.

Descripción general de los famosos “Packages” de Julia

- ***Plotting:***
  - Winston.jl para plotear comoMATLAB
  - PyPlot.jl , interfaz al Matplotlib (Python)
- JuliaDiffEq.jl , para ecuacioanes diferenciales
- JuliaStats.jl , para estadistica
- JuliaDSP , para procesamiento de señales
- BackpropNeuralNet, para redes neuronales
- JuliaOpt/JuMP para optimización

- Más “Packages” en: Julia Package Listing

Julia vs. MATLAB: Syntax

	Julia	MATLAB
File ext.	.jl	.m
Comment	# blabla…	% blabla…
Indexing	a[1] to a[end]	a(1) to a(end)
Slicing	a[1:100]	a(1:100)
Operations	Linear Algebra ***special library***	Linear Algebra ***by default***
Block	Use **end** to close all blocks	Use **endif**, **endfor** etc
And	a & b	a && b
Or	a | b	a || b
Array	[1 2; 3 4]	[1 2; 3 4]

JuMP

- JuMP package: lenguaje de modelamiento para optimización
- Interfaces de JuMP con solver de optimización:

Solver	Julia Package	License	Supports
Artelys Knitro	KNITRO.jl	Comm.	LP, MILP, SOCP, MISOCP, NLP, MINLP
Cbc	Cbc.jl	EPL	MILP
Clp	Clp.jl	EPL	LP
CPLEX	CPLEX.jl	Comm.	LP, MILP, SOCP, MISOCP
CSDP	CSDP.jl	EPL	LP, SDP
ECOS	ECOS.jl	GPL	LP, SOCP
FICO Xpress	Xpress.jl	Comm.	LP, MILP, SOCP, MISOCP
GLPK	GLPK.jl	GPL	LP, MILP
Gurobi	Gurobi.jl	Comm.	LP, MILP, SOCP, MISOCP
Ipopt	Ipopt.jl	EPL	LP, QP, NLP
MOSEK	MosekTools.jl	Comm.	LP, MILP, SOCP, MISOCP, SDP
OSQP	OSQP.jl	Apache	LP, QP
SCS	SCS.jl	MIT	LP, SOCP, SDP
SDPA	SDPA.jl	GPL	LP, SDP
SeDuMi	SeDuMi.jl	GPL	LP, SOCP, SDP

- Donde:
  - LP: Linear Programming
  - QP:Quadratic Programming
  - SOCP = Second-order conic programming (including problems with convex quadratic constraints and/or objective)
  - MILP = Mixed-integer linear programming
  - NLP = Nonlinear programming
  - MINLP = Mixed-integer nonlinear programming
  - SDP = Semidefinite programming
  - MISDP = Mixed-integer semidefinite programming

- Otros leguajes de modelamiento:
  - AMPL
  - GAMS
  - MATLAB:YALMIP,CVX
  - Python: Pyomo, PuLP, CVXPy

JuMP: Comparaciones “syntax” (1/2)

$$\min_{x} \sum_{(i,j)\in E} c_{i,j}\, x_{i,j}$$

s.t.

$$\sum_{(i,j)\in E} x_{i,j} = \sum_{(j,k)\in E} x_{j,k}\,, j = 2, \ldots, n-1,$$

$$\sum_{(i,n)\in E} x_{i,n} = 1,$$

$$0 \le x_{i,j} \le C_{i,j}, \forall (i,j) \in E$$

JuMP: Comparaciones “syntax” (1/2)

JuMP

```
immutable Edge
        from; to; cost; capacity
end
edges = [Edge(1,2,1,0.5), Edge(1,3,2,0.4), Edge(1,4,3,0.6),
         Edge(2,5,2,0.3), Edge(3,5,2,0.6), Edge(4,5,2,0.5)]
Mcf = Model()
@variable(mcf, 0 <= flow[e in edges] <= e.capacity
@constraint(mcf, sum{flow[e], e in edges, e.to==5} == 1)
@constraint(mcf, flowcon[n=2:4], sum{flow[e], e in edges; e.to==node}
                                  == sum{flow[e], e in edges; e.from==node})
@objective(mcf, Min, sum{e.cost * flow[e], e in edges})
```

AMPL

```
set edges := {(1,2),(1,3),(1,4),(2,5),(3,5),(4,5)};
param cost{edges}; param capacity{edges};
data …; # Data es alamacenada separadamente en AMPL;
var flow{(i,j) in edges} >= 0.0, <= capacity[i,j];
subject to unitflow: sum{(i,5) in edges} flow[i,5] == 1;
subject to flowconserve {n in 2..4}:
  sum{(i,n) in edges} flow[i,n] == sum{(n,j) in edges} flow[n,j];
minimize flowcost:    sum{(i,j) in edges} cost[i,j] * flow[i,j];
```

JuMP: Comparaciones “syntax” (2/2)

Pyomo

```
edges = [(1,2), (1,3), (1,4), (2,5), (3,5), (4,5)]
cost = {(1,2):1, (1,3):2, (1,4):3, (2,5):2, (3,5):2, (4,5):2}
capacity = {(1,2):0.5, (1,3):0.4, (1,4):0.6, (2,5):0.3, (3,5):0.6, (4,5):0.5}
mcf = ConcreteModel()
mcf.flow = Var(edges, bounds=lambda m,i,j: (0,capacity[(i,j)]))
mcf.uf = Constraint(expr=sum(mcf.flow[e] for e in edges if e[1]==5) == 1)
def con_rule(mcf,n): return sum(mcf.flow[e] for e in edges if e[1]==n) ==
                            sum(mcf.flow[e] for e in edges if e[0]==n)
mcf.flowcon = Constraint([2,3,4],rule=con_rule)
mcf.flowcost = Objective(expr=sum(cost[e]*mcf.flow[e] for e in edges))
```

GAMS

```
SET nodes /n1*n5/; SET midnodes(nodes) /n2*n4/; SET lastnode(nodes) /n5/;
ALIAS(nodes,nodefrom,nodeto,n);
SET edges(nodes,nodes) / n1.n2 n1.n3 n1.n4 n2.n5 n3.n5 n4.n5 /;
PARAMETER cost(nodes,nodes) / … /; * Data omitted
PARAMETER capacity(nodes,nodes) / … /; * for space reasons
POSITIVE VARIABLE flow(nodefrom,nodeto); flow.UP(edges) = capacity(edges);
EQUATION unitflow;
unitflow.. sum{edges(nodefrom,lastnode), flow(nodefrom,lastnode)} =e= 1;
EQUATION flowcon(nodes);
flowcon(midnodes(n)).. sum{edges(nodefrom,n), flow(nodefrom,n)} =e=
                       sum{edges(n,nodeto), flow(n,nodeto)};
FREE VARIABLE obj;
EQUATION flowcost; flowcost.. obj =e= sum{edges, cost(edges)*flow(edges)};
MODEL mincostflow /all/; SOLVE mincostflow USING lp MINIMIZING obj;
```

JuMP: Pro & Cons (1/2)

	GAMS	AMPL	AIMMS	C++	PYOMO	JuMP
Data Input	hard	hard	✓	✓	✓	✓
Data Manipulation	✗	✗	✓	✓	✓	✓
Modeling	✓	✓	✓	hard	✓	✓
Advanced Algorithms	hard	hard	✓	✓	✓	✓
Solvers Availabilty	✓	✓	limited	✗	limited	limited
Visualization	✗	✗	✓	hard	hard	hard
License	$	$	$$	free	free	free

JuMP: Pro & Cons (2/2)

Pros

- Nuevo lenguaje de modelamiento
- Muy rápido
- Open-source
- Facil y simple de codificar
- Acceso a C/C++
- Respaldado por una amplia comunidad científica
- Amplia variedad de librerías

Cons

- Nuevo lenguaje de modelamiento
- Muchas librerías y plataformas aun en desarrollo
- Poco conocido

JuMP: Quiénes aplican en mercados eléctricos?

Joaquim Dias Garcia

Ministerio de Energía y Minas

The Nature Conservancy

BID
Banco Interamericano de Desarrollo

Northwest Power and Conservation Council