Julia, JuMP, MOA and Mathematics of Interdisciplinary Multiobjective Optimization

Prof. Dr. Xavier Gandibleux

Nantes Université, France

February 2025

Why this aventure with Julia and JuMP?

2015: Kick-off of the ANR/DFG research project vOpt

Sub-task 2.2: Experimental Analysis and Prototype Development.

2017: Introduction of vOptSolver (vOptGeneric, vOptSpecific)

Xavier Gandibleux, Gauthier Soleilhac, Anthony Przybylski, Stefan Ruzika. vOptSolver: an open source software environment for multiobjective mathematical optimization. *IFORS2017: 21st Conference of the International Federation of Operational Research Societies*. July 17-21, 2017. Quebec City (Canada).

Purposes of vOptSolver

Natural, intuitive use for mathematicians, informaticians, engineers

- for research:
  support and primitives for the development of new algorithms
- for experimentation:
  methods and algorithms for performing numerical experiments
- for teaching:
  environment for practicing of theories and algorithms

Easy to formulate a problem, provide the data, solve a problem, collect the outputs, analyze the solutions...

+ Solver: robust, efficient, evolutive
+ Software: free, open source, multi-platform

...and easy to install: no need of being expert in computer science!

Purposes of vOptSolver

Natural, intuitive use for mathematicians, informaticians, engineers

- for research:
  support and primitives for the development of new algorithms
- for experimentation:
  methods and algorithms for performing numerical experiments
- for teaching:
  environment for practicing of theories and algorithms

Easy to formulate a problem, provide the data, solve a problem, collect the outputs, analyze the solutions...

+ Solver: robust, efficient, evolutive
+ Software: free, open source, multi-platform

...and easy to install: no need of being expert in computer science!

Today

Introduction to

1. **Julia**
2. **JuMP**
3. **MultiObjectiveAlgorithms**

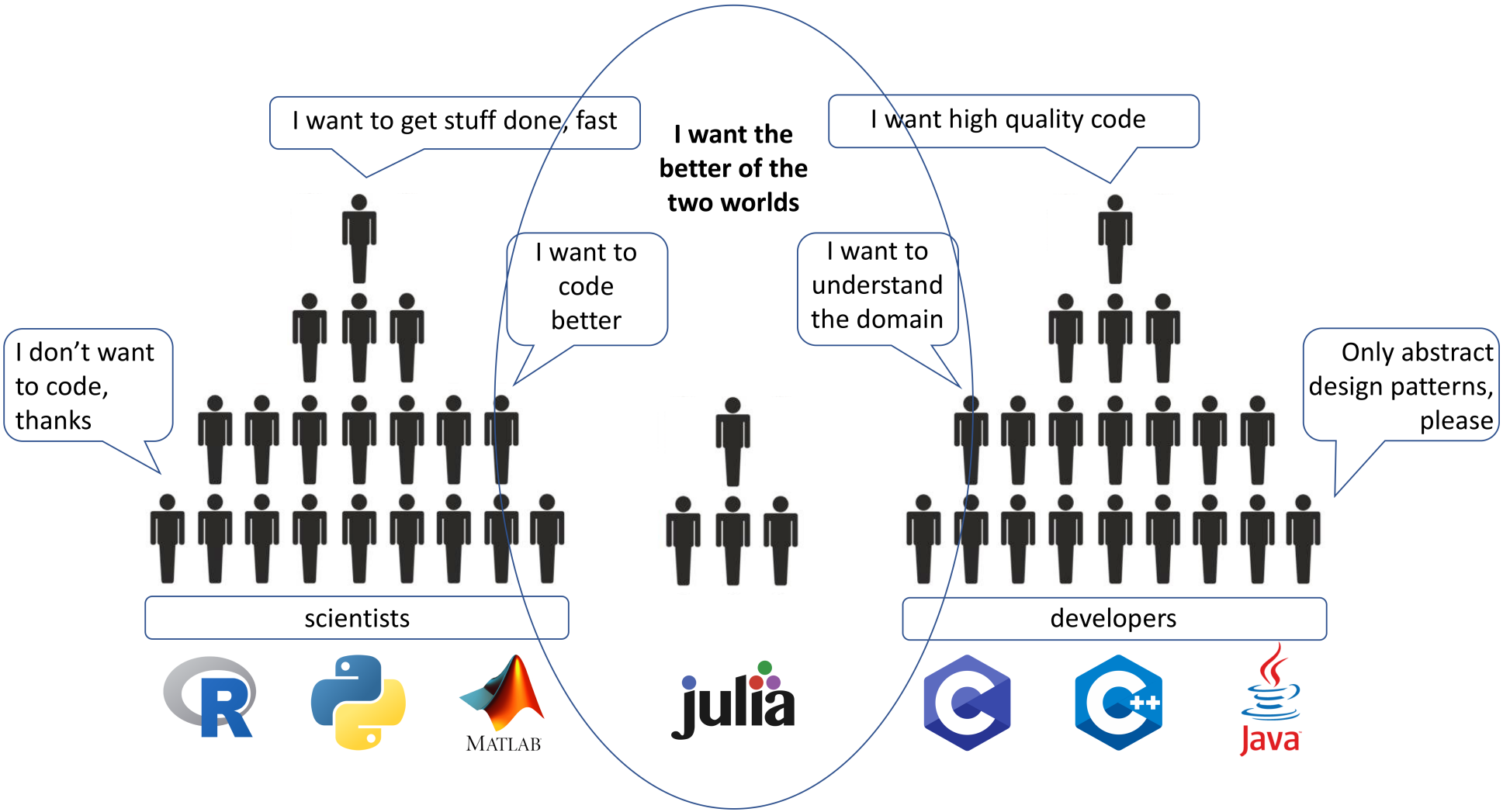
Julia

looks like MATLAB but free and open-source,
easy as Python but fast as C

`https://julialang.org/`

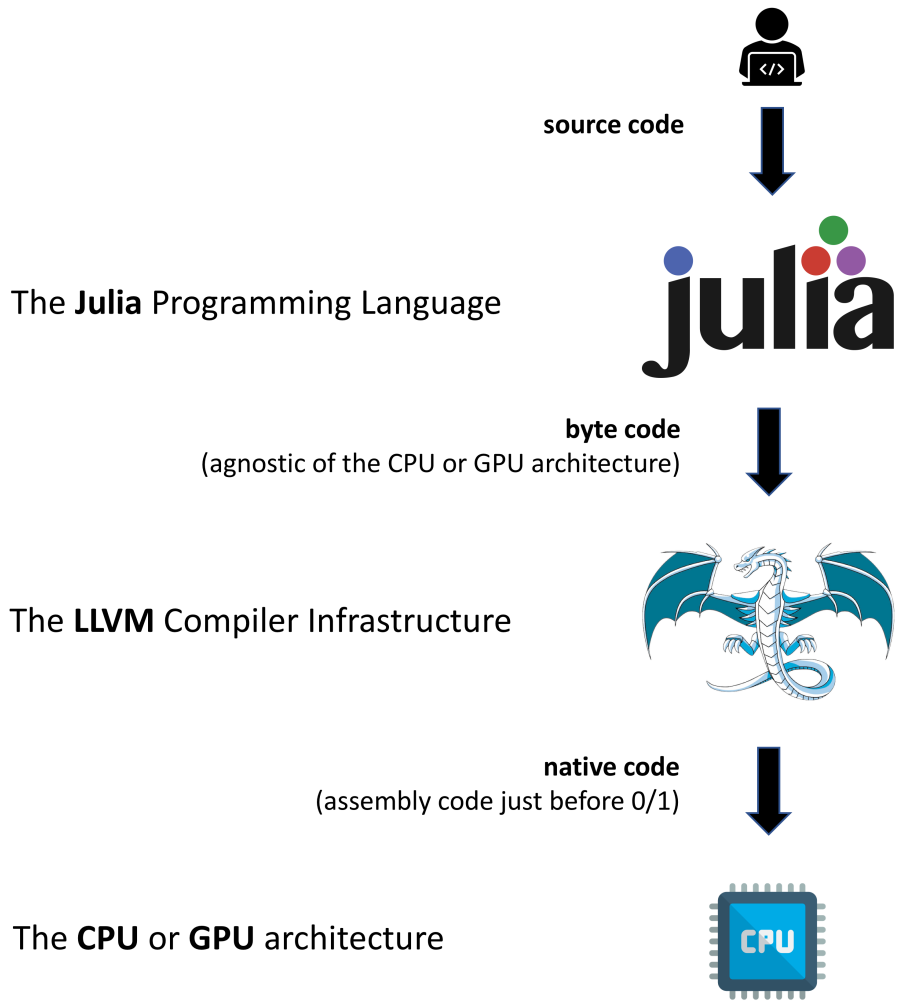
Julia

A programming language for optimization



Julia

LLVM: middleman between source code and compiled native code




```
julia> 1+2
3
```

```
julia> @code_llvm 1+2
;  @ int.jl:87 within `+`
define i64 @"julia_+_115"(i64 signext %0, i64 signext %1) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

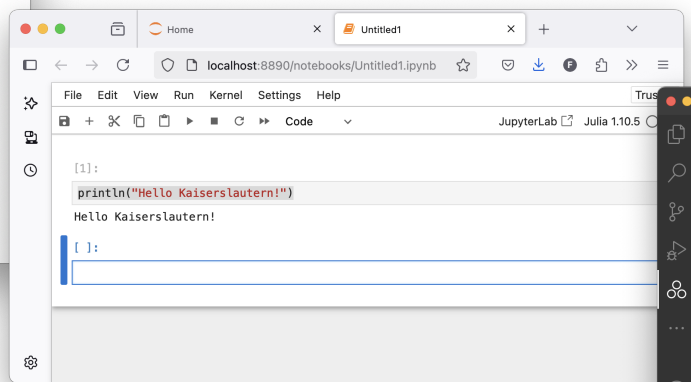
```
julia> @code_native 1+2
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 13, 0
        .globl  "_julia_+_150"                  ; -- Begin function julia_+_150
        .p2align        2
"_julia_+_150":                         ; @"julia_+_150"
; ┌ @ int.jl:87 within `+`
        .cfi_startproc
; %bb.0:                                ; %top
        add     x0, x1, x0
        ret
        .cfi_endproc
; └
                                        ; -- End function
.subsections_via_symbols
```

```
01101100
01101111
01110110
01100101
```

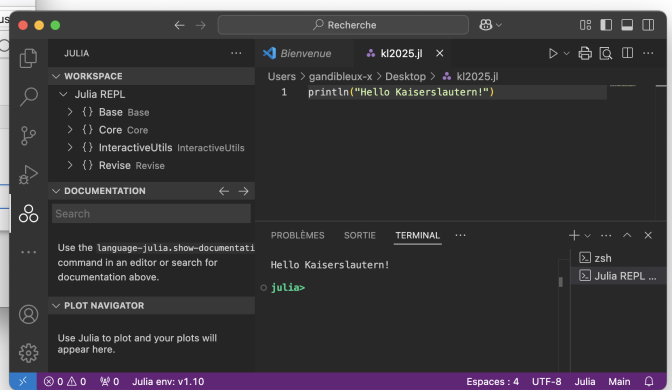
Running a Julia code into...

...the Julia REPL

...a notebook
(e.g. jupyter)



...an IDE
(e.g. VScode)



... or also in command line into a terminal.

Example (1/2)

C

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int A[100][100][100];
    int i,j,k,loop;
    for (loop=0; loop <1000; loop++){
        for (i=0; i<100; i++){
            for (j=0; j<100; j++){
                for (k=0; k<100; k++){
                    A[i][j][k] = loop;
                }
            }
        }
        printf("%d\n", A[99][99][99]);
    }
    return 0;
}
```

MATLAB

```matlab
tic

    A = zeros(100,100,100);
    for loop = 1:1000
        for i = 1:100
            for j = 1:100
                for k = 1:100
                    A(i,j,k) = loop;
                end
            end
        end
        disp(A(100,100,100));
    end

toc
```

Julia

```julia
@time begin

    A = zeros(100,100,100)
    for loop = 1:1000
        for i = 1:100
            for j = 1:100
                for k = 1:100
                    A[i,j,k] = loop
                end
            end
        end
        println(A[100,100,100])
    end

end
```

Example (2/2)

- C

  without optimization option

  `1.286582 seconds`

  with optimization option `-O3`

  `0.087682 seconds`

- Julia

  1-to-1 straightforward implementation

  `26.732819 seconds (489.05 M allocations: 7.297 GiB, 0.29% gc time)`

  julia style compliant implementation

  `0.252995 seconds (22.39 k allocations: 8.499 MiB)`

  polished implementation

  `0.114872 seconds (20.75 k allocations: 4.318 MiB)`

Example (2/2)

- C

  without optimization option

  `1.286582 seconds`

  with optimization option `-O3`

  `0.087682 seconds`

- Julia

  1-to-1 straightforward implementation

  `26.732819 seconds (489.05 M allocations: 7.297 GiB, 0.29% gc time)`

  julia style compliant implementation

  `0.252995 seconds (22.39 k allocations: 8.499 MiB)`

  polished implementation

  `0.114872 seconds (20.75 k allocations: 4.318 MiB)`

Example (2/2)

- C

  without optimization option

  `1.286582 seconds`

  with optimization option `-O3`

  `0.087682 seconds`

- Julia

  1-to-1 straightforward implementation

  `26.732819 seconds (489.05 M allocations: 7.297 GiB, 0.29% gc time)`

  julia style compliant implementation

  `0.252995 seconds (22.39 k allocations: 8.499 MiB)`

  polished implementation

  `0.114872 seconds (20.75 k allocations: 4.318 MiB)`

Example (2/2)

- C

  without optimization option

  `1.286582 seconds`

  with optimization option `-O3`

  `0.087682 seconds`

- Julia

  1-to-1 straightforward implementation

  `26.732819 seconds (489.05 M allocations: 7.297 GiB, 0.29% gc time)`

  julia style compliant implementation

  `0.252995 seconds (22.39 k allocations: 8.499 MiB)`

  polished implementation

  `0.114872 seconds (20.75 k allocations: 4.318 MiB)`

More (1/2)...

Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.

The Julia Programming Language (`https://julialang.org/`):
download, documentation, resources, community, etc.

JuliaCon (`https://juliacon.org/`):
Annual International Conference since 2014 (Chicago, Cambridge, Berkeley, London, Baltimore, online, Eindhoven). 2025: Pittsburgh

JuliaDays:
the Julia & Optimization days in France since 2019 (Nantes, Paris, Toulouse). 2025: Paris (JuliaCon Local)

More (2/2)...

Nice material for mathematicians:

Book

Algorithms for Optimization
Mykel J. Kochenderfer, and Tim A. Wheeler
The MIT Press Cambridge, Massachusetts London, England
`https://algorithmsbook.com/optimization/files/optimization.pdf`

Course

Practical Scientific Computing
Urbain Vaes
École des Ponts (Paris)
`https://jfbarthelemy.github.io/Cours_ENPC_pratique_calcul_scientifique/poly.html`

JuMP

Coding and solving
a mathematical programming model

`JuMP.jl`

What is JuMP?

An Algebraic Modeling Language (AML) among the existing ones . . .

... for mathematical optimization (linear, mixed-integer, conic, semidefinite, nonlinear) written in the Julia language.

Overview of JuMP

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: Currently 56 solvers are supported.

  Among the (MI)LP solvers: GLPK, Cbc, Clp, HiGHS, SCIP, CPLEX, Gurobi, FICO Xpress, COPT...
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Overview of JuMP

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: Currently 56 solvers are supported.

  Among the (MI)LP solvers: GLPK, Cbc, Clp, HiGHS, SCIP, CPLEX, Gurobi, FICO Xpress, COPT...
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Overview of JuMP

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: Currently 56 solvers are supported.

  Among the (MI)LP solvers: GLPK, Cbc, Clp, HiGHS, SCIP, CPLEX, Gurobi, FICO Xpress, COPT...
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Overview of JuMP

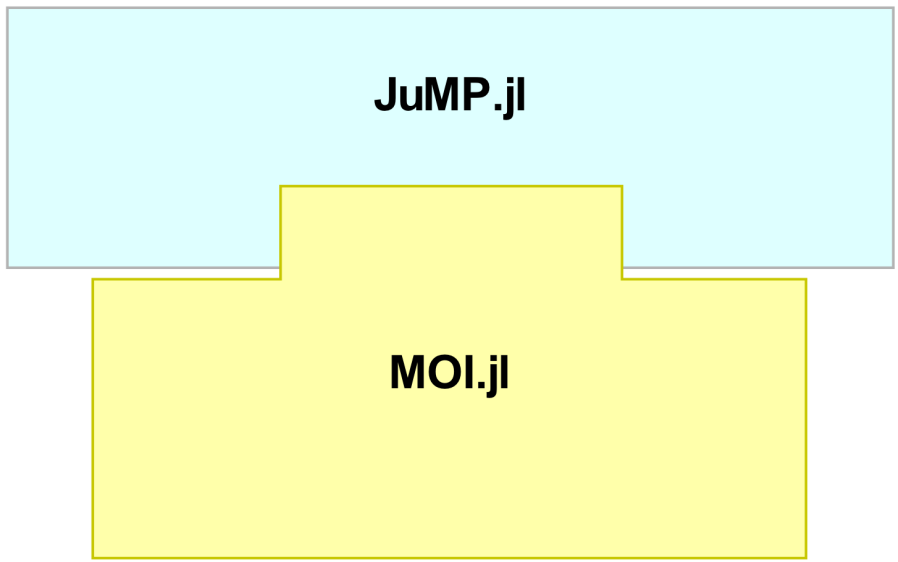
- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: Currently 56 solvers are supported.

  Among the (MI)LP solvers: GLPK, Cbc, Clp, HiGHS, SCIP, CPLEX, Gurobi, FICO Xpress, COPT...
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Getting started with JuMP

Getting started

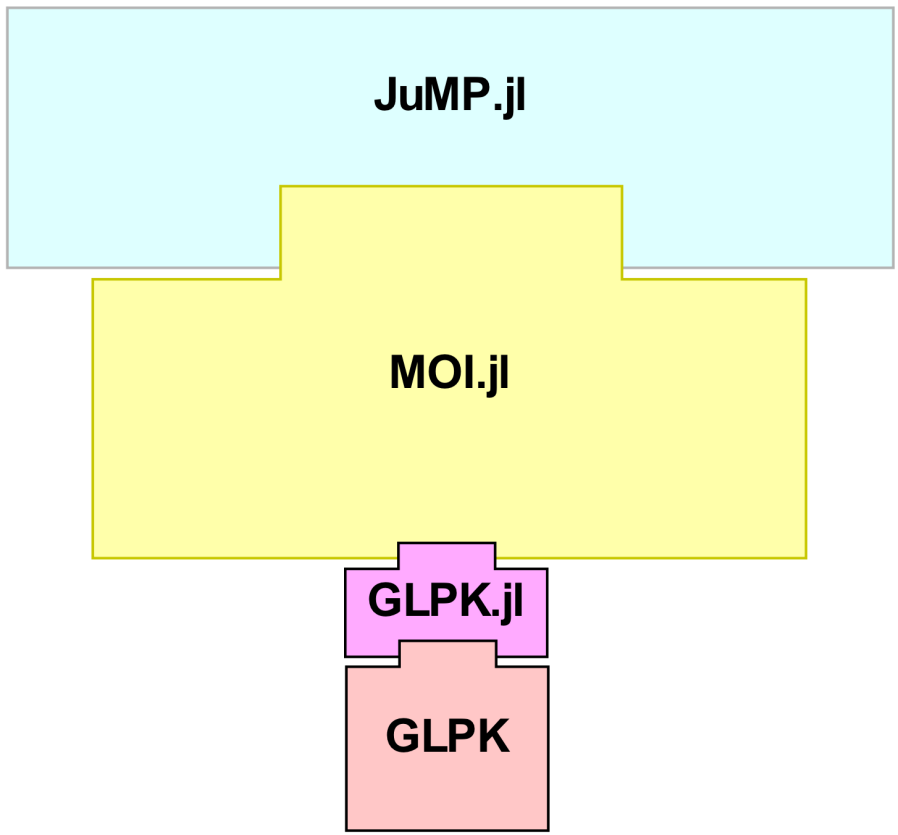
```
julia> using Pkg
julia> Pkg.add("JuMP")
```



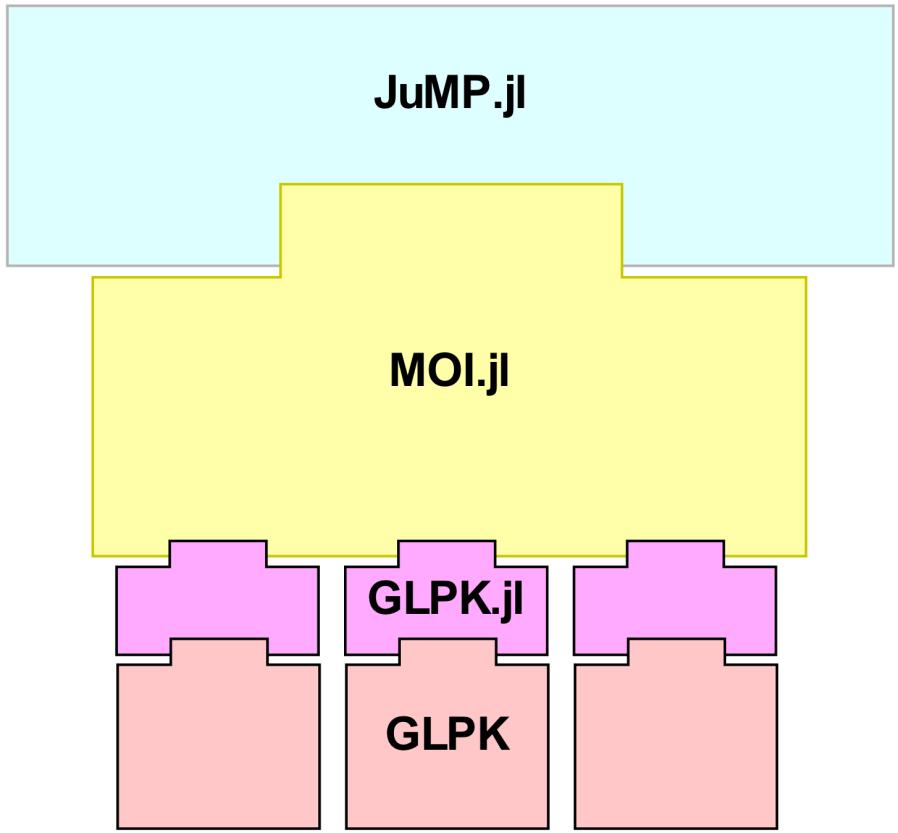
JuMP uses *MathOptInterface (MOI)*, an abstraction layer designed to provide a unified interface to mathematical optimization solvers.

Getting started

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



Getting started



Modeling and solving a (MI)LP with JuMP

The optimization problem

$$
\left[\begin{array}{rrrrrrr}
\max z(x) = & x_1 & + & 3x_2 & & & (0) \\
s.t & x_1 & + & x_2 & \leqslant & 14 & (1) \\
& -2x_1 & + & 3x_2 & \leqslant & 12 & (2) \\
& 2x_1 & - & x_2 & \leqslant & 12 & (3) \\
& x_1 & , & x_2 & \geqslant & 0 & (4)
\end{array}\right]
$$

Declaring the packages to use

```
julia> using JuMP, GLPK
```

Coding the optimization problem (1/4)

Creating a model:

```
model = Model(solver)
```

```
julia> modLP = Model(GLPK.Optimizer)
```

Result:

$$\begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Coding the optimization problem (2/4)

Defining a variable:

```
@variable(model, variable)
```

```
julia> @variable(modLP, x1 >= 0)
julia> @variable(modLP, x2 >= 0)
```

Result:

$$x_1 \quad , \quad x_2 \quad \geqslant \quad 0 \qquad (4)$$

Coding the optimization problem (3/4)

Defining an objective function:

```
@objective(model, sense, function)
```

```
julia> @objective(modLP, Max, x1 + 3x2)
```

Result:

$$\left[\begin{array}{lllll} \max z(x) = & x_1 + 3x_2 & & & (0) \\ & & & & \\ & & & & \\ & x_1 , & x_2 \geqslant & 0 & (4) \end{array}\right]$$

Coding the optimization problem (4/4)

Defining a constraint:

```
@constraint(model, id, constraint)
```

```
julia> @constraint(modLP, cst1, x1 + x2 <= 14)
julia> @constraint(modLP, cst2, -2x1 + 3x2 <= 12)
julia> @constraint(modLP, cst3, 2x1 - x2 <= 12)
```

Result:

$$\left[\begin{array}{rrrrrrr} \max z(x) = & x_1 & + & 3x_2 & & & (0) \\ s.t & x_1 & + & x_2 & \leqslant & 14 & (1) \\ & -2x_1 & + & 3x_2 & \leqslant & 12 & (2) \\ & 2x_1 & - & x_2 & \leqslant & 12 & (3) \\ & x_1 & , & x_2 & \geqslant & 0 & (4) \end{array}\right]$$

Solving the optimization problem

```
optimize!(model)
```

```
julia> optimize!(modLP)
```

Querying the result

```
julia> @show termination_status(modLP)
```

```
julia> @show objective_value(modLP)
```

```
julia> @show value(x1)
julia> @show value(x2)
```

Querying the result

```
julia> @show termination_status(modLP)
```

```
julia> @show objective_value(modLP)
```

```
julia> @show value(x1)
julia> @show value(x2)
```

Querying the result

```
julia> @show termination_status(modLP)
```

```
julia> @show objective_value(modLP)
```

```
julia> @show value(x1)
julia> @show value(x2)
```

Summary

```
julia> using JuMP, GLPK

julia> modLP = Model(GLPK.Optimizer)
julia> @variable(modLP, x1 >= 0)
julia> @variable(modLP, x2 >= 0)
julia> @objective(modLP, Max, x1 + 3x2)
julia> @constraint(modLP, cst1, x1 + x2 <= 14)
julia> @constraint(modLP, cst2, -2x1 + 3x2 <= 12)
julia> @constraint(modLP, cst3, 2x1 - x2 <= 12)

julia> optimize!(modLP)

julia> @show termination_status(modLP)
julia> @show objective_value(modLP)
julia> @show value(x1), value(x2)
```

More...

Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, Juan Pablo Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*. 2023.

Miles Lubin. The state of JuMP. *JuliaCon 2023: The 10th Annual JuliaCon - Co-Located with JuMP-dev*, July 25th-29th, 2023. MIT, Cambridge, (USA).

The JuMP website (`https://jump.dev/`):
documentation, resources, community, etc.

JuMP-dev (`https://jump.dev/meetings/jumpdev2025/`):
regular meeting since 2017 (Massachusetts Institute of Technology, University of Bordeaux, Pontificia Universidad Católica de Chile, online, HEC Montréal, Auckland).

MOA

Moving from
single to multiple objectives MIP optimization

`MultiObjectiveAlgorithms.jl`

From `vOptGeneric` to `MultiObjectiveAlgorithms`

The `vOptSolver` ecosystem (`https://github.com/vOptSolver`) is composed of two packages:

- `vOptGeneric.jl`
- `vOptSpecific.jl`

2015: Kick-off of vOpt ANR/DFG project

2017: Introduction of vOptSolver (vOptGeneric, vOptSpecific)

2018: Julia 1.0

2019: End of vOpt project

2019: First discussions on discourse/new design of MOO in JuMP

2022: JuMP 1.0

2023: Introduction of `MultiObjectiveAlgorithms.jl`

2023: End of vOptGeneric

- part concerning the MO model → `JuMP.jl`
- part concerning MO algorithms → `MultiObjectiveAlgorithms.jl`

Multiobjective Optimization Problem (MOP) notations

$$\begin{array}{ll} \min & F(x) \\ s.t. & x \in X \end{array}$$

where:

$$\begin{array}{rcl} X \subseteq \mathbb{R}^n & \longrightarrow & \text{the set of feasible solutions} \\ Y = F(X) \subseteq \mathbb{R}^p & \longrightarrow & \text{the set of images} \end{array}$$

Computing sets Y_N and X_E for MOP:

$$y = F(x)$$

$y^* \in Y$ is nondominated, if $\nexists\, y \in Y$ such that $y_i \leqslant y_i^*,\ \forall i$ and $y \neq y^*$.

Y_N is the set of nondominated points.

$x^* \in X$ is efficient if y^* is nondominated.

X_E is a complete set of efficient solutions.

Existing MP Software for MOP

ADBASE	MOLP	(1975)
BENSOLVE (with GLPK)	MOLP	(2014; last rev 2017)
polySCIP (with SCIP)	MOLP, MOIP	(2016; last rev 2017)
inner (with GLPK)	MOLP	(2016?; last rev 2024)
JuMP, MOA, solvers	MOIP	(2017, last rev 2025)

Gurobi Optimization	(since Ver7.0, 2016)
IBM ILOG CPLEX Optimisation	(since Ver12.7, 2016)
FICO Xpress Optimization	(since Ver9.0, 2022)
Hexaly Optimizer	(since Ver12.0, 2023)

Existing MP Software for MOP

ADBASE	MOLP	(1975)
BENSOLVE (with GLPK)	MOLP	(2014; last rev 2017)
polySCIP (with SCIP)	MOLP, MOIP	(2016; last rev 2017)
inner (with GLPK)	MOLP	(2016?; last rev 2024)
JuMP, MOA, solvers	MOIP	(2017, last rev 2025)

Gurobi Optimization	(since Ver7.0, 2016)
IBM ILOG CPLEX Optimisation	(since Ver12.7, 2016)
FICO Xpress Optimization	(since Ver9.0, 2022)
Hexaly Optimizer	(since Ver12.0, 2023)

Getting Started (con't)

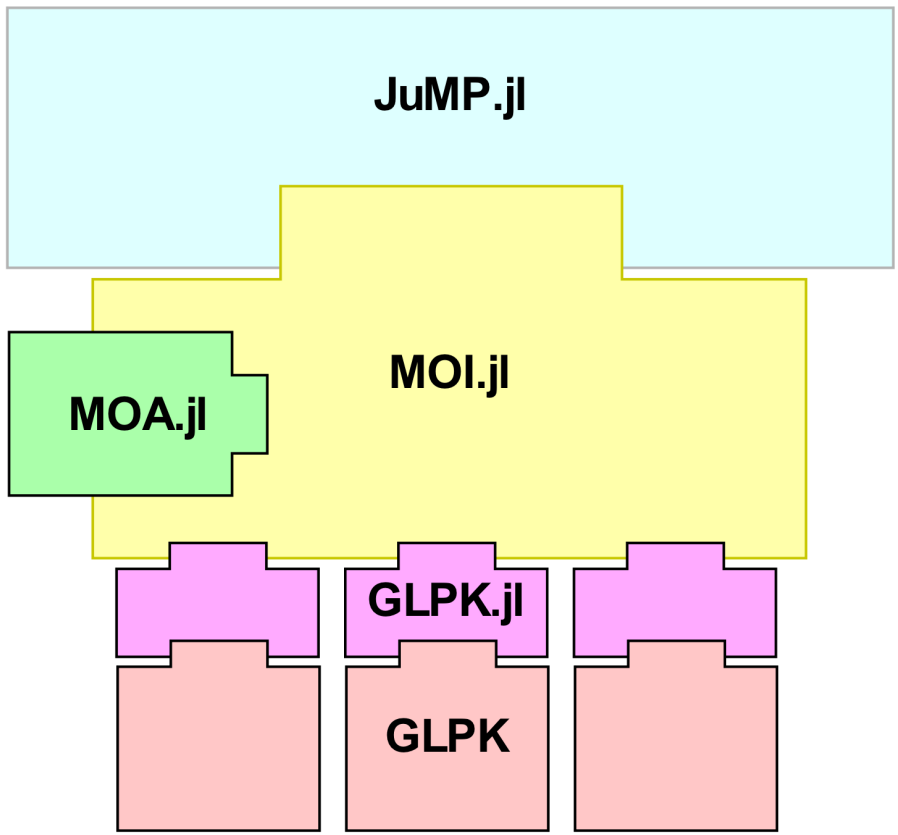
Installing the package:

```
julia> using Pkg
```

```
julia> Pkg.status()
Status `~/.julia/environments/v1.10/Project.toml`
  [60bf3e95] GLPK v1.2.1
  [4076af6c] JuMP v1.23.5
```

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

Getting Started (con't)



Algorithms currently available (v1.3.3)

1. MOA.Lexicographic() [default] $p \geqslant 2$
2. MOA.Dichotomy() $p = 2$
3. MOA.EpsilonConstraint() $p = 2$
4. MOA.Hierarchical() $p \geqslant 2$
5. MOA.Chalmet() $p = 2$
6. MOA.KirlikSayin() $p \geqslant 3$
7. MOA.DominguezRios() $p \geqslant 3$
8. MOA.TambyVanderpooten() $p \geqslant 3$

+ optimization attributes coming with a given algorithm

Algorithms currently available (v1.3.3)

1. MOA.Lexicographic() [default] $p \geqslant 2$
2. MOA.Dichotomy() $p = 2$
3. MOA.EpsilonConstraint() $p = 2$
4. MOA.Hierarchical() $p \geqslant 2$
5. MOA.Chalmet() $p = 2$
6. MOA.KirlikSayin() $p \geqslant 3$
7. MOA.DominguezRios() $p \geqslant 3$
8. MOA.TambyVanderpooten() $p \geqslant 3$

+ optimization attributes coming with a given algorithm

Algorithms currently available (v1.3.3)

1. MOA.Lexicographic() [default] $p \geqslant 2$
2. MOA.Dichotomy() $p = 2$
3. MOA.EpsilonConstraint() $p = 2$
4. MOA.Hierarchical() $p \geqslant 2$
5. MOA.Chalmet() $p = 2$
6. MOA.KirlikSayin() $p \geqslant 3$
7. MOA.DominguezRios() $p \geqslant 3$
8. MOA.TambyVanderpooten() $p \geqslant 3$

+ optimization attributes coming with a given algorithm

Algorithms currently available (v1.3.3)

1. MOA.Lexicographic() [default] $p \geqslant 2$
2. MOA.Dichotomy() $p = 2$
3. MOA.EpsilonConstraint() $p = 2$
4. MOA.Hierarchical() $p \geqslant 2$
5. MOA.Chalmet() $p = 2$
6. MOA.KirlikSayin() $p \geqslant 3$
7. MOA.DominguezRios() $p \geqslant 3$
8. MOA.TambyVanderpooten() $p \geqslant 3$

+ optimization attributes coming with a given algorithm

Example

Algorithm:

- `EpsilonConstraint()`

  Y.V. Haimes, L.S. Lasdon, D.A. Wismer (1971). On a bicriterion formation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man and Cybernetics*, Volume SMC-1, Issue 3, Pages 296-297.

Attributs:

- `MOA.EpsilonConstraintStep()`

  algorithm uses this value as the ϵ by which it partitions the first-objective's space. The default is 1, so that for a pure integer program this algorithm will enumerate Y_N.

- `MOA.SolutionLimit()`

  if this attribute is set then, instead of using the `MOA.EpsilonConstraintStep`, with a slight abuse of notation, `EpsilonConstraint` divides the range of the first-objective's domain in objective space by `SolutionLimit` to obtain the ϵ to use when iterating. Thus, at most `SolutionLimit` solutions are returned.

Example (1/5)

Compute X_E and Y_N for the following MOP:

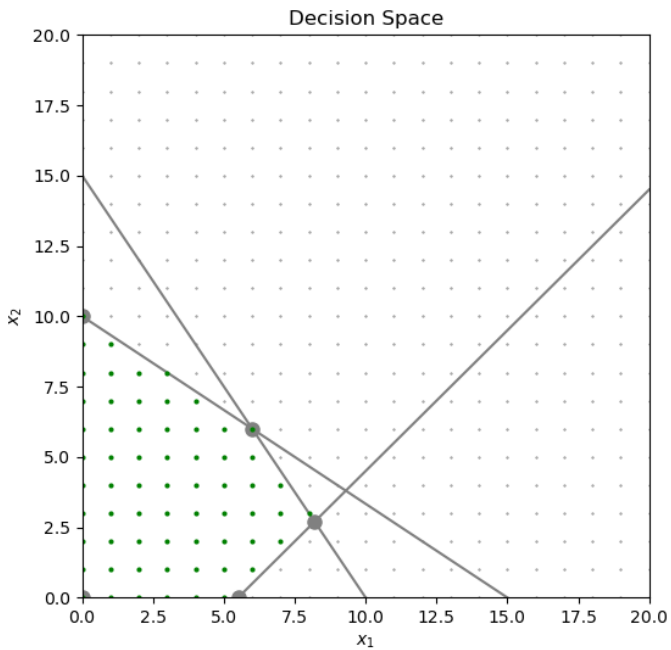
$$
\begin{array}{lcrcrcr}
\max z_1 & = & x_1 & + & x_2 & & \\
\min z_2 & = & x_1 & + & 3x_2 & & \\
s.t. & & 2x_1 & + & 3x_2 & \leqslant & 30 \\
 & & 3x_1 & + & 2x_2 & \leqslant & 30 \\
 & & x_1 & - & x_2 & \leqslant & 5.5 \\
 & & x_1 & , & x_2 & \in & \mathbb{N}
\end{array}
$$

Example (2/5)

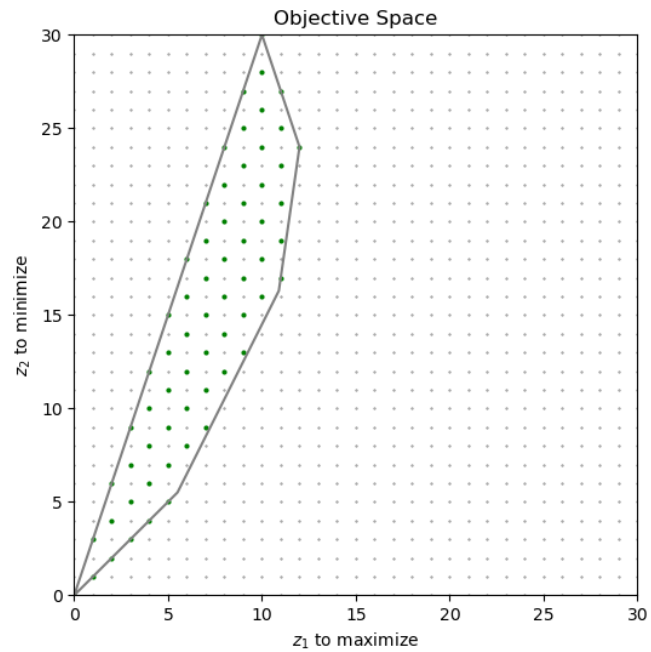
$$
\begin{array}{lcrcrcr}
\max z_1 & = & x_1 & + & x_2 & & \\
\min z_2 & = & x_1 & + & 3x_2 & & \\
s.t. & & 2x_1 & + & 3x_2 & \leqslant & 30 \\
 & & 3x_1 & + & 2x_2 & \leqslant & 30 \\
 & & x_1 & - & x_2 & \leqslant & 5.5 \\
 & & x_1 & , & x_2 & \in & \mathbb{N}
\end{array}
$$

```
julia> c1 = [1,1]
julia> c2 = [1,3]
julia> A = [2 3; 3 2; 1 -1]
julia> b = [30, 30, 5.5]
```

Decision space:



Objective space:



Example (3/5)

Coding the MOO problem with JuMP

```
julia> using JuMP, GLPK
julia> import MultiObjectiveAlgorithms as MOA
```

```
julia> model = Model( )
julia> @variable(model, x1⩾0, Int)
julia> @variable(model, x2⩾0, Int)
julia> @expression(model, fct1, x1 + x2)                    # to maximize
julia> @expression(model, fct2, x1 + 3 * x2)                # to minimize
julia> @objective(model, Max, [fct1, (-1) * fct2] ))
julia> @constraint(model, 2*x1 + 3*x2 ⩽ 30))
julia> @constraint(model, 3*x1 + 2*x2 ⩽ 30))
julia> @constraint(model, x1 - x2 ⩽ 5.5))
```

Example (4/5)

Setup the MIP solver: e.g. `GLPK`

```
julia> set_optimizer(model,()->MOA.Optimizer(GLPK.Optimizer))
```

Setup the algorithm: ϵ-constraint; step=1 (default value)

```
julia> set_attribute(model, MOA.Algorithm(), MOA.EpsilonConstraint())
```

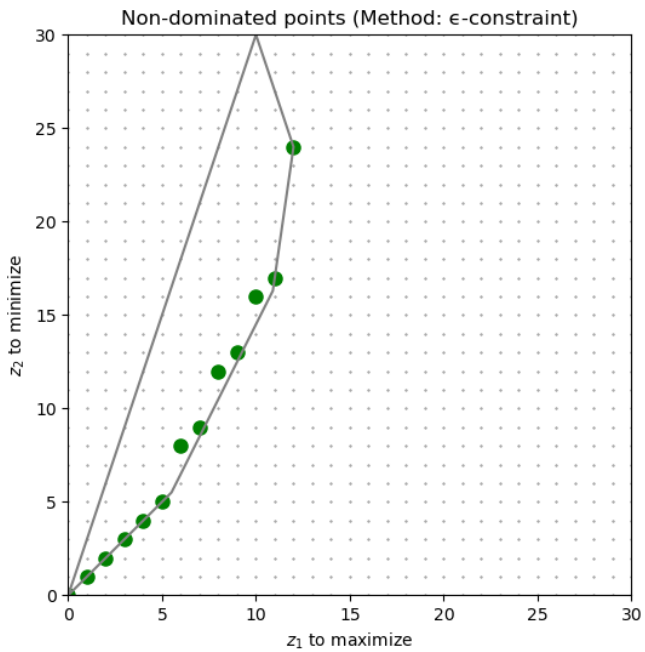
Optimize the MOO problem

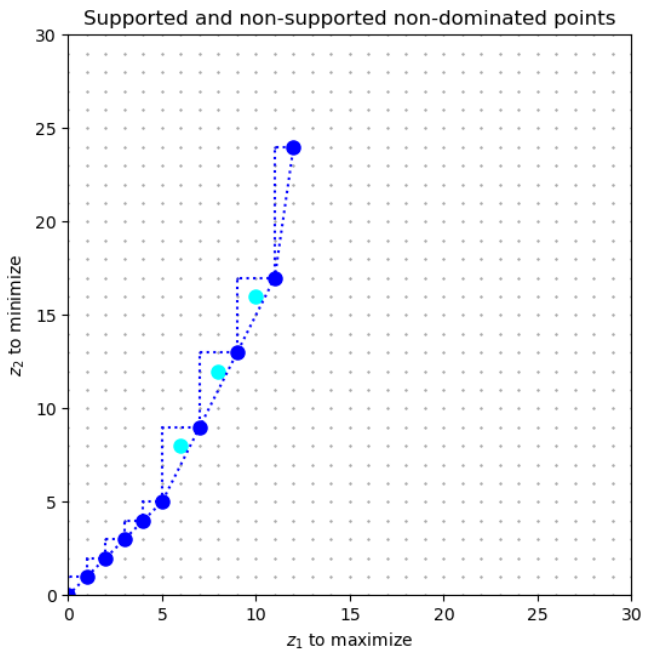
```
julia> optimize!(model)
```

Example (5/5)

Get X_E and Y_N

```
julia> for i in 1:result_count(model)
julia>     z1_opt = objective_value(model; result = i)[1]
julia>     z2_opt = -1 * objective_value(model; result = i)[2]
julia>     x1_opt = value(x1; result = i)
julia>     x2_opt = value(x2; result = i)
julia> end
```





More...

Oscar Dowson, Xavier Gandibleux, Gokhan Kof. From vOptGeneric.jl to MultiObjectiveAlgorithms.jl *JuliaDays 2023*, 4-6 October 2023. Paris (France).

JuMP:
`https://jump.dev/`

MultiObjectiveAlgorithms:
`https://github.com/jump-dev/MultiObjectiveAlgorithms.jl`

Examples:
`https://jump.dev/JuMP.jl/stable/tutorials/linear/multi_objective_examples/`

vOptSolver:
`https://github.com/vOptSolver`

Tomorrow

Practice of

1. Julia
2. JuMP
3. MultiObjectiveAlgorithms