

# Use of Julia/JuMP for Multi-Objective Optimization

Prof. Dr. Xavier Gandibleux

Nantes Université, France

June 2025

# Slides, examples, notebooks

<https://github.com/xgandibleux/EUR02025/>



# Why this adventure with Julia and JuMP for MOO?

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

Sub-task 2.2: Experimental Analysis and Prototype Development.

- ↳ The project of a multi-objective MILP solver was born!

# Purposes of the solver (1/2)

Expectations:

Easy to

- ▶ formulate a problem
  - ▶ provide the data
  - ▶ solve a problem
  - ▶ query the results
  - ▶ analyze the solutions
- 
- ▶ install (no need of being a geek)

↳ Julia programming language: free, open source, multi-platform  
JuMP modeling language: expressive, efficient, evolutive

# Purposes of the solver (1/2)

Expectations:

Easy to

- ▶ formulate a problem
- ▶ provide the data
- ▶ solve a problem
- ▶ query the results
- ▶ analyze the solutions
  
- ↳ Julia programming language: free, open source, multi-platform  
JuMP modeling language: expressive, efficient, evolutive

## Purposes of the solver (2/2)

Public and usages:

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

2017: Introduction of vOptSolver

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).

<https://github.com/vOptSolver>

## Purposes of the solver (2/2)

Public and usages:

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

### 2017: Introduction of **vOptSolver**

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).

<https://github.com/vOptSolver>

# From vOptGeneric to MultiObjectiveAlgorithms

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

# From vOptGeneric to MultiObjectiveAlgorithms

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

# Agenda

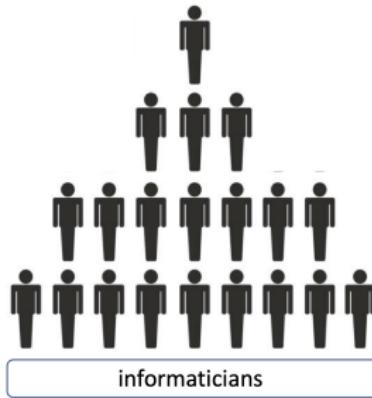
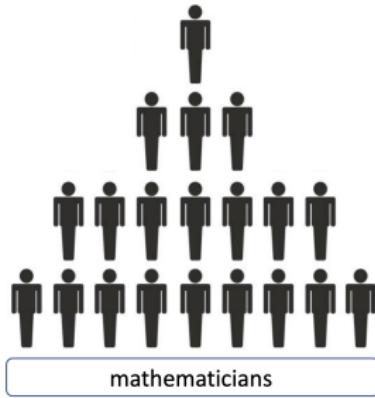
1. Few words about [Julia](#)
2. Brief introduction to [JuMP](#)
3. Optimization with [MultiObjectiveAlgorithms](#)

# Julia

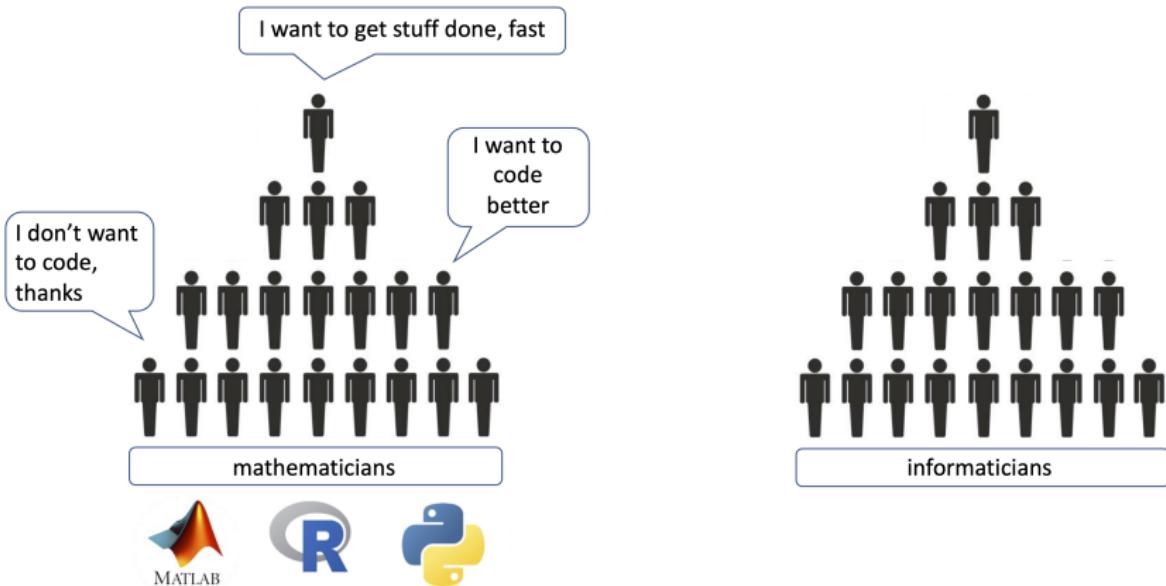
Coding and Running  
Scientific Computing Algorithms

<https://julialang.org/>

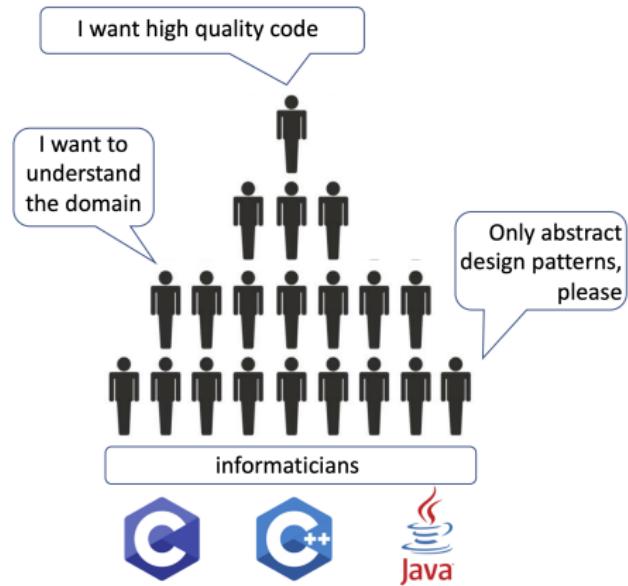
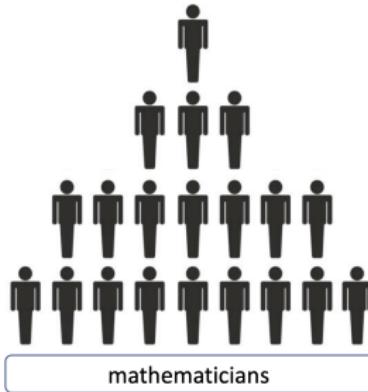
# Julia



# Julia

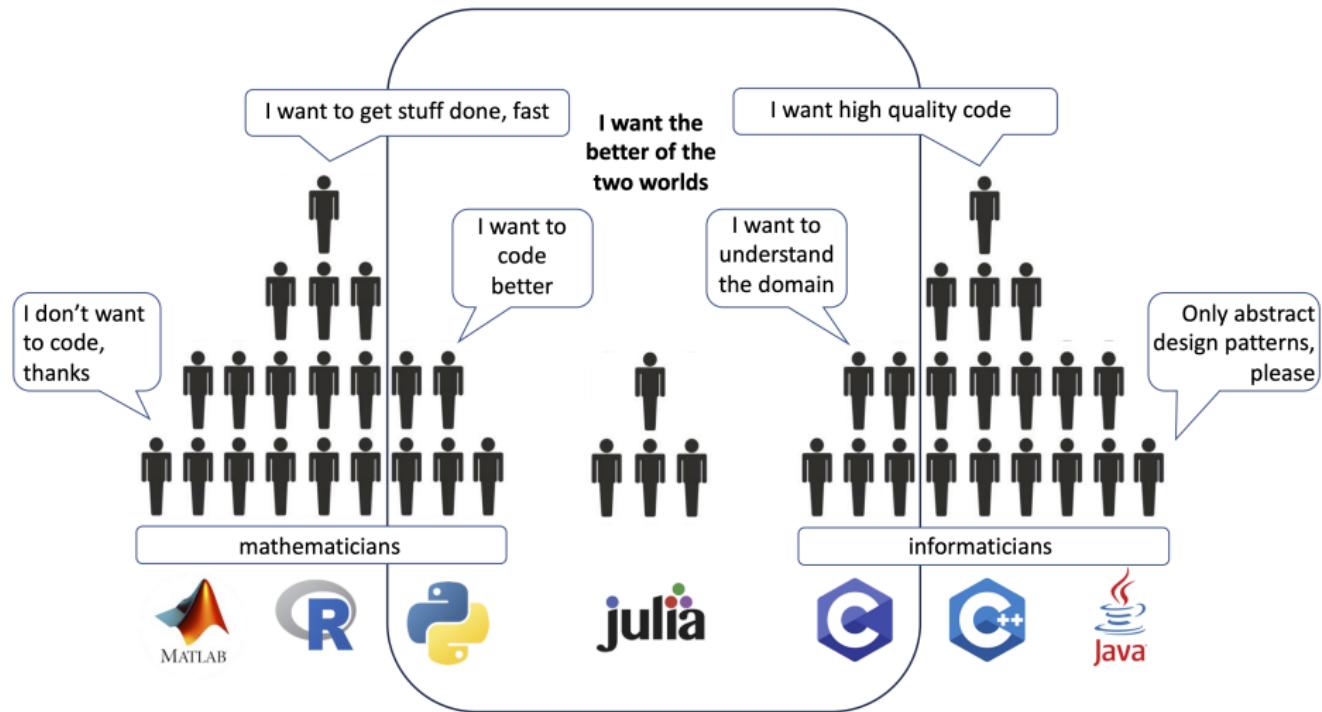


# Julia



# Julia

## A programming language for optimization



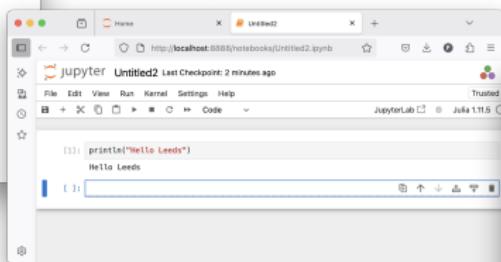
# Running a Julia code into...

...the Julia REPL



```
xavierg - Julia - julia - 80x24
Last login: Fri May 30 13:26:50 on ttys003
(base) xavierg@xaviers-Mac ~ % julia
julia> println("Hello Leeds")
Hello Leeds
julia>
```

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

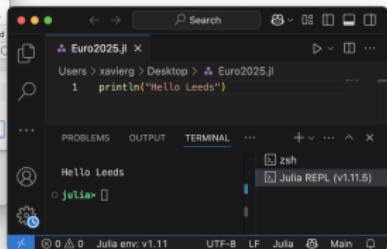


jupyter Untitled2 Last Checkpoint: 2 minutes ago

```
File Edit View Run Kernel Settings Help
JupyterLab □ Julia 1.11.5 Trusted
```

```
[[{"id": "1", "label": "Hello Leeds"}]]
```

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



Euro2025.jl

```
Users > xavierg > Desktop > Euro2025.jl
1   println("Hello Leeds")
```

PROBLEMS OUTPUT TERMINAL ... + v ^ x

zsh
Julia REPL (v1.15)

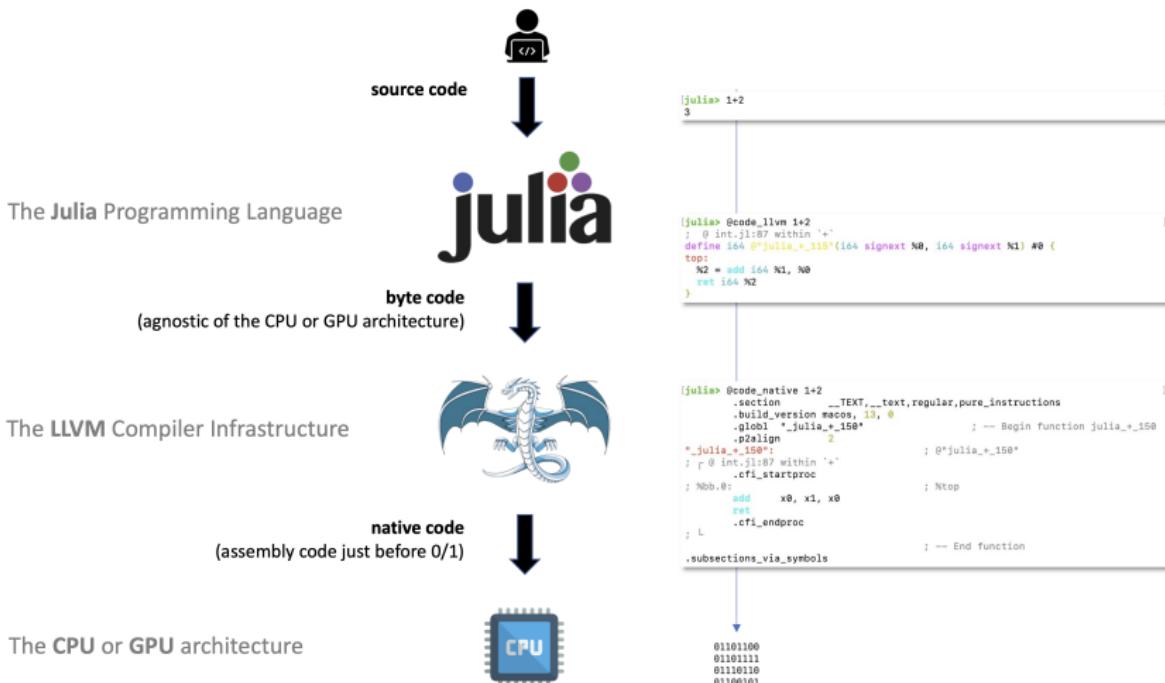
... or also in command line into a terminal.

# Julia



# Julia

LLVM: middleman between source code and compiled native code



# Example (1/2)

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

```
@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
```

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;
```

## 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)

↳ <https://github.com/xgandibleux/EURO2025/tree/main/2.examples>

## 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)

↳ <https://github.com/xgandibleux/EURO2025/tree/main/2.examples>

## 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)

↳ <https://github.com/xgandibleux/EURO2025/tree/main/2.examples>

## 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)

↳ <https://github.com/xgandibleux/EURO2025/tree/main/2.examples>

## 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.

Resources:

[The Julia Programming Language](https://julialang.org/) (<https://julialang.org/>):  
download, documentation, resources, community, etc.

Conferences:

[JuliaCon Global](https://juliacon.org/) (<https://juliacon.org/>):  
Annual International Conference since 2014 (Chicago, Cambridge, Berkeley,  
London, Baltimore, online, Eindhoven). **2025: Pittsburgh**

[JuliaCon Local](https://juliacon.org/) (<https://juliacon.org/>):  
Annual International Conference since 2023 (Eindhoven). **2025: Paris**

[JuliaDays](#):

the Julia & Optimization days in France since 2019 (Nantes, Paris, Toulouse).

## More (2/2)...

Example of material for optimizers:

Course

### **An introduction to Julia and JuMP for Operations Research**

Xavier Gandibleux

ÖGOR Summer-Workshop for PhD-candidates and Post-Docs (Krems, Austria)

<https://github.com/xgandibleux/Krems2023>

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>

# Practice

1. Installing Julia on your computer  
↳ <https://julialang.org/>
2. Discovering the four modes of the REPL
3. Installing the package IJulia
4. Running Julia code into Jupyter notebook
5. Installing the Julia extension into VScode
6. Editing and running Julia code into VScode

# JuMP

Coding and Solving  
Mathematical Programming Models

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.

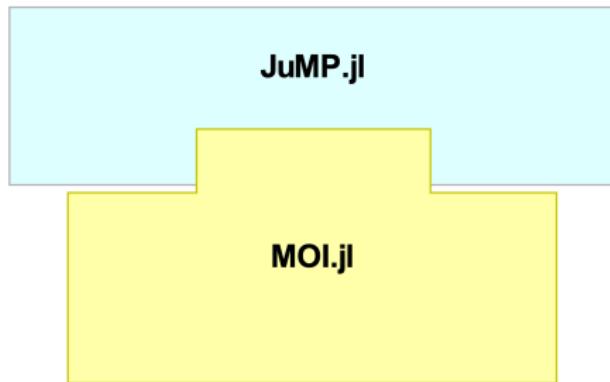
# Overview of JuMP

- ▶ **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

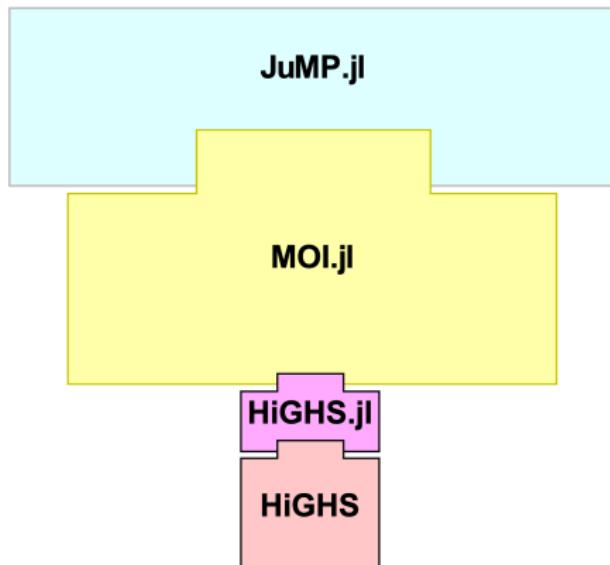
```
julia> using Pkg  
julia> Pkg.add("JuMP")
```



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

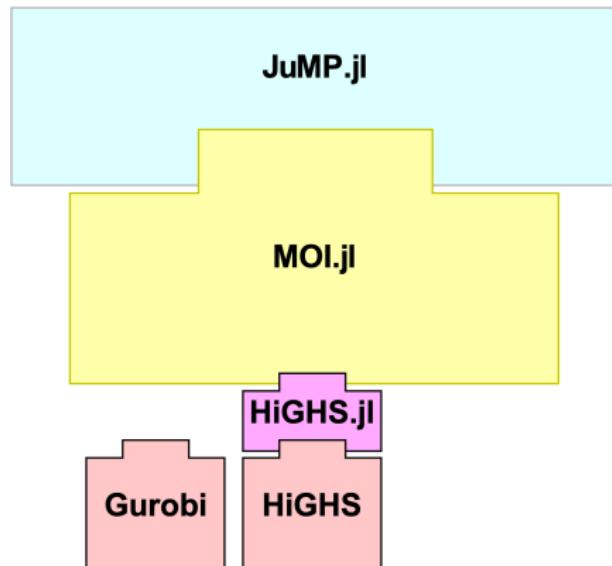
# Getting started

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



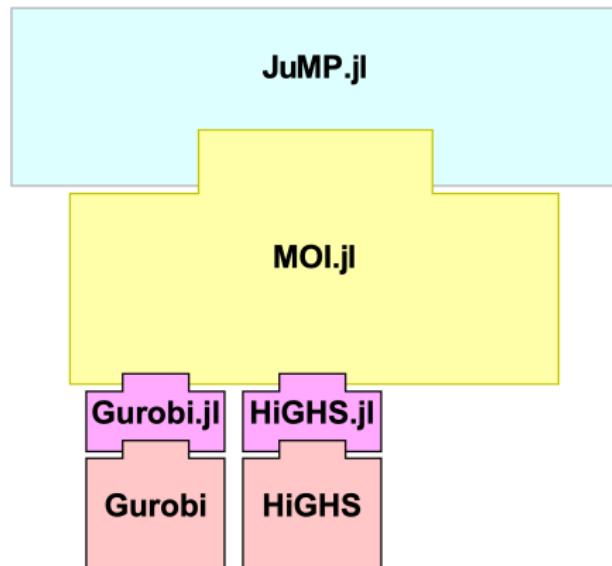
# Getting started

With Gurobi or CPLEX or Xpress MP solver properly installed on your computer...



# Getting started

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



# Modeling and solving a (MI)LP with JuMP

## The optimization problem

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

# Declaring the packages to use

```
julia> using JuMP, HiGHS
```

# Coding the optimization problem (1/4)

Creating a model:

```
modelId = Model( )
```

```
julia> modLP = Model( )
```

Result:

```
[ ]
```

## Coding the optimization problem (2/4)

Defining a variable:

```
@variable(modelId, variable)
```

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

Result:

$$\left[ \begin{array}{c} \\ \\ \\ \end{array} \quad , \quad \begin{array}{c} x_1 \\ x_2 \end{array} \quad \geqslant \quad \begin{array}{c} 0 \end{array} \quad (4) \end{array} \right]$$

## Coding the optimization problem (3/4)

Defining an objective function:

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

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

Result:

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

## Coding the optimization problem (4/4)

Defining a constraint:

```
@constraint(modelId, id, constraint)
```

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

Result:

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

# Solving the optimization problem

```
set_optimizer(modelId, solver )
```

```
julia> set_optimizer(modLP, HiGHS.Optimizer)
```

```
set_silent(modelId )
```

```
julia> set_silent(modLP)
```

```
optimize!(modelId )
```

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

# Solving the optimization problem

```
set_optimizer(modelId, solver )
```

```
julia> set_optimizer(modLP, HiGHS.Optimizer)
```

```
set_silent(modelId )
```

```
julia> set_silent(modLP)
```

```
optimize!(modelId)
```

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

# Solving the optimization problem

```
set_optimizer(modelId, solver )
```

```
julia> set_optimizer(modLP, HiGHS.Optimizer)
```

```
set_silent(modelId )
```

```
julia> set_silent(modLP)
```

```
optimize!(modelId )
```

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

## Querying the result

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

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

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

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

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

## Querying the result

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

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

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

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

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

## Querying the result

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

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

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

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

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

## Summary (explicit model)

```
julia> using JuMP, HiGHS  
  
julia> modLP = Model( )  
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> set_optimizer(modLP, HiGHS.Optimizer)  
julia> set_silent(modLP)  
julia> optimize!(modLP)  
  
julia> @show is_solved_and_feasible(modLP)  
julia> @show objective_value(modLP)  
julia> @show value(x1), value(x2)
```

↳ <https://github.com/xgandibleux/EUR02025/blob/main/2.examples/1IP.ipynb>

## 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. *JuMP-dev 2024 - Co-Located with ISMP'2024*, July 19-21, 2024. GERAD research center, Montréal, Canada.

<https://www.youtube.com/watch?v=cMUTdfoAMpo>

### Resources:

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

### Workshops:

JuMP-dev (<https://jump.dev/meetings/jumpdev2025/>):  
Regular meeting since 2017: Cambridge (USA), Bordeaux (France), Santiago (Chile), online, Montréal (Canada). **2025: Auckland (New Zealand)**

# Practice

1. Installing the packages JuMP and HiGHS
2. Coding and solving an IP problem

↳ <https://github.com/xgandibleux/EUR02025/blob/main/3.practice/Part2.ipynb>

# MOA

Moving from  
single to multiple objectives MIP optimization

MultiObjectiveAlgorithms.jl

# Multiobjective Optimization Problem (MOP) notations

$$\begin{aligned} \min \quad & \textcolor{red}{F}(x) \\ \text{s.t.} \quad & x \in X \end{aligned}$$

where:

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

Computing sets  $\textcolor{blue}{X}_E$  and  $\textcolor{blue}{Y}_N$  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  $y \neq y^*$ .

$\textcolor{blue}{Y}_N$  is the set of nondominated points.

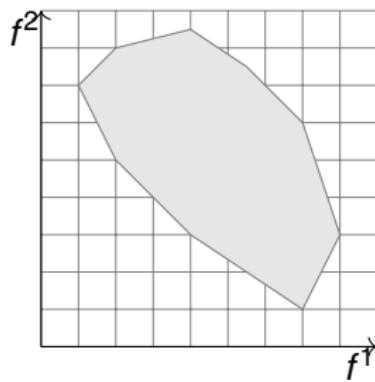
$x^* \in X$  is efficient if  $y^*$  is nondominated.

$\textcolor{blue}{X}_E$  is a complete set of efficient solutions.

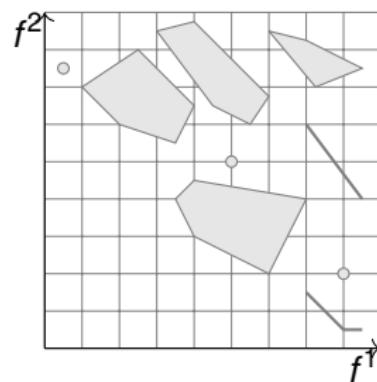
# Examples of $Y$ and $Y_N$ when $p = 2$

Computing  $Y_N$  for Multiobjective (linear) Optimization Problems (MOP):

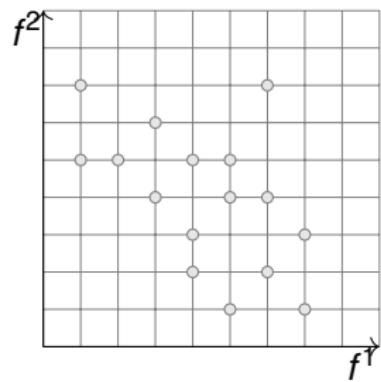
$$x \in \mathbb{R}^{n_1}$$



$$x \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}$$



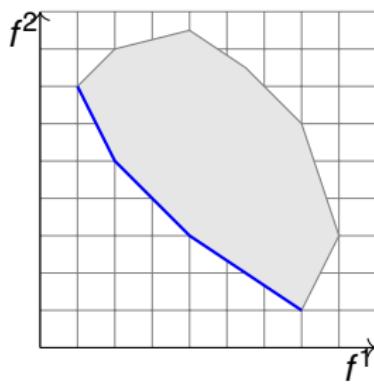
$$x \in \mathbb{Z}^{n_2}$$



# Examples of $Y$ and $Y_N$ when $p = 2$

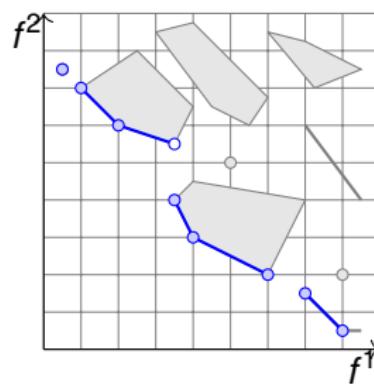
Computing  $Y_N$  for Multiobjective (linear) Optimization Problems (MOP):

$$x \in \mathbb{R}^{n_1}$$



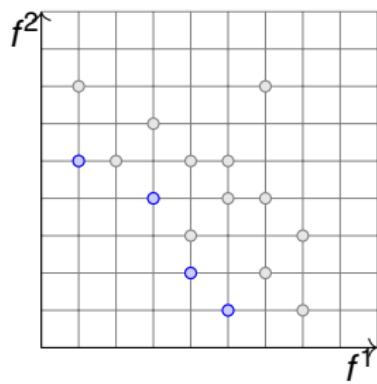
Continuous  
nondominated set  
(edges)

$$x \in \mathbb{R}^{n_1} \times \mathbb{Z}^{n_2}$$



Nondominated set  
composed of edges  
that are either closed,  
half-open, open or  
reduced to a point

$$x \in \mathbb{Z}^{n_2}$$



Discrete set of  
nondominated  
points

# Existing MP Software for MOP

▶ ADBASE	MOLP	(1975)
▶ BENSOOLVE (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 + vOptSolver MOA + solvers	MOIP	(2017, last rev 2025)
▶ PaMILO (with Gurobi/CPLEX)	MOMILP( $Y_{SN}$ )	(2020?; last rev'25)
▶ HiGHS	...	(since Ver1.10?, 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 + vOptSolver MOA + solvers | MOIP               | (2017, last rev 2025)   |
| ▶ PaMILO (with Gurobi/CPLEX)      | MOMILP( $Y_{SN}$ ) | (2020?; last rev'25)    |
| ▶ HiGHS                           | ...                | (since Ver1.10?, 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 + vOptSolver MOA + solvers | MOIP               | (2017, last rev 2025)   |
| ▶ PaMILO (with Gurobi/CPLEX)      | MOMILP( $Y_{SN}$ ) | (2020?; last rev'25)    |
| ▶ HiGHS                           | ...                | (since Ver1.10?, 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)

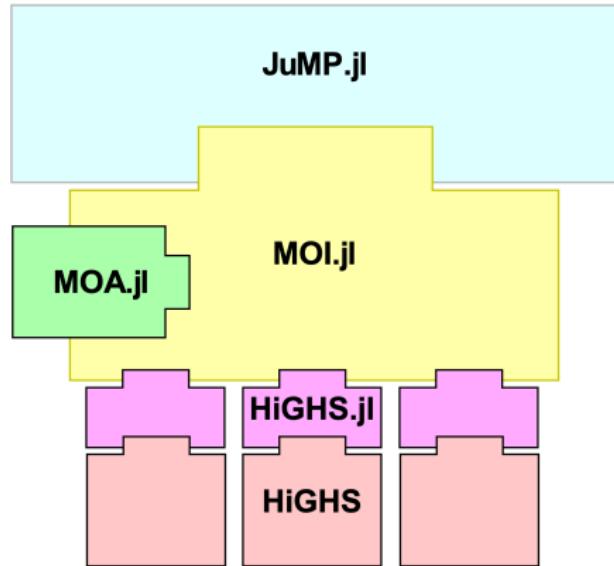
Installing the package:

```
julia> using Pkg
```

```
julia> Pkg.status()
Status `~/.julia/environments/v1.11/Project.toml'
[60bf3e95] HiGHS v1.18.1
[4076af6c] JuMP v1.26.0
```

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

## Getting Started (con't)



## Algorithms currently available (v1.4.3)

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

+ optimization attributes coming with a given algorithm

## Algorithms currently available (v1.4.3)

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

+ optimization attributes coming with a given algorithm

## Algorithms currently available (v1.4.3)

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

+ optimization attributes coming with a given algorithm

# Zoom on the $\epsilon$ -constraint algorithm

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.

Attributes:

- ▶ `MOA.EpsilonConstraintStep()`

algorithm uses this value as the  $\epsilon$  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()`

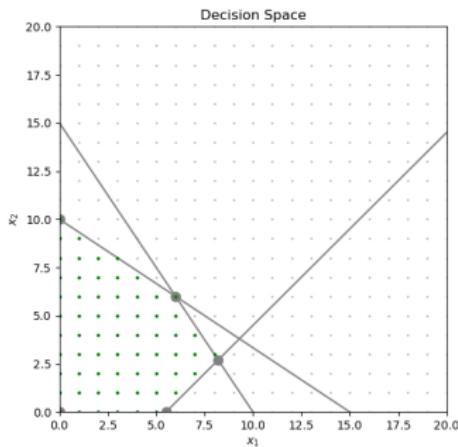
if this attribute is set then, instead of using the `MOA.EpsilonConstraintStep`, with a slight abuse of notation, `EpsilonConstraint` divides the range of the first-objective's domain in objective space by `SolutionLimit` to obtain the  $\epsilon$  to use when iterating. Thus, at most `SolutionLimit`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

## Example (1/4)

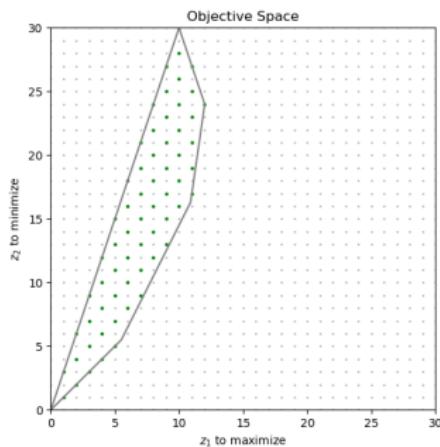
Compute  $X_E$  and  $Y_N$  for the following MOP:

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

Decision space:



Objective space:



## Example (2/4)

### Coding the MOO problem with JuMP

```
julia> using JuMP, HiGHS  
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 (3/4)

Setup the MIP solver: e.g. HiGHS

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

Setup the algorithm:  $\epsilon$ -constraint; step=1 (default value)

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

Optimize the MOO problem

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

## Example (3/4)

Setup the MIP solver: e.g. HiGHS

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

Setup the algorithm:  $\epsilon$ -constraint; step=1 (default value)

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

Optimize the MOO problem

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

## Example (3/4)

Setup the MIP solver: e.g. HiGHS

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

Setup the algorithm:  $\epsilon$ -constraint; step=1 (default value)

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

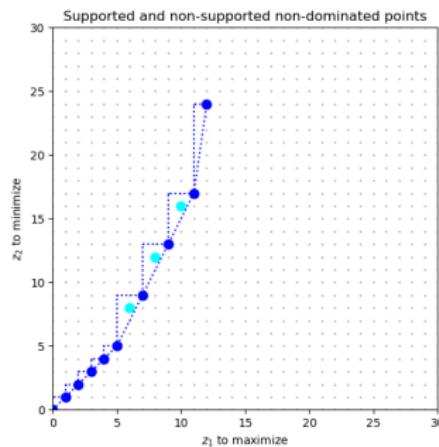
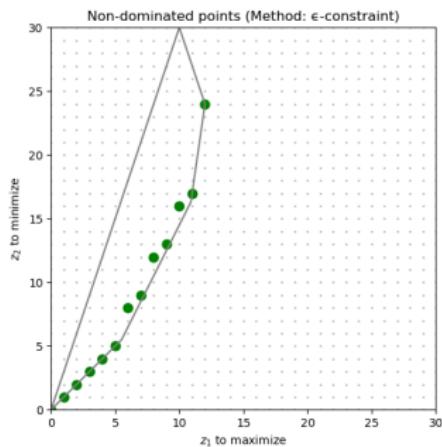
Optimize the MOO problem

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

## Example (4/4)

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
```



↳ <https://github.com/xgandibleux/EUR02025/blob/main/2.examples/2IPex.ipynb>

From  
an explicit to an implicit model  
with JuMP

## Example (1/2)

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

$$\begin{array}{llll} \max z_k & = & \sum_{j=1}^n c_j^k x_j & k = 1, \dots, p \\ \text{s.t.} & \sum_{j=1}^n a_{ij} x_j & \leq b_i & i = 1, \dots, m \\ & x_j & \in \mathbb{N} & j = 1, \dots, n \end{array}$$

```
julia> c1 = [1,1]; c2 = -1 * [1,3]; c = vcat(c1',c2')
```

```
julia> a = [2 3; 3 2; 1 -1]
```

```
julia> b = [30, 30, 5.5]
```

```
julia> m,n = size(a)
```

```
julia> p = size(c,1)
```

```
julia> md = Model( )
```

```
julia> @variable(md, x[1:n] ≥ 0, Int)
```

```
julia> @expression(md, z[k=1:p], sum(c[k,j]*x[j] for j=1:n))
```

```
julia> @objective(md, Max, [z[k] for k=1:p])
```

```
julia> @constraint(md, ct[i=1:m], sum(a[i,j]*x[j] for j=1:n) ≤ b[i])
```

## Example (1/2)

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

$$\begin{array}{llll} \max z_k & = & \sum_{j=1}^n c_j^k x_j & k = 1, \dots, p \\ \text{s.t.} & & \sum_{j=1}^n a_{ij} x_j \leq b_i & i = 1, \dots, m \\ & & x_j \in \mathbb{N} & j = 1, \dots, n \end{array}$$

```
julia> c1 = [1,1]; c2 = -1 * [1,3]; c = vcat(c1',c2')
julia> a = [2 3; 3 2; 1 -1]
julia> b = [30, 30, 5.5]
julia> m,n = size(a)
julia> p = size(c,1)
```

```
julia> md = Model( )
julia> @variable(md, x[1:n] ≥ 0, Int)
julia> @expression(md, z[k=1:p], sum(c[k,j]*x[j] for j=1:n))
julia> @objective(md, Max, [z[k] for k=1:p])
julia> @constraint(md, ct[i=1:m], sum(a[i,j]*x[j] for j=1:n) ≤ b[i])
```

## Example (1/2)

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

$$\begin{array}{llll} \max z_k & = & \sum_{j=1}^n c_j^k x_j & k = 1, \dots, p \\ \text{s.t.} & & \sum_{j=1}^n a_{ij} x_j \leq b_i & i = 1, \dots, m \\ & & x_j \in \mathbb{N} & j = 1, \dots, n \end{array}$$

```
julia> c1 = [1,1]; c2 = -1 * [1,3]; c = vcat(c1',c2')
```

```
julia> a = [2 3; 3 2; 1 -1]
```

```
julia> b = [30, 30, 5.5]
```

```
julia> m,n = size(a)
```

```
julia> p = size(c,1)
```

```
julia> md = Model( )
```

```
julia> @variable(md, x[1:n] ≥ 0, Int)
```

```
julia> @expression(md, z[k=1:p], sum(c[k,j]*x[j] for j=1:n))
```

```
julia> @objective(md, Max, [z[k] for k=1:p])
```

```
julia> @constraint(md, ct[i=1:m], sum(a[i,j]*x[j] for j=1:n) ≤ b[i])
```

## Example (1/2)

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

$$\begin{array}{llll} \max z_k & = & \sum_{j=1}^n c_j^k x_j & k = 1, \dots, p \\ s.t & & \sum_{j=1}^n a_{ij} x_j \leq b_i & i = 1, \dots, m \\ & & x_j \in \mathbb{N} & j = 1, \dots, n \end{array}$$

```
julia> c1 = [1,1]; c2 = -1 * [1,3]; c = vcat(c1',c2')
```

```
julia> a = [2 3; 3 2; 1 -1]
```

```
julia> b = [30, 30, 5.5]
```

```
julia> m,n = size(a)
```

```
julia> p = size(c,1)
```

```
julia> md = Model( )
```

```
julia> @variable(md, x[1:n] ≥ 0, Int)
```

```
julia> @expression(md, z[k=1:p], sum(c[k,j]*x[j] for j=1:n))
```

```
julia> @objective(md, Max, [z[k] for k=1:p])
```

```
julia> @constraint(md, ct[i=1:m], sum(a[i,j]*x[j] for j=1:n) ≤ b[i])
```

## Example (1/3)

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

$$\begin{array}{llll} \max z_k & = & \sum_{j=1}^n c_j^k x_j & k = 1, \dots, p \\ \text{s.t.} & & \sum_{j=1}^n a_{ij} x_j \leq b_i & i = 1, \dots, m \\ & & x_j \in \mathbb{N} & j = 1, \dots, n \end{array}$$

```
julia> c1 = [1,1]; c2 = -1 * [1,3]; c = vcat(c1',c2')
julia> a = [2 3; 3 2; 1 -1]
julia> b = [30, 30, 5.5]
julia> m,n = size(a)
julia> p = size(c,1)
```

```
julia> md = Model( )
julia> @variable(md, x[1:n] ≥ 0, Int)
julia> @expression(md, z[k=1:p], sum(c[k,j]*x[j] for j=1:n))
julia> @objective(md, Max, [z[k] for k=1:p])
julia> @constraint(md, ct[i=1:m], sum(a[i,j]*x[j] for j=1:n) ≤ b[i])
```

## Example (2/3)

Get  $X_E$  and  $Y_N$

```
julia> for i in 1:result_count(model)
julia>     z_opt = [objective_value(md; result = i)[k] for k=1:p]
julia>     x_opt = value.(x; result = i)
julia> end
```

## Example (3/3)

### Plot $Y_N$

```
julia> using Plots
julia> Plots.scatter(
julia>   [value(z[1]; result = i) for i in 1:result_count(md)],
julia>   [-1 * value(z[2]; result = i) for i in 1:result_count(md)];
julia>   xlabel = "objective 1",
julia>   ylabel = "objective 2",
julia>   title = "objective space",
julia>   legend = false,
julia>   xlims = (0,25),
julia>   ylims = (0,25),
julia>   aspect_ratio=:equal,
julia> )
```



<https://github.com/xgandibleux/EUR02025/blob/main/2.examples\2IPim.ipynb>

## 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/

# Roadmap

Working on it:

- ▶ making the 3-obj algorithms reliable
  - ↳ working on a set of test instances
  - ↳ <https://github.com/xgandibleux/EUR02025/blob/main/2.examples/3IP.ipynb>

Missing parts:

- ▶ solving MOLP with a **multi-objective simplex algorithm**
  - ↳ working on HiGHS2LP, a parametric variant of HiGHS
- ▶ solving MOMILP with a **branch-and-xx algorithm**
  - ↳ works in progress in the literature
- ▶ **detecting problem-specific structure** in a presolve
  - ↳ solving MOCO with specific algorithms

# Roadmap

Working on it:

- ▶ making the 3-obj algorithms reliable
  - ↳ working on a set of test instances
  - ↳ <https://github.com/xgandibleux/EUR02025/blob/main/2.examples/3IP.ipynb>

Missing parts:

- ▶ solving MOLP with a **multi-objective simplex algorithm**
  - ↳ working on HiGHS2LP, a parametric variant of HiGHS
- ▶ solving MOMILP with a **branch-and-xx algorithm**
  - ↳ works in progress in the literature
- ▶ **detecting problem-specific structure** in a presolve
  - ↳ solving MOCO with specific algorithms

# Practice

1. Installing the packages MultiObjectiveAlgorithms and Plots
2. Coding and solving a MOIP problem

↳ <https://github.com/xgandibleux/EUR02025/blob/main/3.practice/Part3.ipynb>

# Call for participation



European conference on the Julia programming language  
October 2-3, 2025 — Paris (France)

<https://juliacon.org/>

## Keynote Speakers:

Laura Grigori (EPFL, Switzerland)  
Ivet Galabova (HiGHS, Scotland)  
Tim Besard (JuliaHub, Belgium)

## Venue:

