Disclaimer

Environment: This notebook is only working under the versions:

- Julia v1.0.x
- JuMP v0.18.x
- GLPKMathProgInterface v0.4.4
- GLPK v0.8.2
- vOptSpecific v1.0.0
- vOptGeneric v0.2.0

Description: This notebook introduces the minimal notions to know on Julia and JuMP for using vOptSolver. The topics covered are:

- Installing Julia, JuMP, and vOptSolver
- Implementing algorithms and subroutines in Julia
- Representing vectors and matrices in Julia
- Plotting the results with and without graphical environment
- Solving LP and MIP with JuMP
- Solving multiple objective linear optimization problems with vOptSolver

Author: Xavier Gandibleux, Nantes (France)

Shortest path from Julia and JuMP to vOptSolver for optimizers

Residential Seminar *Recent Advances in Multi-Objective Optimization (RAMOO)*

November 11-16, 2018, Nantes, France

1. Getting started

1.1 Julia programming language

Versions:

- First: August 23, 2009
- Stable: August, 8 2018
- Current: September 29, 2018

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

Julia Channel on YouTube: https://www.youtube.com/user/JuliaLanguage (https://www.youtube.com/user/JuliaLanguage)

1.2 Working with Julia

- a distant use

  on the cloud with

  (1) *JuliaBox* (http://juliabox.com (http://juliabox.com))

- a local use

  on macOS, linux, windows with

  (2) *JuliaLang* full local installation (https://julialang.org/ (https://julialang.org/))

  (3) *JuliaPro* (https://juliacomputing.com/products/juliapro.html (https://juliacomputing.com/products/juliapro.html))

1.3 Interacting with Julia

- via the *REPL*

  Reads what you type

  Evaluates it

  Prints out the return value, then

  Loops back and does it all over again

- via the *Jupyter* notebook

  Julia

  Python

  R

1.4 Programming in Julia

- coding *interactively*

  in a terminal: `julia`

  with the REPL or Jupyter

  in a terminal: `jupyter notebook`

- coding *offline*

  with your prefered editor (i.e. ATOM, Juno, emacs, etc.) and executing with the REPL

2. Packages

2.1 The collection of packages

Julia Observer provides a visual interface for exploring packages:

```
https://juliaobserver.com/
```

Some useful packages:

- **Pkg**: primitives for managing packages
- **Printf**: primitives for printf display
- **Random**: primitives for random number
- **LinearAlgebra**: primitives for linear algebra

- **UnicodePlots**: Plotting in text mode
- **PyPlot**: Plotting for Julia based on matplotlib.pyplot
- **Plots**: a high-level plotting package that interfaces with other plotting packages

- **JuMP**: Modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear)
- **GLPK**: GLPK wrapper module for Julia
- **GLPKMathProgInterface**: Interface between the GLPK.jl wrapper and MathProgBase.jl
- **Gurobi**: Julia interface for Gurobi Optimizer
- **CPLEX**: Julia interface for the CPLEX optimization software
- **vOptSpecific**: Solver of multiobjective linear optimization problems (MOCO, MOILP, MOMILP, MOLP): specific part
- **vOptGeneric**: Solver of multiobjective linear optimization problems (MOCO, MOILP, MOMILP, MOLP): generic part

2.2 Using a package

Commands to invoque **each time** before the first use of the given package:

In [ ]:
```
using Pkg
using JuMP
using Printf
using LinearAlgebra
```

2.3 Adding a package

Commands to invoque **once** before the first use of the given package:

In [ ]:
```
Pkg.add("UnicodePlots")
Pkg.add("JuMP")
Pkg.add("GLPK")
Pkg.add("GLPKMathProgInterface")
```

2.4 REPL in Package mode

In [ ]:
```
]
```

Commands to know:

`add` : adding a package

`status` : list of packages installed

`rm` : remove a package

`help` : list of commands available

`CTRL-C` : leave the package mode

---

3. First instructions (Julia v1.0 and higher)

3.1 Assign a value to a variable

Integer (Int8, Int16, Int32, Int64):

In [ ]:
```
zipcode = 44000
```

Float (Float16, Float32, Float64):

In [ ]:
```
price = 29.95
```

Character:

In [ ]:
```
dot = '.'
```

String:

In [ ]:
```
sentence = "."
```

3.2 Display the value of a variable

In [ ]:
```
print(zipcode)
println()
print(price)
```

In [ ]:
```
println("Zip Code: ",zipcode)
println(price)
```

In [ ]:
```
println(zipcode, " | ", price, " | ", dot, " | ", sentence)
```

In [ ]:
```
using Printf
@printf("Zip Code: %d \n",zipcode)
```

In [ ]:
```
@show zipcode
```

3.3 Get the type of a variable

In [ ]:
```
typeof(zipcode)
```

3.4 Write a comment

In [ ]:
```
# this is a line comment
```

In [ ]:
```
#= And this
   is a block
   of comment
=#
```

3.5 The conditionnals

if ... endIf

In [ ]:
```
if zipcode == 44000
    println("Welcome to Nantes")
end
```

In [ ]:
```
if zipcode == 44000
    println("Welcome to Nantes")
else
    println("Welcome to France")
end
```

In [ ]:
```
if zipcode == 44000
    println("Welcome to Nantes Central area")
elseif zipcode == 44100
    println("Welcome to Nantes West area")
elseif zipcode == 44200
    println("Welcome to Nantes South area")
elseif zipcode == 44300
    println("Welcome to Nantes North-East area")
else
    println("Welcome to France")
end
```

ifelse

In [ ]:
```
println("Welcome to ", ifelse(zipcode == 44000, "Nantes", "France"))
```

3.6 The repetitives

for ... endFor

In [ ]:
```
for i=1:10
    print(i , " ")
end
```

In [ ]:
```
for i in 1:10 # a interval
    print(i , " ")
end
```

In [ ]:
```
for i in 1:2:10 # a interval with a step
    print(i , " ")
end
```

In [ ]:
```
for i in (44000, 44100, 44200, 44300) # a tuple
    print(i , " ")
end
```

In [ ]:
```
for i in [44000, 44100, 44200, 44300] # a vector
    print(i , " ")
end
```

In [ ]:
```
for i in "44000" # a string
    print(i , " ")
end
```

In [ ]:
```
for i in Dict("Central"=>44000, "University"=>44300) # a dictionary
    print(i , " ")
end
```

In [ ]:
```
for i in Set([44000, 44300]) # a set
    print(i , " ")
end
```

while ... endWhile

In [ ]:
```
zipcode = 44000
while zipcode ≤ 44300
    print(zipcode, " ")
    global zipcode = zipcode + 100 # global required outside a function
end
```

repeat ... until

In [ ]:
```
zipcode = 44000
while true
    print(zipcode, " ")
    global zipcode = zipcode + 100 # global required outside a function
    zipcode > 44300 && break  # repeat ... until
end
```

3.7 Writing and calling a function

Single expression function:

In [ ]:
```
f(x) = x^2 + 7
```

In [ ]:
```
f(2)
```

Anonymous function:

In [ ]:
```
map(x -> x^2 + 7 , 2) # over a scalar
```

In [ ]:
```
map(x -> x^2 + 7 , [2, 7, 4]) # over a vector
```

In [ ]:
```
map((x,b) -> x^2 + b , 2, 7) # multiple parameters
```

Generalized function:

In [ ]:
```
function affinefct( x::Int64 ) # single parameter
    println("1 parameter")
    y = x^2 + 7
    return y
end
```

In [ ]:
```
affinefct(2)
```

In [ ]:
```
function affinefct( x::Int64, b::Int64 ) # multiple parameters
    println("2 parameters")
    y = x^2 + b
    return y
end
```

In [ ]:
```
affinefct(2,7)
```

3.8 Help online

In [ ]:
```
? for
```

3.9 Exercice

Let $f : [a; b] \to \mathbb{R}$ a function strictly monotonic on the interval $[a, b]$. We suppose the equation $f(x) = 0$ has one and only one solution on the interval. Determine this value for a given precision using a dichotomic method.

---

4. Mathematic functionalities responding to the optimizer's needs

4.1 Vectors

Non initialized vector:

In [ ]:
```
v = Array{Int64}(undef,5)
```

In [ ]:
```
v = Vector{Int64}(undef,5)
```

Vector of zeros:

In [ ]:
```
v0 = zeros(Int64,5)
```

Vector of ones:

In [ ]:
```
v1 = ones(Int64,5)
```

Assign one value:

In [ ]:
```
v[2]= 6
```

In [ ]:
```
v[end] = 0
```

Assign several values:

In [ ]:
```
v = [2,3.5, '.',"brol",5]
```

In [ ]:
```
v[2:3] = [3, 7]
```

4.2 Matrices

Non initialized matrix:

In [ ]:
```
m = Array{Int64}(undef,2,3)
```

In [ ]:
```
m = Matrix{Float64}(undef,2,3)
```

Matrix of zeros:

In [ ]:
```
m0 = zeros(Float64,2,3,4)
```

Matrix of ones:

In [ ]:
```
m1 = ones(Float64,2,3,4)
```

Matrix initialized with a given value:

In [ ]:
```
m = fill(3.14,2,3)
```

4.3 Example of FILO list

Create empty list:

In [ ]:
```
L=(Int64)[]
```

Push one item:

In [ ]:
```
push!(L,6)
```

Pop one item:

In [ ]:
```
pop!(L)
```

4.4 Random numbers

Float in [0;1]

In [ ]:
```
rand()
```

Integer in [2;7]

In [ ]:
```
rand(2:7)
```

Serie of 5 integers in [40;100]

In [ ]:
```
rand(40:100,5)
```

Serie of 5 floats in [0;1]

In [ ]:
```
rand(5)
```

4.5 Sample of available functions

In [ ]:
```
maximum([2,7,4])
```

In [ ]:
```
minimum([2,7,4])
```

In [ ]:
```
sum([2,7,4])
```

In [ ]:
```
prod([2,7,4])
```

In [ ]:
```
v1 = [2,7,4]
v2 = [3,8,1]
```

In [ ]:
```
v1.*v2
```

In [ ]:
```
transpose(v1)*v2   # or v1'v2   or dot(v1,v2)
```

In [ ]:
```
sort([2,7,4])
```

In [ ]:
```
argmin([2,7,4])
```

In [ ]:
```
argmax([2,7,4])
```

4.5 Exercice

For the unidimensional 01 knapsack problem,

$$z = \max \left\{ px \mid wx \leq c,\ x \in \{0, 1\}^n \right\}$$

with

$$\begin{aligned} n &= 5 \\ p &= (5, 3, 2, 7, 4) \\ w &= (2, 8, 4, 2, 5) \\ c &= 10 \end{aligned}$$

compute the linear relaxation.

---

5 Others important concepts of JuliaLang to learn

- Structure
- Dictionary
- Set
- Text file

- Types and multiple dispatch
- Embedding C code
- Parallelism

- REPL in shell mode

In [ ]:
```
;
```

- Loading your julia code from a file (myprg.jl)

In [ ]:
```
include("myprog.jl")
```

---

6 Plotting

6.1 Overview

- **UnicodePlots**: a quick and easy way to draw plots in the REPL

  https://github.com/Evizero/UnicodePlots.jl (https://github.com/Evizero/UnicodePlots.jl)

- **PyPlot**: a collection of command style functions that make matplotlib work like MATLAB

  https://github.com/JuliaPy/PyPlot.jl (https://github.com/JuliaPy/PyPlot.jl)

  https://matplotlib.org/tutorials/introductory/pyplot.html (https://matplotlib.org/tutorials/introductory/pyplot.html)

- **Plots**: a high-level plotting package that interfaces with other plotting packages

  https://github.com/JuliaPlots/Plots.jl (https://github.com/JuliaPlots/Plots.jl)

To avoid any interactions (and thus error messages) between plotting packages, use only one plotting package simultaneously.

6.2 Getting UnicodePlots

In [ ]:
```
#using Pkg
#Pkg.add("UnicodePlots")
using UnicodePlots
```

6.3 Example

In [ ]:
```
x = Vector(-5:5) ; y = 4*x .+ 10
myPlot = lineplot(x, y, title = "y=f(x)", name = "y=4x+10")
```

6.4 Getting PyPlot

In [ ]:
```
#using Pkg
#Pkg.add("PyPlot")
using PyPlot
```

6.5 Example

In [ ]:
```
#title("y=f(x)")
#xlabel("x values") ; ylabel("y values")

# To avoid interaction with others plotting packages :
PyPlot.title("y=f(x)")
PyPlot.xlabel("x values") ; PyPlot.ylabel("y values")

grid()
x = Vector(-5:5) ; y = 4*x .+ 10
plot(x,y,label="y=4x+10")
legend(loc=4, fontsize="small")
show()
```

6.6 Getting Plots

In [ ]:
```julia
#using Pkg
#Pkg.add("Plots")
using Plots
```

In [ ]:
```julia
x = Vector(-5:5) ; y = 4*x .+ 10
plot(x,y,title="y=f(x)",label=["y=4x+10"],lw=2, legend=:bottomright, fmt = :png
)
```

7. JuMP (v0.18.4) with illustrated with GLPK

https://github.com/JuliaOpt/JuMP.jl (https://github.com/JuliaOpt/JuMP.jl)

Miles Lubin and Iain Dunning. Computing in Operations Research Using Julia. *INFORMS Journal on Computing*, 27(2), 238-248, 2015.

Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, 59 (2), 295-320, 2017.

ROADEF 2018 - Miles Lubin, *JuMP: past, present, and future* (YouTube video): https://www.youtube.com/watch?v=iJyEYKZjlnI (https://www.youtube.com/watch?v=iJyEYKZjlnI)

7.1 Overview

A modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear):

- **User friendliness**

  Syntax that mimics natural mathematical expressions.

- **Speed**

  Similar speeds to special-purpose modeling languages such as AMPL.

- **Solver independence**

  JuMP uses a generic solver-independent interface provided by the MathProgBase package. Currently supported solvers: Artelys Knitro, Bonmin, Cbc, Clp, Couenne, CPLEX, ECOS, FICO Xpress, GLPK, Gurobi, Ipopt, MOSEK, NLopt, and SCS.

- **Access to advanced algorithmic techniques**

  Including efficient LP re-solves and callbacks for mixed-integer programming.

- **Ease of embedding**

  JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Example:

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

7.2 Getting JuMP

In [ ]:
```
using JuMP
```

7.3 Getting a MIP solver

In [ ]:
```
using GLPK, GLPKMathProgInterface # example for GLPK
```

7.4 Creating a Model

In [ ]:
```
lp = Model( solver = GLPKSolverLP() ) # example for GLPK in mode LP
```

7.5 Defining Variables

In [ ]:
```
@variable(lp, x1 >= 0) # continuous variable
@variable(lp, x2 >= 0)
```

`x >= lb` : continuous variable (lb : lower bound value)

`x, Int` : discrete varibale

`x, Bin` : binary variable

7.6 Defining Objective

In [ ]:
```
@objective(lp, Max, x1 + 3x2)
```

7.7 Defining Constraints

In [ ]:
```
@constraint(lp, cst1, x1 + x2 <= 14)
@constraint(lp, cst2, -2x1 + 3x2 <= 12)
@constraint(lp, cst3, 2x1 - x2 <= 12)
```

7.8 Display the model

In [ ]:
```
print(lp)
```

7.9 Solve the model

In [ ]:
```
status = solve(lp)
```

7.10 Retrieve the results

In [ ]:
```
if status == :Optimal
    zOpt  = getobjectivevalue(lp)
    @printf("  z=%5.2f  x1=%5.2f  x2=%5.2f \n",zOpt, JuMP.getvalue(x1), JuMP.getvalue(x2))
    @printf(" u1=%5.2f  u2=%5.2f  u3=%5.2f \n",getdual(cst1), getdual(cst2), getdual(cst3))

elseif status == :Unbounded
    println("problem unbounded")

elseif status == :Infeasible
    println("problem infeasible")

end
```

7.11 Implicit description

In [ ]:
```
c = [1, 3]
d = [14, 12, 12]
T = [1  1 ; -2  3; 2  -1]
n,m = size(T)

lp = Model( solver = GLPKSolverLP() )
@variable(lp, x[1:m] >= 0)
@objective(lp, Max, sum(c[j]*x[j] for j=1:m) )
@constraint(lp, cst[i=1:n], sum(T[i,j]*x[j] for j=1:m) <= d[i])
print(lp)
```

7.12 Exercice

For the unidimensional 01 knapsack problem,

$$z = \max \left\{ px \mid wx \leq c,\ x \in \{0,1\}^n \right\}$$

with

$$\begin{aligned} n &= 5 \\ p &= (5,3,2,7,4) \\ w &= (2,8,4,2,5) \\ c &= 10 \end{aligned}$$

compute the optimal solution.

---

8. vOptSolver

[https://github.com/vOptSolver (https://github.com/vOptSolver)](https://github.com/vOptSolver)

An production of the ANR/DFG-14-CE35-0034-01 research project (Feb/2015-Jan/2019)

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).

ROADEF 2018 - Xavier Gandibleux et Anthony Przybylski, *Algorithmes de branch-and-bound multiobjectif et vOptSolver* (in French): [https://youtu.be/jH3AftHsG2s?t=1725 (https://youtu.be/jH3AftHsG2s?t=1725)](https://youtu.be/jH3AftHsG2s?t=1725)

8.1 Overview

- an ecosystem for modeling and solving multiobjective linear optimization problems (MOCO, MOIP, MOMIP, MOLP)
- it deals with structured and non-structured optimization problems with at least two objectives
- it integrates several specific and generic exact algorithms for computing efficient solutions

- Designed for solving, research, and pedagogic needs
- Natural and intuitive use for mathematicians, informaticians, engineers
- Efficient, flexible, evolutive solver

- Aims to be easy to formulate a problem, to provide data, to solve a problem, to collect the outputs, to analyze the solutions
- Free, open source (MIT licence), multi-platform, reusing existing specifications
- Using usual free (GLPK, Clp/Cbc) and commercial (GUROBI, CPLEX) MILP solvers

8.2 The vOptSpecific package

Multiobjective structured problems / Application Programming Interface (API):

Problem	Description	API	src	Reference
LAP	Linear Assignment Problem	**2LAP2008**	C	Przybylski2008
OSP	One machine Scheduling Problem	**2OSP1980**	Julia	Wassenhove1980
UKP	01 Unidimensional knapsack problem	**2UKP2010**	Julia	Jorge2010
UMFLP	Uncapacitated Mixed variables Facility Location Problem	**2UMFLP2016**	C++	Delmee2017
PATHS	*forthcoming*			
UDFLP	*forthcoming*			
SSCFLP	*project*			
CFLP	*project*			
MKP	*project*			

Input:

- Direct: Julia and the API
- Files: Ad-hoc problem format (2LAP, 2OSP, 2UKP, 2UFLP)

Output:

- Ad-hoc problem format (2LAP, 2OSP, 2UKP, 2UFLP)

Platforms:

- macOS, linux. Project: windows

8.3 Getting vOptSpecific (v1.0 compliant with Julia v1.x)

In [ ]:
```
#using Pkg
#Pkg.add("vOptSpecific")
```

In [ ]:
```
using vOptSpecific
```

8.4 The vOptGeneric package

Multiobjective non-structured problems / Algebraic Language (JuMP extended to multiple objectives)

Problem	Description	Output	Method	Parameter (if required)	Name
2-ILP	bi-objective Integer Linear Program	Y_N	**:epsilon**	step = *realValue*	ϵ-constraint
2-ILP	bi-objective Integer Linear Program	Y_N	**:chalmet** or **:Chalmet**	step = *realValue*	Chalmet
2-ILP	bi-objective Integer Linear Program	Y_{SN}	**:dicho** or **:dichotomy**	(none)	Aneja & Nair
p-ILP	multi-objective Integer Linear Program	Y_{lex}	**:lex** or **:lexico**	(none)	Lexicographic
{2,3}-LP	(forthcoming)				
3-ILP	(project)				
{2,3}-MILP	(project)				

using Julia and JuMP-extendedMO, with GLPK or Clp/Cbc or CPLEX or GUROBI

Input:

- Direct: Julia and JuMP-extendedMO
- Files : (1) Standard MOP format (ILP, MILP, LP) and (2) Specific problem format (MILP)

Output:

- Standard 2MOP format (ILP, MILP, LP)

Platforms:

- macOS, linux, windows

8.5 Getting vOptGeneric (v0.2 compliant with Julia v1.x)

In [ ]:
```
#using Pkg
#Pkg.add("vOptGeneric")
```

In [ ]:
```
using vOptGeneric
```

8.6 Example

For the bi-objective unidimensional 01 knapsack problem,

$$\max \left\{ (p^1 x,\ p^2 x) \mid wx \le c,\ x \in \{0,1\}^n \right\}$$

with

$$\begin{aligned} n &= 5 \\ p^1 &= (\ 6,\ 4,\ 4,\ 4,\ 3) \\ p^2 &= (12, 10,\ 5,\ 3,\ 1) \\ w &= (\ 8,\ 6,\ 4,\ 3,\ 2) \\ c &= 15 \end{aligned}$$

compute Y_N, the set of non-dominated points.

In [ ]:
```
n  = 5
p1 = [ 6,  4,  4,  4,  3]
p2 = [12, 10,  5,  3,  1]
w  = [ 8,  6,  4,  3,  2]
c  = 15
```

8.4 Creating and solving the example with vOptSpecific

In [ ]:
```
#using vOptSpecific

mokp = set2UKP( n, p1, p2, w, c )
algo = UKP_Jorge2010()
```

In [ ]:
```
z1, z2, s, x = vSolve( mokp , algo )
```

In [ ]:
```
println( " i z1 z2 x s")
for i in 1:length(z1)
    println(" ",i," ",z1[i]," ",z2[i]," ",x[i]," ",s[i])
end
```

8.5 Creating and solving the example with vOptGeneric

In [ ]:
```
#using vOptGeneric

moip = vModel( solver = GLPKSolverMIP() )
@variable( moip , x[1:n] , Bin )
@addobjective( moip , Max, sum( p1[j]*x[j] for j=1:n ) )
@addobjective( moip , Max, sum( p2[j]*x[j] for j=1:n ) )
@constraint( moip , sum(w[j]*x[j] for j=1:n) <= c )
```

In [ ]:
```
solve( moip , method = :epsilon , step = 0.5 )
```

In [ ]:
```
printX_E( moip )
```

In [ ]:
```
Y_N = getY_N(moip)
for n = 1:length(Y_N)
    X = vOptGeneric.getvalue(x, n)
    print(findall( v->(v == 1.0), X))  # or print( findall( X .== 1.0 ) )
    println(" | z = ",Y_N[n])
end
```

In [ ]:
```
using PyPlot

z1, z2 = map(x -> x[1], Y_N), map(x -> x[2], Y_N)

PyPlot.title("Knapsack")
PyPlot.xlabel("\$z_1\$ to maximize") ; PyPlot.ylabel("\$z_2\$ to maximize")
grid()

plot(z1, z2, "bx", markersize = "8", label="\$Y_N\$")
legend(loc=1, fontsize="small")
show()
```

8.5 Saving and Parse-and-solve with vOptGeneric

In [ ]:
```
writeMOP(moip, "myRAMOOkp.mop")
```

In [ ]:
```
using vOptGeneric, Gurobi
my_moip = parseMOP("example.MOP", solver = GurobiSolver())
```

8.6 Exercice

Consider now the following bi-objective generalized assignment (2-GAP) problem:

$$
\begin{aligned}
&\left(\max \sum_{i=1}^{m} \sum_{j=1}^{n} p^1_{ij} x_{ij} \ , \ \max \sum_{i=1}^{m} \sum_{j=1}^{n} p^2_{ij} x_{ij} \right) \\
s.t \quad & \sum_{j=1}^{n} w_{ij} x_{ij} \leq b_i, \quad \forall i \in \{1, \dots, m\} \\
& \sum_{i=1}^{m} x_{ij} = 1, \quad \forall j \in \{1, \dots, n\} \\
& x_{ij} \in \{0, 1\}, \ \forall i \in \{1, \dots, m\}, \ \forall j \in \{1, \dots, n\}
\end{aligned}
$$

Generate an instance $m \times n$ with coefficients randomly generated as follow:

- $1 \leq p^1_{ij}, \ p^2_{ij}, \ w_{ij} \leq 10$
- $b_i = \lfloor \frac{\sum_{j=1}^{m} w_{ij}}{2} \rfloor$

and:

- compute Y_N, the set of non-dominated points
- plot Y_N
- compute UBS, an upper bound set of Y_N
- plot on a same figure Y_N and UBS

Answers to exercices

Exercice 3.9

Let $f : [a; b] \rightarrow \mathbb{R}$ a function strictly monotonic on the interval $[a, b]$. We suppose the equation $f(x) = 0$ has one and only one solution on the interval. Determine this value for a given precision using a dichotomic method.

In [ ]:
```
function dichotomy(a, b, f, ϵ)
    @assert b > a "Error: for a,b : b>a"
    x = (a+b)/2
    while ( b-a > ϵ ) && ( abs(f(x)) > ϵ )
        println(a, " ", x, " ", b)
        if ( f(a)*f(x) < 0 )
            b = x
        else
            a = x
        end
        x = (a+b)/2
    end
    return x
end
```

In [ ]:
```
a=-5; b=5; fx=x->x+1; ϵ=0.001
dichotomy(a, b, fx, ϵ)
```

Exercice 4.5

For the unidimensional 01 knapsack problem,

$$z = \max \left\{ px \mid wx \leq c,\ x \in \{0,1\}^n \right\}$$

with

$$\begin{gathered} n = 5 \\ p = (5,3,2,7,4) \\ w = (2,8,4,2,5) \\ c = 10 \end{gathered}$$

compute the linear relaxation.

In [ ]:
```
function computeLP_U01KP( p, w, c )

    n = length( p )
    xLP = zeros(Float64,n)

    # compute the utilities and reorder the items accordingly
    u = p ./ w
    reord = sortperm( u, rev=true )

    # identify the last item integrally selected
    zLP = 0 ; bar_c = c; s = 1
    while ( s <= n ) && ( bar_c - w[reord[s]]  >= 0 )
         xLP[reord[s]] = 1.0
         zLP = zLP + p[reord[s]]
         bar_c = bar_c - w[reord[s]]
         s = s + 1
    end
    s = s - 1

    # constraint not saturated => add the fractionnal part of the blocking item
valuated
    if ( bar_c > 0 ) && ( s < n )
        xLP[reord[s+1]] = bar_c / w[reord[s+1]]
        zLP = zLP + xLP[reord[s+1]] * p[reord[s+1]]
    end

    for i = 1:n
        @printf(" %d (%d %d %.2f) %4.2f \n", reord[i], p[reord[i]], w[reord[i]]
, u[reord[i]], xLP[reord[i]])
    end

    return zLP, xLP
end
```

In [ ]:
```
p = [ 5, 3, 2, 7, 4 ]
w = [ 2, 8, 4, 2, 5 ]
c = 10
computeLP_U01KP( p, w, c )
```

Exercice 7.12

For the unidimensional 01 knapsack problem,

$$z = \max \left\{ px \mid wx \leq c,\ x \in \{0,1\}^n \right\}$$

with

$$\begin{gathered} n = 5 \\ p = (5,3,2,7,4) \\ w = (2,8,4,2,5) \\ c = 10 \end{gathered}$$

compute the optimal solution.

In [ ]:
```
using Printf, JuMP, GLPK, GLPKMathProgInterface

p = [ 5, 3, 2, 7, 4 ]
w = [ 2, 8, 4, 2, 5 ]
c = 10
n = length(p)

kp = Model( solver = GLPKSolverMIP() ) # GLPK in mode MIP
@variable(kp, x[1:n], Bin)
@objective(kp, Max, sum(p[j]*x[j] for j=1:n))
@constraint(kp, sum(w[j]*x[j] for j=1:n) <= c)

status = solve(kp)

println("zOpt: ", getobjectivevalue(kp))
print("xOpt: ")
for i = 1:n
    @printf(" %1d|%1.0f ",i, JuMP.getvalue(x[i]))
end
```

Exercice 8.6

Consider now the following bi-objective generalized assignment (2-GAP) problem:

$$\left(\max \sum_{i=1}^{m} \sum_{j=1}^{n} p^1_{ij} x_{ij} \; , \; \max \sum_{i=1}^{m} \sum_{j=1}^{n} p^2_{ij} x_{ij} \right)$$

$$\begin{aligned} s.t \qquad & \sum_{j=1}^{n} w_{ij} x_{ij} \leq b_i, \quad \forall i \in \{1, \ldots, m\} \\ & \sum_{i=1}^{m} x_{ij} = 1, \quad \forall j \in \{1, \ldots, n\} \\ & x_{ij} \in \{0,1\}, \; \forall i \in \{1, \ldots, m\}, \; \forall j \in \{1, \ldots, n\} \end{aligned}$$

Generate an instance $m \times n$ with coefficients randomly generated as follow:

- $1 \leq p^1_{ij}, \; p^2_{ij}, \; w_{ij} \leq 10$
- $b_i = \lfloor \frac{\sum_{j=1}^{m} w_{ij}}{2} \rfloor$

and:

- compute Y_N, the set of non-dominated points
- plot Y_N
- compute UBS, an upper bound set of Y_N
- plot on a same figure Y_N and UBS

In [ ]:
```
m=3; n=10
p1 = rand(1:10,m,n)
p2 = rand(1:10,m,n)
w  = rand(1:10,m,n)
b  = Matrix{Int64}(undef,m,n)
for i=1:m
    b[i] = floor(Int64,sum(w[i,:])/2)
end
```

2-GAP with variables in $\{0, 1\}$ solved with the epsilon-constraint method using GLPK:

In [ ]:
```
using vOptGeneric, GLPK, GLPKMathProgInterface

GAP = vModel(solver=GLPKSolverMIP())
@variable(GAP, x[1:m, 1:n], Bin)
@addobjective(GAP, Max, sum(p1[i,j]*x[i,j] for i = 1:m, j = 1:n))
@addobjective(GAP, Max, sum(p2[i,j]*x[i,j] for i = 1:m, j = 1:n))
@constraint(GAP, [i=1:m], sum(w[i,j]*x[i,j] for j = 1:n) <= b[i])
@constraint(GAP, [j=1:n], sum(x[i,j] for i = 1:m) == 1)

solve( GAP , method = :epsilon , step = 0.5 )
```

In [ ]:
```
#printX_E( GAP )
Y_N = getY_N(GAP)
```

In [ ]:
```
using PyPlot

z1, z2 = map(x -> x[1], Y_N), map(x -> x[2], Y_N)

PyPlot.title("2-GAP")
PyPlot.xlabel("\$z_1\$ to maximize") ; PyPlot.ylabel("\$z_2\$ to maximize")
grid()

plot(z1, z2, "bx", markersize = "8", label="\$Y_N\$")
legend(loc=1, fontsize="small")
show()
```

2-GAP with variables in $[0, 1]$ solved with the Chalmet method (step of 1) using GLPK:

In [ ]:
```
GAP2 = vModel(solver=GLPKSolverLP())
@variable(GAP2, 0<= x[1:m, 1:n] <= 1)
@addobjective(GAP2, Max, sum(p1[i,j]*x[i,j] for i = 1:m, j = 1:n))
@addobjective(GAP2, Max, sum(p2[i,j]*x[i,j] for i = 1:m, j = 1:n))
@constraint(GAP2, [i=1:m], sum(w[i,j]*x[i,j] for j = 1:n) <= b[i])
@constraint(GAP2, [j=1:n], sum(x[i,j] for i = 1:m) == 1)

solve( GAP2 , method = :chalmet , step = 1 )
```

In [ ]:
```
Y_N = getY_N(GAP2)
```

In [ ]:
```
z12, z22 = map(x -> x[1], Y_N), map(x -> x[2], Y_N)

PyPlot.title("2-GAP")
PyPlot.xlabel("\$z_1\$ to maximize") ; PyPlot.ylabel("\$z_2\$ to maximize")
grid()

plot(z1, z2, "bx", markersize = "8", label="\$Y_N\$")
plot(z12, z22, "o", markersize = "5", label="\$LR(Y_N)\$")
legend(loc=1, fontsize="small")
show()
```

In [ ]:
```
vOptGeneric.getvalue(x, 2)
```