Workshop on Optimization Techniques for Data Science in Python and Julia

1. Introduction

Bogumił Kamiński

Agenda

Day 1: Jan 14, 2019; 9:00 to 16:00

1. Introduction to applications of optimization methods 9:00-9:50
2. Solving linear programming (LP) problems with Pyomo - case study 10:00-10:50
3. Solving mixed integer programming (MIP) problems with Pyomo - case study 11:00-12:00

Lunch 12:00-13:00

5. Solving nonlinear programming problems with Pyomo - case study 13:00-13:50
6. Introduction to Julia 14:00-16:00

Day 2 Jan 15, 2019; 9:00 to 16:00

6. Running optimization models with JuMP 9:00-9:50
7. Solving LP problems with JuMP - case study 10:00-10:50
8. Solving MIP problems with JuMP - case study 11:00-12:00

Lunch 12:00-13:00

9. Solving nonlinear programming problems with JuMP - case study 13:00-13:50
10. Advanced optimization approaches in Julia JuMP – case study 14:00-14:50
11. Summary: comparison of Pyomo and JuMP 15:00-16:00

References used

- Hart et al., Pyomo — Optimization Modeling in Python, 2nd ed, Springer, 2017
- Gueret et al.: Applications of optimization with Xpress-MP, Dash Optimization, 2007, https://www.researchgate.net/publication/248591052_Applications_of_optimization_with_Xpress_-_MP
- Kaminski: The Julia Express, http://bogumilkaminski.pl/files/julia_express.pdf
- Kamiński B., Szufel P., Julia 1.0 Programming Cookbook: Over 100 numerical and distributed computing recipes for your daily data science workflow, Packt 2018
- JuMP tutorial: http://www.juliaopt.org/JuMP.jl/v0.18/quickstart.html

Warm-up (Gueret et al., 2007, chap. 1.1)

A company makes two different sizes of boxwood chess sets. The small one requires 3 hours of work and the large one 2 hours. The company has four workers who each work 40 hours per week. The small chess set requires 1 kg of wood and the large one. Company can buy up to 200 kg of wood per week. When selling the small chess set company’s profit is 5$ and selling the large one yields 20$ of profit.

What should be a production plan of the company to maximize the profits?

Solving optimization problems

1. Mathematical formulation
2. Problem type identification
3. Software implementation
4. Solution

Mathematical formulation

1. Decision variables

2. Objective

3. Constraints

Mathematical formulation

1. Decision variables
   - x_s number of small boxes to produce each week
   - x_l number of large boxes to produce each week
2. Objective
3. Constraints

Mathematical formulation

1. Decision variables
   - x_s number of small boxes to produce each week
   - x_l number of large boxes to produce each week
2. Objective
   - maximize $5x_s + 20x_l$
3. Constraints

Mathematical formulation

1. Decision variables
   - x_s number of small boxes to produce each week
   - x_l number of large boxes to produce each week
2. Objective
   - maximize $5x_s + 20x_l$
3. Constraints
   - $3x_s + 2x_l \leq 160$ (hours worked)
   - $1x_s + 3x_l \leq 200$ (amount of boxwood)

Mathematical formulation

1. Decision variables
   - x_s number of small boxes to produce each week
   - x_l number of large boxes to produce each week
2. Objective
   - maximize $5x_s + 20x_l$
3. Constraints
   - $3x_s + 2x_l \leq 160$ (hours worked)
   - $1x_s + 3x_l \leq 200$ (amount of boxwood)
   - $x_s \geq 0, x_l \geq 0$ (non negativity)

Problem type identification

- Decision variables: real
- Objective: linear in variables
- Constraints: linear in variables

Problem type identification

- Decision variables: real
- Objective: linear in variables
- Constraints: linear in variables

Type of problem:
linear programming

Software implementation (chessset1.py)

```
from pyomo.environ import *

model = ConcreteModel(name="Chess set 1")
model.xs = Var(within=NonNegativeReals)
model.xl = Var(within=NonNegativeReals)
model.obj = Objective(expr= 5*model.xs + 20*model.xl, sense=maximize)
model.latehours = Constraint(expr= 3*model.xs + 2*model.xl <= 160)
model.boxwood = Constraint(expr= 1*model.xs+3*model.xl <= 200)
```

Solution (chessset1.py)

```
solver = SolverFactory("glpk")
results = solver.solve(model)
print(results)
print(value(model.xs), value(model.xl))
model.obj.pprint()
model.latehours.pprint()
model.boxwood.pprint()
```

Model specification is separated from solver selection

Solution (chessset1.py)

```
>>> print(results)

Problem:
- Name: unknown
  Lower bound: 1333.33333333333
  Upper bound: 1333.33333333333
  Number of objectives: 1
  Number of constraints: 3
  Number of variables: 3
  Number of nonzeros: 5
  Sense: maximize
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
  Error rc: 0
  Time: 0.06509137153625488
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Solution (chessset1.py)

```
>>> print(value(model.xs), value(model.xl))
0.0 66.6666666666667
>>> model.obj.pprint()
obj : Size=1, Index=None, Active=True
    Key  : Active : Sense    : Expression
    None :   True : maximize : 5*xs + 20*xl
>>> model.latehours.pprint()
latehours : Size=1, Index=None, Active=True
    Key  : Lower : Body        : Upper : Active
    None :  -Inf : 3*xs + 2*xl : 160.0 :   True
>>> model.boxwood.pprint()
boxwood : Size=1, Index=None, Active=True
    Key  : Lower : Body      : Upper : Active
    None :  -Inf : xs + 3*xl : 200.0 :   True
```

Software implementation (chessset2.py)

```
from pyomo.environ import *

model = ConcreteModel(name="Chess set 1")
model.xs = Var(within=NonNegativeIntegers)
model.xl = Var(within=NonNegativeIntegers)
model.obj = Objective(expr= 5*model.xs + 20*model.xl, sense=maximize)
model.latehours = Constraint(expr= 3*model.xs + 2*model.xl <= 160)
model.boxwood = Constraint(expr= 1*model.xs+3*model.xl <= 200)
```

Refined mathematical formulation

1. Decision variables
   - x_s **integer** number of small boxes to produce each week
   - x_l **integer** number of large boxes to produce each week
2. Objective
   - maximize $5x_s + 20x_l$
3. Constraints
   - $3x_s + 2x_l \leq 160$ (hours worked)
   - $1x_s + 3x_l \leq 200$ (amount of boxwood)
   - $x_s \geq 0, x_l \geq 0$ (non negativity)

Solution (chessset2.py)

```
>>> print(results)

Problem:
- Name: unknown
  Lower bound: 1330.0
  Upper bound: 1330.0
  Number of objectives: 1
  Number of constraints: 3
  Number of variables: 3
  Number of nonzeros: 5
  Sense: maximize
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 1
      Number of created subproblems: 1
  Error rc: 0
  Time: 0.08750557899475098
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Solution (chessset2.py)

```
>>> print(value(model.xs), value(model.xl))
2.0 66.0
>>> model.obj.pprint()
obj : Size=1, Index=None, Active=True
    Key  : Active : Sense    : Expression
    None :   True : maximize : 5*xs + 20*xl
>>> model.latehours.pprint()
latehours : Size=1, Index=None, Active=True
    Key  : Lower : Body        : Upper : Active
    None :  -Inf : 3*xs + 2*xl : 160.0 :   True
>>> model.boxwood.pprint()
boxwood : Size=1, Index=None, Active=True
    Key  : Lower : Body      : Upper : Active
    None :  -Inf : xs + 3*xl : 200.0 :   True
```

The power of Pyomo (chessset3.py): weave Python into problem specification

```
from pyomo.environ import *

def objective(m):
    return 5*m.xs + 20*m.xl

def latehours(m):
    return 3*m.xs + 2*m.xl <= 160

def boxwood(m):
    return 1*m.xs + 3*m.xl <= 200

constraints = [latehours, boxwood]

def constraint(m, idx):
    return constraints[idx](m)
```

```
def chess_set(onlyint):
    typ = NonNegativeIntegers if onlyint else NonNegativeReals
    model = ConcreteModel(name="Chess set")
    model.xs = Var(within=typ)
    model.xl = Var(within=typ)
    model.obj = Objective(rule=objective, sense=maximize)
    model.constr = Constraint([0,1], rule=constraint)
    solver = SolverFactory("glpk")
    solver.solve(model)
    return model
```

The power of Pyomo (chessset3.py): weave Python into problem specification

```
>>> res = chess_set(False)
>>> res.pprint()
1 Set Declarations
    constr_index : Dim=0, Dimen=1, Size=2, Domain=None, Ordered=False, Bounds=(0, 1)
        [0, 1]

2 Var Declarations
    xl : Size=1, Index=None
        Key  : Lower : Value            : Upper : Fixed : Stale : Domain
        None :     0 : 66.6666666666667 :  None : False : False : NonNegativeReals
    xs : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :   0.0 :  None : False : False : NonNegativeReals

1 Objective Declarations
    obj : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : 5*xs + 20*xl

1 Constraint Declarations
    constr : Size=2, Index=constr_index, Active=True
        Key : Lower : Body        : Upper : Active
          0 :  -Inf : 3*xs + 2*xl : 160.0 :   True
          1 :  -Inf :   xs + 3*xl : 200.0 :   True

5 Declarations: xs xl obj constr_index constr
```

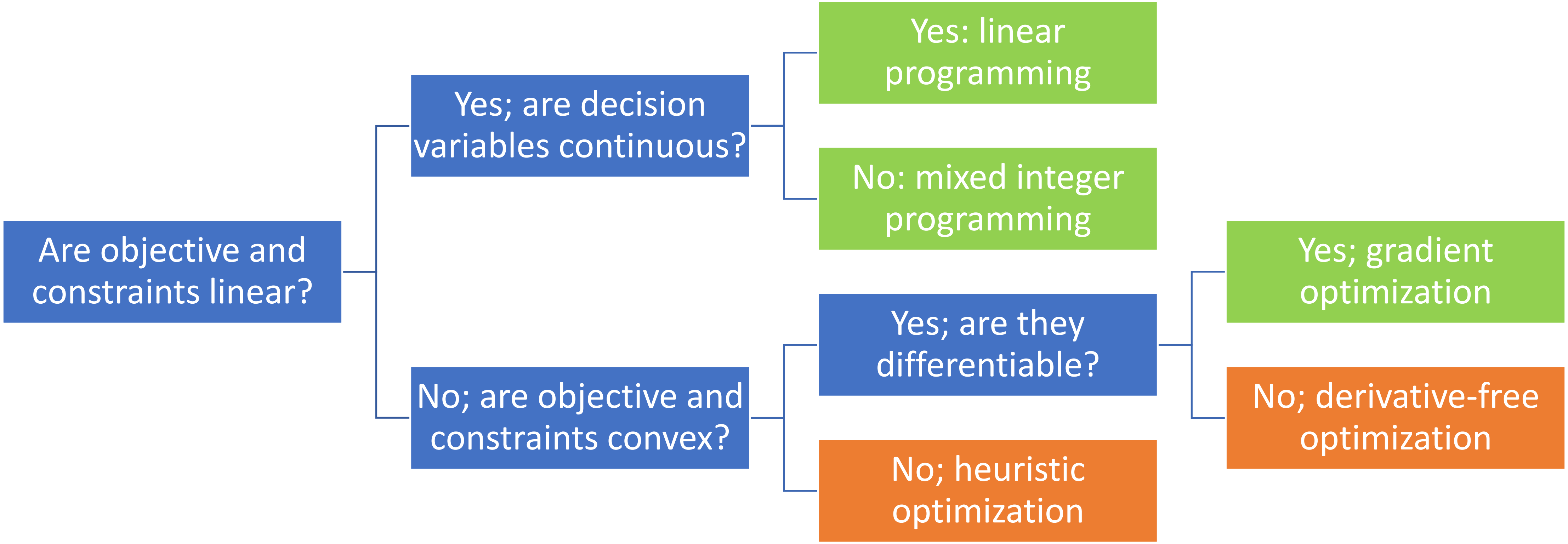
The power of Pyomo (chessset3.py): weave Python into problem specification

```
>>> res = chess_set(True)
>>> res.pprint()
1 Set Declarations
    constr_index : Dim=0, Dimen=1, Size=2, Domain=None, Ordered=False, Bounds=(0, 1)
        [0, 1]

2 Var Declarations
    xl : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :  66.0 :  None : False : False : NonNegativeIntegers
    xs : Size=1, Index=None
        Key  : Lower : Value : Upper : Fixed : Stale : Domain
        None :     0 :   2.0 :  None : False : False : NonNegativeIntegers

1 Objective Declarations
    obj : Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : maximize : 5*xs + 20*xl

1 Constraint Declarations
    constr : Size=2, Index=constr_index, Active=True
        Key : Lower : Body        : Upper : Active
          0 :  -Inf : 3*xs + 2*xl : 160.0 :   True
          1 :  -Inf :   xs + 3*xl : 200.0 :   True

5 Declarations: xs xl obj constr_index constr
```

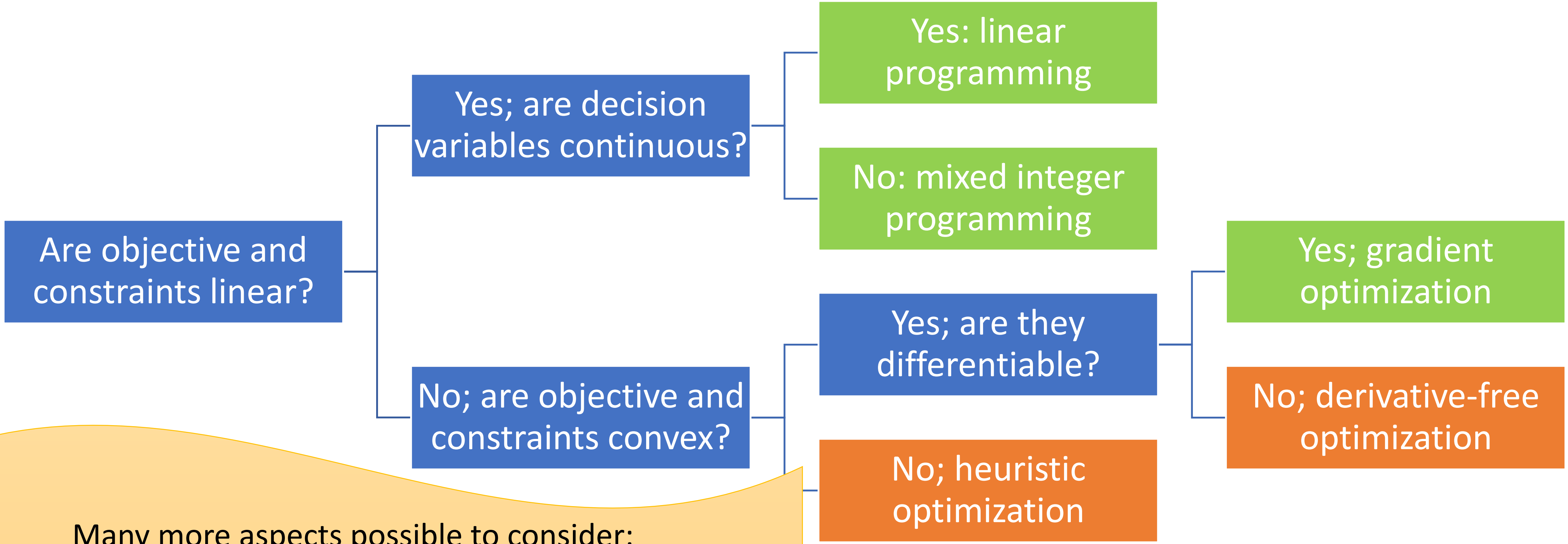
Self-check task

What would change if the company started to plan yearly instead of weekly (assume that a quarter consists of 51 weeks of work and 1 week of technical break).

A basic classification of optimization problems



A basic classification of optimization problems



Many more aspects possible to consider:
multiple objectives, robust optimization, stochastic optimization, equilibrium constraints, constraint programming
(see https://neos-guide.org/optimization-tree)

Different solvers are able to handle different classes of problems

- GLPK, open source (https://www.gnu.org/software/glpk/)
  - Linear programming
  - Mixed integer programming
- IPOPT, open source (https://projects.coin-or.org/Ipopt)
  - Constrained nonlinear, twice differentiable (possibly non-convex, finds local extrema)

- CBC, open source; alternative to GLPK
- CPLEX, Gurobi: commercial alternatives to GLPK
- BARON: commercial, global optimization
- MiniZinc: open source, constraint programming

(a more comprehensive review can be found at https://aimms.com/english/developers/resources/solvers/#lp-and-mip-solver-features)