Introduction

Brandon Moretz

15th May 2020

Simple Linear Optimization

```
# optimization framework
using JuMP;

# solvers
using CPLEX; using Gurobi; using Cbc;
```

Linear Programs

A straight forward approach to linear programming problems:

Given,

`\frac{n!}{k!(n - k)!} = \binom{n}{k}`

$ max ; x_1 + 2x_2 + 5x_3 $

subject to,

$ \begin{aligned} -x_1 + x_2 + 3x_3 -5 \ x_1 3x_2 - 7x_3 10 \ 0 x_1 10 \ x_2 0 \ x_3 0 \end{aligned} $

In Julia:

```
# optimization model
model = Model(Gurobi.Optimizer)

# variables
@variable(model, 0 <= x1 <= 10)
@variable(model, x2 >= 0)
@variable(model, x3 >= 0)

# objective
@objective(model, Max, x1 + 2x2 + 5x3)

@constraint(model, constraint1, -x1 + x2 + 3x3 <= -5)
@constraint(model, constraint2, x1 + 3x2 - 7x3 <= 10)
```

```
# take a peek
display(model)
```

```
A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: GenericAffExpr{Float64,VariableRef}
`GenericAffExpr{Float64,VariableRef}`-in-`MathOptInterface.LessThan{Float64
}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
`VariableRef`-in-`MathOptInterface.LessThan{Float64}`: 1 constraint
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Gurobi
Names registered in the model: constraint1, constraint2, x1, x2, x3
```

```
optimize!(model)

values = [ JuMP.value(x1), JuMP.value(x2), JuMP.value(x3) ]

display(values)
```

```
3-element Array{Float64,1}:
 10.0
  2.1875
  0.9375
```

```
JuMP.dual(constraint1)
JuMP.dual(constraint2)
```

```
-0.06249999999999989
```

Alternative

```
# model
m2 = Model(Gurobi.Optimizer)

# non-zero constraints
@variable(m2, x[1:3] >= 0)
```

```
3-element Array{VariableRef,1}:
 x[1]
 x[2]
 x[3]
```

$max \sum_{i=1}^{3} c_i x_i$

```
# coefficents
c = [1; 2; 5]
@objective(m2, Max, sum( c[i]*x[i] for i = 1:3))
```

```
x[1] + 2 x[2] + 5 x[3]
```

Matrix Notation: $Ax \leq b$

```
A = [-1 1 3;
      1  3 -7]

b = [-5; 10]

@constraint(m2, # model
  constraint[j=1:2], # num rows
  sum( A[j,i] * x[i] for i=1:3) <= b[j] )

# boundary constraint
@constraint(m2, bound, x[1] <= 10)
```

```
bound : x[1] <= 10.0
```

Solve it

```
optimize!(m2)
```

Results

```
values = [ JuMP.value(x1), JuMP.value(x2), JuMP.value(x3) ]

display(values)
```

```
3-element Array{Float64,1}:
 10.0
  2.1875
  0.9375
```

```
JuMP.dual(constraint1)
JuMP.dual(constraint2)
```

```
-0.06249999999999989
```

Yet Another Way

```
m3 = Model(Gurobi.Optimizer)

c = [ 1; 2; 5]
A = [-1  1  3;
      1  3 -7]
b = [-5; 10]

index_x = 1:3
index_constraints = 1:2

@variable(m3, x[index_x] >= 0)

@objective(m3, Max, sum( c[i] * x[i] for i in index_x) )

@constraint(m3, constraint[j in index_constraints],
                sum( A[j, i] * x[i] for i in index_x) <= b[j] )

@constraint(m3, bound, x[1] <= 10)

optimize!(m3)

display(m3)
```

```
A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: GenericAffExpr{Float64,VariableRef}
`GenericAffExpr{Float64,VariableRef}`-in-`MathOptInterface.LessThan{Float64
}`: 3 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 3 constraints
Model mode: AUTOMATIC
CachingOptimizer state: ATTACHED_OPTIMIZER
Solver name: Gurobi
Names registered in the model: bound, constraint, x
```

```
println("Optimal Solutions:")
```

```
Optimal Solutions:
```

```
for i in index_x
  println("x[$i] = ", value(x[i]))
end
```

```
x[1] = 10.0
x[2] = 2.1875
x[3] = 0.9375
```

```
println("Dual Variables:")
```

```
Dual Variables:
```

```julia
for j in index_constraints
  println("dual[$j] = ", dual(constraint[j]))
end
```

```
dual[1] = -1.8124999999999998
dual[2] = -0.06249999999999989
```

Mixed Integer Liner Programming

$max\ x_1 + 2x_2 + 5x_3$

subject to,

$ \begin{aligned} -x_1 + x_2 + 3x_3 -5 \ x_1 3x_2 - 7x_3 10 \ 0 x_1 10 \ x_2 0 ; Integer \ x_3 0, 1 \end{aligned} $

```julia
m4 = Model(Gurobi.Optimizer)

@variable(m4, 0 <= x1 <= 10)
@variable(m4, x2 >= 0, Int)
@variable(m4, x3, Bin)

@objective(m4, Max, x1 + 2x2 + 5x3)

@constraint(m4, constraint1, -x1 + x2 + 3x3 <= -5)
@constraint(m4, constraint2, x1 + 3x2 - 7x3 <= 10)

display(m4)
```

```
A JuMP Model
Maximization problem with:
Variables: 3
Objective function type: GenericAffExpr{Float64,VariableRef}
`GenericAffExpr{Float64,VariableRef}`-in-`MathOptInterface.LessThan{Float64
}`: 2 constraints
`VariableRef`-in-`MathOptInterface.GreaterThan{Float64}`: 2 constraints
`VariableRef`-in-`MathOptInterface.LessThan{Float64}`: 1 constraint
`VariableRef`-in-`MathOptInterface.Integer`: 1 constraint
`VariableRef`-in-`MathOptInterface.ZeroOne`: 1 constraint
Model mode: AUTOMATIC
CachingOptimizer state: EMPTY_OPTIMIZER
Solver name: Gurobi
Names registered in the model: constraint1, constraint2, x1, x2, x3
```

```julia
optimize!(m4)

println("Optimal Solutions:")
```

```
Optimal Solutions:
```

```julia
for i in index_x
  println("x[$i] = ", value(x[i]))
end
```

```
x[1] = 10.0
x[2] = 2.1875
x[3] = 0.9375
```

```julia
println("Dual Variables:")
```

```
Dual Variables:
```

```
for j in index_constraints
  println("dual[$j] = ", dual(constraint[j]))
end
```

```
dual[1] = -1.8124999999999998
dual[2] = -0.062499999999999989
```