Quick JuMP Tutorial (Linear Programming)

Antonio Flores-Tlacuahuac

Energy Research Group

Tecnológico de Monterrey, México

August 26, 2019

Simple Linear Programming Problem

Let us try to solve the following LP[1] :

$$
\begin{aligned}
\underset{x_1,x_2,x_3}{\text{Maximize}} \quad & 5x_1 + 4x_2 + 3x_3 \\
\text{subject to} \quad & 2x_1 + 3x_2 + x_3 \leq 5 \\
& 4x_1 + x_2 + 2x_3 \leq 11 \\
& 3x_1 + 4x_2 + 2x_3 \leq 8 \\
& 0 \leq x_1 \leq 5 \\
& x_2 \geq 0 \\
& x_3 \leq 7
\end{aligned}
$$

[1] Linear Programming, V.Chvatal, W.H. Freeman and Company, 1983

Simple Linear Programming Problem...

- Load the algebraic language and solver to be used

```
using JuMP
using Clp
```

- Declare the name of your problem and solver to be used

```
my_model = Model(with_optimizer(Clp.Optimizer))
```

- Declare the decision variables as well as their bounds

```
@variable(my_model, 0 <= x1 <= 5)
@variable(my_model, x2 >= 0)
@variable(my_model, x3 <= 7)
```

Simple Linear Programming Problem...

- Append the objective function

```
@objective(my_model, Max, 5x1 + 4x2 + 3x3)
```

- Include constraints

```
@constraint(my_model, con1, 2x1 + 3x2 + x3 <= 5)
@constraint(my_model, con2, 4x1 + x2 + 2x3 <= 11)
@constraint(my_model, con3, 3x1 + 4x2 + 2x3 <= 8)
```

- Print formulation for visual checking

```
print(my_model)
```

- Solve the LP problem

```
optimize!(my_model)
```

Simple Linear Programming Problem...

- Query termination status

```
termination_status(my_model)
```

- Print optimal solution

```
obj_value = objective_value(my_model)
x1v = value(x1)
x2v = value(x2)
x3v = value(x3)
println(" Objective function value = ", obj_value)
println(" x1 = ", x1v)
println(" x2 = ", x2v)
println(" x3 = ", x3v)
```

Simple Linear Programming Problem...

- Full code

```
using JuMP
using Clp
my_model = Model(with_optimizer(Clp.Optimizer))
@variable(my_model, 0 <= x1 <= 5)
@variable(my_model, x2 >= 0)
@variable(my_model, x3 <= 7)
@objective(my_model, Max, 5x1 + 4x2 + 3x3)
@constraint(my_model, con1, 2x1 + 3x2 + x3 <= 5)
@constraint(my_model, con2, 4x1 + x2 + 2x3 <= 11)
@constraint(my_model, con3, 3x1 + 4x2 + 2x3 <= 8)
print(my_model)
optimize!(my_model)
termination_status(my_model)
obj_value = objective_value(my_model)
x1v = value(x1)
x2v = value(x2)
x3v = value(x3)
println(" Objective function value = ", obj_value)
println(" x1 = ", x1v)
println(" x2 = ", x2v)
println(" x3 = ", x3v)
```

Problem output

```
Max 5 x1 + 4 x2 + 3 x3
Subject to
 2 x1 + 3 x2 + x3 <= 5
 4 x1 + x2 + 2 x3 <= 11
 3 x1 + 4 x2 + 2 x3 <=  8
 0 <= x1 <= 5
 x2 >= 0
 x3 <= 7

OPTIMAL: TerminationStatusCode = 1
Objective function value = 12.999999999999998
x1 = 2.0
x2 = 0.0
x3 = 0.9999999999999996
```

Simple Linear Programming Problem

Let us try to solve the same past LP problem:

$$
\begin{aligned}
\underset{x_1,x_2,x_3}{\text{Maximize}} \quad & 5x_1 + 4x_2 + 3x_3 \\
\text{subject to} \quad & 2x_1 + 3x_2 + x_3 \leq 5 \\
& 4x_1 + x_2 + 2x_3 \leq 11 \\
& 3x_1 + 4x_2 + 2x_3 \leq 8 \\
& 0 \leq x_1 \leq 5 \\
& x_2 \geq 0 \\
& x_3 \leq 7
\end{aligned}
$$

Simple Linear Programming Problem...

- Load the algebraic language and solver to be used

```
using JuMP, Clp
```

- Declare the name of your problem and solver to be used

```
my_model = Model(with_optimizer(Clp.Optimizer))
```

- Set constant terms
  We collect the constant coefficients of the objective function:

$$5x_1 + 4x_2 + 3x_3$$

into the following 'c" vector:

$$c = [5; 4; 3]$$

set a constant terms matrix "A" and a "b" vector to hold the coefficients of the LHS and RHS constraints, respectively:

$$\begin{aligned} 2x_1 + 3x_2 + x_3 &\leq 5 \\ 4x_1 + x_2 + 2x_3 &\leq 11 \\ 3x_1 + 4x_2 + 2x_3 &\leq 8 \end{aligned}$$

as follows:

$$A = [2 \;\; 3 \;\; 1; 4 \;\; 1 \;\; 2; 3 \;\; 4 \;\; 2]$$

$$b = [5; \;\; 11; \;\; 8]$$

```
c = [5; 4; 3]
A = [2 3 1; 4 1 2; 3 4 2]
b = [5; 11; 8]
```

- Declare decision variables

```
@variable(my_model, x[1:3])
```

- Append the objective function

```
@objective(my_model, Max, sum(c[i]*x[i] for i=1:3))
```

- Include constraints

```
@constraint(my_model, constraint[j=1:3], sum(A[j,i]*x[i] for i=1:3) <= b[j])
@constraint(my_model, boundx1, 0 <= x[1] <=5)
@constraint(my_model, boundx2, x[2] >= 0)
@constraint(my_model, boundx3, x[3] <= 7)
```

- Print formulation for visual checking

```
print(my_model)
```

- Solve the LP problem

```
optimize!(my_model)
```

- Print optimal solution

```
termination_status(my_model)
obj_value = objective_value(my_model)
xsol = JuMP.value.(x)
println(" Objective function value = ", obj_value)
for i=1:3
println("x[$i] = ", xsol[i])
end
```

Simple Linear Programming Problem...

- Full code

```
using JuMP, Clp
my_model = Model(with_optimizer(Clp.Optimizer))
nc = 1:3
nx = 1:3
c = [5; 4; 3]
A = [2 3 1; 4 1 2; 3 4 2]
b = [5; 11; 8]
@variable(my_model, x[nx])
@objective(my_model, Max, sum(c[i]*x[i] for i in nx))
@constraint(my_model, constraint[j in nc], sum(A[j,i]*x[i] for i in nx) <= b[j])
@constraint(my_model, boundx1, 0 <= x[1] <=5)
@constraint(my_model, boundx2, x[2] >= 0)
@constraint(my_model, boundx3, x[3] <= 7)
print(my_model)
optimize!(my_model)
termination_status(my_model)
obj_value = objective_value(my_model)
xsol = JuMP.value.(x)
println(" Objective function value = ", obj_value)
for i=1:3
println("x[$i] = ", xsol[i])
end
```

Problem output

```
Max 5 x[1] + 4 x[2] + 3 x[3]
Subject to
 2 x[1] + 3 x[2] + x[3] <= 5
 4 x[1] + x[2] + 2 x[3] <=11
 3 x[1] + 4 x[2] + 2 x[3] <= 8
 0 <= x[1] <= 5
 x[2] >= 0
 x[3] <=7
 x[i] , i = {1,2,3}

OPTIMAL: TerminationStatusCode = 1
Objective function value = 12.999999999999998
x1 = 2.0
x2 = 0.0
x3 = 0.9999999999999996
```

Simple Linear Programming Problem

Finally, let us try to solve the same past LP problem:

$$
\begin{aligned}
\underset{x_1, x_2, x_3}{\text{Maximize}} \quad & 5x_1 + 4x_2 + 3x_3 \\
\text{subject to} \quad & 2x_1 + 3x_2 + x_3 \leq 5 \\
& 4x_1 + x_2 + 2x_3 \leq 11 \\
& 3x_1 + 4x_2 + 2x_3 \leq 8 \\
& 0 \leq x_1 \leq 5 \\
& x_2 \geq 0 \\
& x_3 \leq 7
\end{aligned}
$$

Simple Linear Programming Problem...

- Load the algebraic language and solver to be used

```
using JuMP, Clp
```

- Declare the name of your problem and solver to be used

```
my_model = Model(solver=ClpSolver())
```

- This time instead of declaring constant dimensions on vectors a, b and matrix A, and everywhere where they are deployed, we just declare once such dimensions for the number of decision variables (nx) and constraints (nc):

$$nx = 1, 2, 3$$

$$nc = 1, 2, 3$$

as follows:

```
nx = 1:3
nc = 1:3
```

- Declare decision variables

```
@variable(my_model, x[nx])
```

- Append the objective function

```
@objective(my_model, Max, sum(c[i]*x[i] for i in nx))
```

- Include constraints

```
@constraint(my_model, constraint[j in nc], sum(A[j,i]*x[i] for i in nx) <= b[j])
@constraint(my_model, boundx1, 0 <= x[1] <=5)
@constraint(my_model, boundx2, x[2] >= 0)
@constraint(my_model, boundx3, x[3] <= 7)
```

- Print formulation for visual checking

```
print(my_model)
```

- Print optimal solution

```
println("Status of the problem is: ", status_mymodel)
println("Objective function value = ", getobjectivevalue(my_model))
for i in nx
println("x[$i] = ", getvalue(x[i]))
end
```

- Full code

```
using JuMP, Clp
my_model = Model(solver=ClpSolver())
nx = 1:3
nc = 1:3
@variable(my_model, x[nx])
@objective(my_model, Max, sum(c[i]*x[i] for i in nx))
@constraint(my_model, constraint[j in nc], sum(A[j,i]*x[i] for i in nx) <= b[j])
@constraint(my_model, boundx1, 0 <= x[1] <=5)
@constraint(my_model, boundx2, x[2] >= 0)
@constraint(my_model, boundx3, x[3] <= 7)
print(my_model)
println("Status of the problem is: ", status_mymodel)
println("Objective function value = ", getobjectivevalue(my_model))
for i in nx
println("x[$i] = ", getvalue(x[i]))
end
```

Problem output

```
Max 5 x[1] + 4 x[2] + 3 x[3]
Subject to
 2 x[1] + 3 x[2] + x[3] <= 5
 4 x[1] + x[2] + 2 x[3] <= 11
 3 x[1] + 4 x[2] + 2 x[3] <= 8
 0 <= x[1] <= 5
 x[2] >= 0
 x[3] <= 7
 x[i] , i = {1,2,3}

Status of the problem is: Optimal
Objective function value = 12.999999999999998
x[1] = 2.0
x[2] = -0.0
x[3] = 0.9999999999999996
```