Getting Started with JuMP

Arpit Bhatia

June 5, 2019

This tutorial is aimed at providing a quick introduction to writing JuMP code. It assumes familiar with basic optimization and the notion of an AML.

1 What is JuMP?

JuMP ("Julia for Mathematical Programming") is an open-source modeling language that is embedded in Julia. It allows users to users formulate various classes of optimization problems (linear, mixed-integer, quadratic, conic quadratic, semidefinite, and nonlinear) with easy-to-read code. These problems can then be solved using state-of-the-art open-source and commercial solvers. JuMP also makes advanced optimization techniques easily accessible from a high-level language.

2 Installing JuMP

JuMP is a package for Julia. From Julia, JuMP is installed by using the built-in package manager.

```
import Pkg
Pkg.add("JuMP")
```

3 A Complete Example

Let's try to solve the following linear programming problem by using JuMP and GLPK (a linear and mixed integer programming solver). We will first look at the complete code to solve the problem and then go through it step by step.

$$\begin{aligned} \min \quad & 12x + 20y \\ \text{subject to} \quad & 6x + 8y \geq 100 \\ & 7x + 12y \geq 120 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$

```
using JuMP
using GLPK

model = Model(with_optimizer(GLPK.Optimizer))
@variable(model, x >= 0)
@variable(model, y >= 0)
@constraint(model, 6x + 8y >= 100)
@constraint(model, 7x + 12y >= 120)
@objective(model, Min, 12x + 20y)

optimize!(model)

@show value(x)
@show value(y)
@show objective_value(model)
```

```
value(x) = 14.999999999999993
value(y) = 1.2500000000000047
objective_value(model) = 205.0
205.0
```

4 Step by Step JuMP Code

Once JuMP is installed, to use JuMP in your programs, we just need to write-

```
using JuMP
```

We also need to include a Julia package which provides an appropriate solver. We want to use GLPK.Optimizer here which is provided by the GLPK.jl package.

```
using GLPK
```

A model object is a container for variables, constraints, solver options, etc. Models are created with the Model() function. The `with_optimizer` syntax is used to specify the optimizer to be used which is GLPK in this case.

```
model = Model(with_optimizer(GLPK.Optimizer))
```

A variable is modelled using `@variable(name of the model object, variable name and bound, variable type)`. The bound can be a lower bound, an upper bound or both. If no variable type is defined, then it is treated as real.

```
@variable(model, x >= 0)
@variable(model, y >= 0)
```

A constraint is modelled using `@constraint(name of the model object, constraint)`.

```
@constraint(model, 6x + 8y >= 100)
@constraint(model, 7x + 12y >= 120)
```

The objective is set in a similar manner using `@objective(name of the model object, Min/Max, function to be optimized)`

```
@objective(model, Min, 12x + 20y)
```

To solve the optimization problem, we call the optimize function.

```
optimize!(model)
```

Let's now check the value of objective and variables.

```
@show value(x)
```

```
value(x) = 14.999999999999993
```

```
@show value(y)
```

```
value(y) = 1.2500000000000047
```

```
@show objective_value(model)
```

```
objective_value(model) = 205.0
205.0
```