BEE 4750 Lab 3: Linear Programming with JuMP

2025-06-12

> Due Date
>
> Wednesday, 10/16/24, 9:00pm

Setup

The following code should go at the top of most Julia scripts; it will load the local package environment and install any needed packages. You will see this often and shouldn't need to touch it.

```
import Pkg
Pkg.activate(".")
Pkg.instantiate()
```

```
using JuMP # optimization modeling syntax
using HiGHS # optimization solver
```

Overview

In this lab, you will write and solve a resource allocation example using `JuMP.jl`. `JuMP.jl` provides an intuitive syntax for writing, solving, and querying optimization problems.

`JuMP` requires the loading of a solver. [Each supported solver works for certain classes of problems, and some are open source while others require a commercial license]. We will use the `HiGHS` solver, which is open source and works for linear, mixed integer linear, and quadratic programs.

In this lab we will walk through the steps involved in coding a linear program in HiGHS, solving it, and querying the solution.

Exercise (3 points)

Your task is to decide how much lumber to produce to maximize profit from wood sales. You can purchase wood from a managed forest, which consists of spruce (320,000 bf) and fir (720,000 bf). Spruce costs \$0.12 per bf to purchase and fir costs \$0.08 per bf.

At the lumber mill, wood can be turned into plywood of various grades (see Table 1 for how much wood of each type is required for and the revenue from each grade). Any excess wood is sent to be recycled into particle board, which yields no revenue for the mill.

Table 1: Wood inputs and revenue by plywood grade. S refers to spruce inputs, F fir inputs.

Plywood Grade	Inputs (bf/bf plywood)	Revenue (\$/1000 bf)
1	0.5 (S) + 1.5 (F)	400
2	1.0 (S) + 2.0 (F)	520
3	1.5 (S) + 2.0 (F)	700

First, we need to identify our decision variables. While there are several options, we will use G_i, the amount of each grade the mill produces (in 1000 bf).

Using these decision variables, formulate a linear program to maximize the profit of the mill subject to the supply constraints on spruce and fir.

JuMP Syntax

The core pieces of setting up a `JuMP` model involve specifying the model and adding variables, the objective, and constraints. At the most simple level, this syntax looks like this:

```
m = Model(HiGHS.Optimizer)
@variable(m, lb <= x <= ub) # if you do not have upper or lower bounds,
 ↪  you can drop those accordingly
@variable(m, lb <= y <= ub)
@objective(m, Max, 100x + 250y) # replace Max with Min depending on the
 ↪  problem
@constraint(m, label, 6x + 12y <= 80) # replace "label" with some
 ↪  meaningful string you would like to use later to query shadow prices,
 ↪  or drop it
```

You can add more constraints or more variables as needed.

Using Array Syntax

You can set up multiple variables or constraints at once using array syntax. For example, the following are equivalent:

```
@variable(m, G1 >= 0)
@variable(m, G2 >= 0)
@variable(m, G3 >= 0)
```

and

```
@variable(m, G[1:3] >= 0)
```

You can also set up multiple constraints using arrays of coefficients and/or bounds. For example:

```
I = 1:3
d = [0; 3; 5]
@constraint(m, demand[i in I], G[i] >= d[i])
```

`JuMP` is finicky about changing objects and constraints, so I recommend setting up all of the model syntax in one notebook cell, which is what we will do here.

```
forest_model = Model(HiGHS.Optimizer) # initialize model object
@variable(forest_model, G[1:3] >= 0) # non-negativity constraints
# uncomment the following lines and add the objective and constraints as
 ↪  needed for the model
# @objective(forest_model, )
# @constraint(forest_model, )
print(forest_model) # this outputs a nicely formatted summary of the model so
 ↪  you can check your specification
```

```
Feasibility
Subject to
 G[1]   0
 G[2]   0
 G[3]   0
```

Next, to optimize, use the `optimize!()` function:

```
optimize!(forest_model)
```

```
Running HiGHS 1.7.2 (git hash: 5ce7a2753): Copyright (c) 2024 HiGHS under MIT licence terms
Coefficient ranges:
  Cost   [0e+00, 0e+00]
  Bound  [0e+00, 0e+00]
Solving LP without presolve, or with basis, or unconstrained
Solving an unconstrained LP with 3 columns
Model   status      : Optimal
Objective value     :  0.0000000000e+00
HiGHS run time      :          0.00
```

You should get confirmation that a solution was found; if one was not, there's a chance something was wrong with your model formulation.

To find the values of the decision variables, use `value()` (which can be broadcasted over variable arrays):

```
@show value.(G);
```

```
value.(G) = [0.0, 0.0, 0.0]
```

Similarly, `objective_value()` finds the optimal value of the objective:

```
@show objective_value(forest_model);
```

```
objective_value(forest_model) = 0.0
```

Finally, we can find the dual values of the constraints with `shadow_price()`. Do this for the constraints in your model using the block below.

```
# @show shadow_price(name_of_constraint);
```

`JuMP` also lets you evaluate other expressions that you might be interested in based on the solutions. For example, you can use the following block to calculate the total amount of plywood the mill would produce under the optimal solution:

```
@expression(forest_model, total_plywood, sum(G))
@show value.(total_plywood);
```

```
value.(total_plywood) = 0.0
```

References

Put any consulted sources here, including classmates you worked with/who helped you.