BEE 4750 Lab 3: Linear Programming with JuMP

Name: Jonathan Marcuse

ID: jrm564

> **Due Date**
>
> Friday, 10/13/23, 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.

In [ ]:
```
import Pkg
Pkg.activate(".")
Pkg.instantiate()
```

In [ ]:
```
using JuMP # optimization modeling syntax
using HiGHS # optimization solver
using Plots # plotting
```

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.

For an example of using `JuMP.jl` to solve linear programs, see the relevant tutorial on the class website.

Free free to delete some of the illustrative cells and code blocks in your notebook as you go through and solve the lab problems...this might help reduce some potential confusion while grading about what your answer is.

Introduction

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. This resource allocation problem is diagrammed in Figure 1.

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

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

Problem 1: Problem Formulation (5 points)

In this problem, you will go through the steps of formulating a linear program for this problem.

Problem 1.1 (1 point)

What are your decision variables? Clearly define your notation, including what variables you are using, what they mean, and what their units are.

My decision variable will be how much wood in total is allocated towards producing each type of plywood grade. The variable will be P, which will be a vector of length 3 with each value being the corresponding amount of wood. P[1] will be the amount going towards grade 1 plywood, P[2] for grade 2 and P[3] for grade 3. The units are bf of wood.

Problem 1.2 (1 point)

Derive your objective function. Support your function with justifications and/or equations as necessary. You will not receive credit just for the function alone.

The objective function will be based on the total amount of wood inputted into each grade of production. We know from the table earlier that for grade 1 id we input 2000bf of raw wood we get 1000bf of grade 1 plywood and therefore 400 dollars. Grade 2 produces 520 dollars for every 3000bf of raw wood and grade 3 produces 700 dollars for every 3500bf of raw wood. The ratios of wood necessary for each one produced is also important for determining costs. For grade 1 the ratio of spruce to fir is 1:3, for grade 2 it is 1:2 and for grade 3 it is 3:4.

The cost of each raw wood material must also be subtracted with these appropriate ratios and the fraction of total inputted wood for each grade is multiplied by the decision variable for the amount ofwood entering each plywood production process.

Therefore the objective function will be 400(P[1]/2000) +520(P[2]/3000) +700(P[3]/3500) -0.12((1/4)P[1]+(1/3)P[2]+(3/7)P[3]) -0.08((3/4)P[1]+(2/3)P[2]+(4/7)P[3])

Which simplifies to: 0.11P[1]+0.08[2]+0.10286P[3]

Problem 1.3 (2 point)

Derive any needed constraints. Support your function with justifications and/or equations as necessary. You will not receive credit just for the final constraints alone.

Constraints will include the total amount of wood in the forest and how the sum of each element in the P vector for total raw wood inputs contribute to each type of wood based on fixed ratios.

The total sum of (1/4)P[1]+(1/3)P[2]+(3/7)P[3] must not exceed 320000bf and the total sum of all elements in P minus this amount must not exceed 720000 bf.

The ratios from above come from the ratios in the original prompt. Since the ratio is 1:3 for grade 1, 1/4 of the inputted wood must be spruce. For grade 2 the ratio is 1:2 so 1/3 of the inputted wood must be spruce. And finally, for grade 3 the ratio is 3:4 so 3/7 of the total inputted wood must be spruce. That is where the contraint equation for the total amount of spruce comes from, and then the total amount of wood in variable P minus that amount must be Spruce.

Problem 1.4 (1 point)

Put this optimization problem in mathematical programming form. For an example of the syntax for this, see lines 82–91 here.

$$
\begin{aligned}
\max \quad & 0.11000000000000001P_1 + 0.08000000000000003P_2 + 0.102857142857 \\
\text{Subject to} \quad & 0.25P_1 + 0.3333333333333333P_2 + 0.42857142857142855P_3 \leq 320000 \\
& 0.75P_1 + 0.6666666666666666P_2 + 0.5714285714285714P_3 \leq 720000 \\
& P_1 \geq 0 \\
& P_2 \geq 0 \\
& P_3 \geq 0
\end{aligned}
$$

Problem 2: Find the Solution (5 points)

Problem 2.1 (2 points)

Code your linear program using `JuMP`. Feel free to consult the website's `JuMP` tutorial for syntax help. The keys:

1. Initialize your model with a solver; in this case, we'll use the `HiGHS` solver, but there are other solvers listed here for different types of problems, some of which are open and some of which require a commercial license: https://jump.dev/JuMP.jl/stable/installation/#Supported-solvers:

   ```
   example_model = Model(HiGHS.Optimizer)
   ```
2. Define variables with syntax like

   ```
   @variable(example_model, 1 >= example_x >= 0)
   ```
   This will create a variable `example_x` which is constrained to be between 0 and 1; you can leave off any of the bounds if a variable is unbounded in a particular direction. You can also add a vector of variables:

   ```
   T = 1:3 # define set to index variables
   @variable(example_model, 1 >= example_z[t in T] >= 0)
   ```
   which will create a vector of 3 variables `example_z[1]`, ..., `example_z[3]`, all of which are bounded between 0 and 1.

3. Add an objective with

   ```
   @objective(example_model, Max, example_x + sum(example_z))
   ```
   which will add an objective to maximize (replace with `Min` to minimize).

4. Add constraints:

   ```
   @constraint(example_model, constraint1, 2example_x +
   3*sum(example_z) <= 10)
   @constraint(example_model, constraint2, 5example_x -
   example_z[1] <= 2)
   ```
   which will name the constraints `constraint1` and `constraint2` (you should make yours more descriptive about what the constraint actually is). The value of adding a name is to facilitate later querying of shadow prices, which we will discuss later. You can also add a vector of constraints which have similar structure or rely on different elements of a data vector:

   ```
   A = [2; 4]
   b = [8; 12]
   I = 1:2 # set indices for constraint
   @constraint(example_model, vector_constraint[i in I], A[i] *
   sum(example_z) .<= b[i])
   ```
   You can also define matrices of constraints which depend on two index sets by generalizing this syntax, e.g.

   ```
   @constraint(example_model, matrix_constraint[i in I, j in J,
   ...])
   ```

> **Tip**
>
> Specifying higher-dimensional vectors and matrices of variables and constraints will be important when we start looking at more complex applications, so don't skip over this! You don't want to manually enter thousands of constraints to ensure hourly electricity demand is met...

Finally, you can (and probably should) `print` your model to make sure that you get something that looks like the equations that you wrote down (in a notebook, this will be nicely rendered):

```
print(example_model)
```

$$
\begin{aligned}
\max \quad & example\_x + example\_z_1 + example\_z_2 + example\_z_3 \\
\text{Subject to} \quad & 2example\_x + 3example\_z_1 + 3example\_z_2 + 3example\_z_3 \leq 10 \\
& 5example\_x - example\_z_1 \leq 2 \\
& 2example\_z_1 + 2example\_z_2 + 2example\_z_3 \leq 8 \\
& 4example\_z_1 + 4example\_z_2 + 4example\_z_3 \leq 12 \\
& example\_x \geq 0 \\
& example\_z_1 \geq 0 \\
& example\_z_2 \geq 0 \\
& example\_z_3 \geq 0 \\
& example\_x \leq 1 \\
& example\_z_1 \leq 1 \\
& example\_z_2 \leq 1 \\
& example\_z_3 \leq 1
\end{aligned}
$$

> **Define your entire model in one cell**
>
> `JuMP` has great and intuitive syntax, but it doesn't like re-defining variables or constraints once they've been set. I recommend putting all of your model-definition code (starting from `model = Model(...)`) for a particular optimization problem in a single notebook cell, so you can re-set up the entire problem with a single click when you want to make a change.

In [ ]:
```
#First I will define the wood model
wood_model = Model(HiGHS.Optimizer)

#Next I will define the Spruce (S) and Fir (F) variables
T=1:3
@variable(wood_model,P[t in T] >= 0)

#The objective function will be the sum of revenues
    #minus the cost of raw wood materials
@objective(wood_model, Max,
400*(P[1]/2000)
+520*(P[2]/3000)
```

```
+700*(P[3]/3500)
-0.12*((1/4)*P[1]+(1/3)*P[2]+(3/7)*P[3])
-0.08*((3/4)*P[1]+(2/3)*P[2]+(4/7)*P[3]))

#The total amount of wood resources will be contrained
    #based on availability of Spruce and Fir
@constraint(wood_model,Spruce_Resources,
(1/4)*P[1]+(1/3)*P[2]+(3/7)*P[3]<=320000)
@constraint(wood_model,Fir_Resources,
(3/4)*P[1]+(2/3)*P[2]+(4/7)*P[3]<=720000)

print(latex_formulation(wood_model))
```

$$
\begin{aligned}
\max \quad & 0.11000000000000001P_1 + 0.08000000000000003P_2 + 0.102857142857 \\
\text{Subject to} \quad & 0.25P_1 + 0.3333333333333333P_2 + 0.42857142857142855P_3 \leq 320000 \\
& 0.75P_1 + 0.6666666666666666P_2 + 0.5714285714285714P_3 \leq 720000 \\
& P_1 \geq 0 \\
& P_2 \geq 0 \\
& P_3 \geq 0
\end{aligned}
$$

Problem 2.2 (1 points)

Find the solution to your program and find the optimal values of the decision variables. Once you've defined your model, you can find the solution with `optimize!():

In [ ]:
```
optimize!(wood_model)
```

```
Running HiGHS 1.5.3 [date: 1970-01-01, git hash: 45a127b78]
Copyright (c) 2023 HiGHS under MIT licence terms
Presolving model
2 rows, 3 cols, 6 nonzeros
2 rows, 3 cols, 6 nonzeros
Presolve : Reductions: rows 2(-0); columns 3(-0); elements 6(-0) - Not reduc
ed
Problem not reduced by presolve: solving the LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0    -2.9285683924e-01 Ph1: 2(3); Du: 3(0.292857) 0s
          2     1.1200000000e+05 Pr: 0(0) 0s
Model   status      : Optimal
Simplex   iterations: 2
Objective value     :  1.1200000000e+05
HiGHS run time      :          0.00
```

> **What if I Get An Error?**
>
> If `optimize!()` throws an error, that's usually a sign that something is wrong with the formulation (for example, a variable might not be bounded or a constraint might not be specified correctly) or a typo in the model definition. Linear programs should be well behaved!

To find the values of variables after optimizing, use `value.()` (the broadcasting ensures this will work for vector-valued variables as well):

In [ ]:
```
value.(P)
```

```
1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, 1:3
And data, a 3-element Vector{Float64}:
 704000.0
      0.0
 336000.0
```

In [ ]:
```
objective_value(wood_model)
```

```
112000.00000000003
```

Problem 2.3 (1 point)

How would your profit change if you could buy 1,000 additional bf of spruce? You can answer this by getting the shadow price of a particular variable with:

In [ ]:
```
shadow_price(Spruce_Resources)*1000
# this is why we named the constraints when we defined them
```

```
79.99999999999999
```

The profit would increase by $80 if an additional 1000bf of spruce were added.

Problem 2.4 (1 point)

Would you prefer to have 2,000 additional bf of spruce or 1,000 additional bf of fir?

In [ ]:
```
display(shadow_price(Spruce_Resources)*2000)
display(shadow_price(Fir_Resources)*1000)
```

```
159.99999999999997
120.00000000000003
```

I would prefer 2000 bf of extra Spruce because it would increase profits by 160 dollars whereas 1000 bf of additional fir would increase profits by 120 dollars. The difference would be an extra 40 dollars if 2000 bf of spruce is chosen over 1000 bf of fir.

References

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