Introduction to JuMP.jl

Modeling for Mathematical Optimization in Julia

What is JuMP.jl?

JuMP (Julia for Mathematical Programming) is an open-source modeling language for mathematical optimization in Julia.

- **It's not a solver:** JuMP doesn't solve problems directly. It provides a clear, high-level syntax to *describe* the problem.
- **Interfaces with Solvers:** JuMP connects to dozens of state-of-the-art solvers (both commercial and open-source) like HiGHS, Gurobi, CPLEX, Ipopt, etc.
- **Algebraic Syntax:** You write mathematical equations much like you would on paper.
- **High Performance:** Being built in Julia, it benefits from the language's speed and composability.

The Anatomy of a JuMP Model

The workflow for solving any problem in JuMP follows these steps:

1. **Model:** Create a "container" for your problem.
2. **Solver:** Attach the tool that will solve the problem.
3. **Variables (`@variable`):** Declare the problem's unknowns.
4. **Objective (`@objective`):** Define the function you want to maximize or minimize.
5. **Constraints (`@constraint`):** Add the rules, limits, and equations that the variables must obey.
6. **Optimize (`optimize!`):** Tell the solver to solve the problem.
7. **Analyze the Results:** Check the solution status, optimal value, and variable values.

Our Example: The Farmer's Problem

A farmer has **100 hectares** of land and wants to decide how much **wheat (x)** and **barley (y)** to plant to maximize their profit.

Problem Data:

Crop	Profit (€/ha)	Fertilizer (kg/ha)	Pesticide (L/ha)
Wheat (x)	150	20	5
Barley (y)	220	35	3

Available Resources:

- Fertilizer: **2500 kg**
- Pesticide: **400 L**

Our Example: Mathematical Formulation

The goal is to find the values of `x` and `y` that solve:

Maximize (Profit):

$$150x + 220y$$

Subject to (Constraints):

1. **Land:** $x + y \leq 100$
2. **Fertilizer:** $20x + 35y \leq 2500$
3. **Pesticide:** $5x + 3y \leq 400$
4. **Non-negativity:** $x, y \geq 0$

This is a classic **Linear Programming (LP)** problem.

Hands-On: JuMP Implementation (1/3)

Let's translate the problem into JuMP code.

Step 1: Prepare the environment

First, we import the packages and create the model, connecting it to a solver. We'll use **HiGHS**, an excellent open-source solver for LP.

```
# Import the necessary packages
using JuMP
using HiGHS

# 1. Create an empty model
# 2. Attach the HiGHS solver
model = Model(HiGHS.Optimizer)
```

Adding the mathematical formulation (2/3)

This is where we add the core algebra. We use JuMP's macros (`@variable` , `@objective` , `@constraint`) to build the problem piece by piece.

```
# 3: Declare variables
@variable(model, x >= 0, base_name = "wheat")
@variable(model, y >= 0, base_name = "barley")

# 4: Define the objective function
@objective(model, Max, 150x + 220y)

# 5: Add the constraints
@constraint(model, c_land, x + y <= 100)
@constraint(model, c_fertilizer, 20x + 35y <= 2500)
@constraint(model, c_pesticide, 5x + 3y <= 400)
```

Optimize model and observe results (3/3)

With the model built, we call `optimize!` to run the solver and then use accessor functions to retrieve the solution.

```
# Print the model for review (optional)
print(model)

# 6: Optimize
optimize!(model)

# 7: Analyze the results
println("--- Optimization Results ---")
println("Solution status: ", termination_status(model))
println("Maximum Profit: ", objective_value(model))
println("\n--- Planting Plan ---")
println("Hectares of Wheat (x): ", value(x))
println("Hectares of Barley (y): ", value(y))
```