HW1 Solutions

February 15, 2022

Problem 1: Getting Started with Julia

Purpose

This document aims to get you set up:

1. **Julia**: the programming language.
2. **JuMP**: a Julia package that lets you express optimization models and pass them to solvers.
3. **Gurobi**: a popular commercial solver with free academic licensing, which is faster than the default open-source optimization solvers.
4. **IJulia**: a popular environment for writing and sharing Julia code, with a nice UI.
5. Additional (useful) Julia packages: e.g., the CSV.jl package for reading and manipulating CSV files and the Plots.jl package for creating fancy plots.

Task 1

Please download the most recent version of Julia. As of January 2022, the current stable release is v1.7.1. You can download here. To open Julia, navigate to your applications folder and select the Julia 1.7 icon. You can also open it by navigating to a terminal and typing ”julia”. (If you have multiple versions, you must create an alias for version 1.7 to be able to open it in a terminal.)

Because Julia is a relatively new language, it is constantly under development. It is possible to have multiple Julia versions on your computer without running into issues.

Julia Packages

We will be running a few Jupyter notebooks throught the course. IJulia basically hijacks Python’s Jupyter notebook, allowing us to run Julia code in line with text, math, and visualizations. You can add IJulia by running the following commands in a Julia session.

```
julia> using Pkg
julia> Pkg.add("IJulia")
```

The package manager provides a less verbose installation alternative.

```
julia> ] # Note (do not type this): a close square bracket opens a Julia
package manager prompt, i.e., "pkg>"
(@v1.7) pkg> add IJulia
```

You can exit the package manager by pressing delete/backspace.

While you're there, please install the following packages:

- DataFrames
- JuMP
- CSV
- Plots
- Random
- LinearAlgebra
- Printf

Task 2

Gurobi is a commercial solver with free academic licenses. They are a pain in the booty to install, but Gurobi is quite powerful and worth the hassle! Please follow these steps to get the software onto your computer.

1. Go to Gurobi's website and sign up for a free account.
2. The downloads page is here. Accept the license agreement and download the most recent version of the Gurobi optimizer (v9.5.0). Follow the installation instructions as prompted.
3. You will need an MIT IP address for step 4. If you are off campus, you can use the MIT VPN to connect to the network. (I would recommend only trying this after step 4 fails to work for you. Also this didn't work with the new MIT VPN when tried last, but does with the older one)
4. After you have downloaded the optimizer software, you must obtain an Academic license. The license eventually expires, so you will have to repeat these steps every so often. (Julia will notify you when your Gurobi license is about to expire or if it has already expired–you won't be able to solve your models until it has been renewed.) Navigate here and accept the conditions. Scroll to the bottom of the page and you should see something like this:

```
    grbgetkey xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Copy the command (not this one–the one at the bottom of your page with your actual license key!) and paste it into a terminal (not a Julia REPL). Follow the default installation instructions as prompted. Now your computer is allowed to use Gurobi. 5. Install the Gurobi wrapper in Julia. Before adding Gurobi to our Julia environment, we must point the wrapper to the location of the Gurobi Optimizer's installation. Open a Julia REPL and type the following. If you are on a Mac:

```
    julia> ENV["GUROBI_HOME"] = "/Library/gurobi950/mac64/"
```

Windows:

```
    julia> ENV["GUROBI_HOME"] = "C:\\Program Files\\gurobi950\\win64\\"
```

Linux/Unix:

```
ENV["GUROBI_HOME"] = "/opt/gurobi950/linux64/"
```

Now, regardless of operating system, run the following commands.

```
julia> using Pkg
julia> Pkg.add("Gurobi")
julia> Pkg.build("Gurobi")
```

A very common error is that the GUROBI_HOME environment variable is not properly configured. Feel free to reach out with any difficulties/questions.

If you encounter many obstacles, you can also install the package GLPK. This optimizer is not as powerful but does not require the installation of other software.

Testing Gurobi

To test that everything works correctly, enter the following code at the Julia prompt (you do not need to worry about what the code means, we will explain it in the Recitations)

```
using JuMP, Gurobi
m = Model(Gurobi.Optimizer)
@variable(m, 0 <= x <= 2)
@variable(m, 0 <= y <= 30)
@constraint(m, 1*x+5*y <= 3.0)
@objective(m, Max, 5*x+3*y)
optimize!(m)
println("Objective value:", objective value(m))
println("x =", value(x))
println("y =", value(y))
```

You should get an objective value of 10.6, $x = 2.0$, and $y = 0.2$.

Testing Installation

To ensure your installation environment works, please do the following. In a terminal, open Julia. Then run the following commands.

```
julia> using IJulia
julia> notebook()
```

Create a new notebook by selecting New > Julia 1.7.1. Run the following command in a cell:

```
using CSV, DataFrames, JuMP, Gurobi, LinearAlgebra, Plots, Random, Printf
```

The installation is successful if the cell does not return any errors (warnings are ok).

Submit a screenshot of the full web browser page depicting the notebook outputs and the Julia kernel in use.

Additional Resources

In Recitations, we will briefly introduce Julia; here are some additional introductory resources:

- A cheatsheet.
- Learn X in Y minutes, Julia edition.
- The official Julia manual
- The official JuMP manual
- The official Gurobi manual

Problem 2: Linear Optimization Fundamentals

Consider the family of 3 optimization problems:

$$\begin{aligned}
\underset{x,y}{\text{maximize}} \quad & f_i(x,y) \\
\text{subject to} \quad & x + y \geq 1 \\
& -x + 2y \leq 4 \\
& x + 2y \leq 8 \\
& -2x + y \geq -6 \\
& 0 \leq x \leq 3
\end{aligned}$$

with the objective functions defined as

- $f_1(x,y) = 4x + y$
- $f_2(x,y) = -4x + y$
- $f_3(x,y) = x^2 \exp(y)$

Since each of the problems share the same constraints they share the same feasible region. However due to their differing objectives, the optimal solutions may not be the same. For each of the optimization problems:

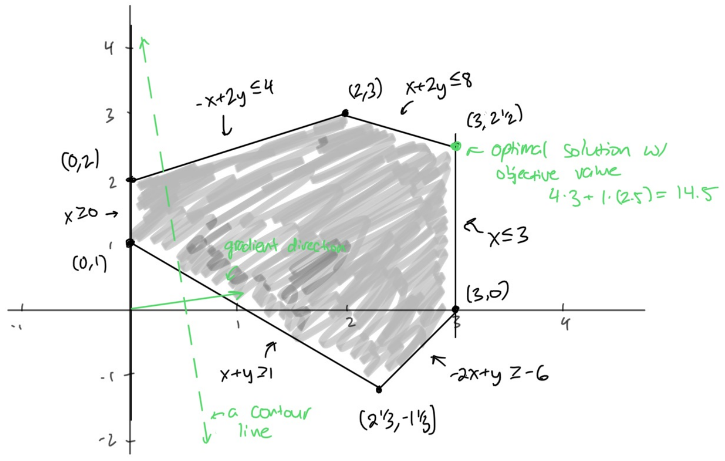
1. If it is a linear optimization problem, draw the problem geometrically. Graph a line showing the boundary of each of the constraints, shade the feasible region and label the vertices of the feasible region. Also draw the direction of the gradient and a contour line of the objective function. Finally, identify what the optimal solution and associated objective value of the problem is. If it isn't a linear optimization problem, write down any philosophical question.
2. If it is a linear optimization problem, use Julia, JuMP and Gurobi to solve the linear optimization problem. Turn in a copy of your code.

Notice that the only difference between the first and second objective functions is the sign of the coefficient of x. Briefly explain what the effect is on the optimal value of x and why this makes sense.

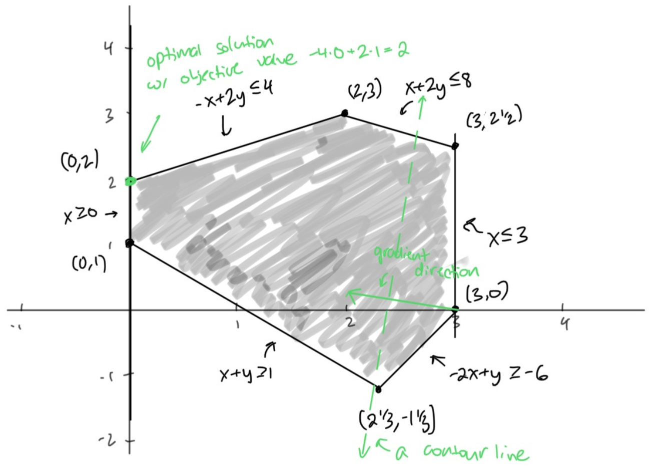
Solution

Here are the graphs:

$f_1(x,y) = 4x + y$



$f_2(x,y) = -4x + y$



$f_3(x,y) = x^2 e^y$

Where do we come from, where do we go?
Where do we come from, Cotton-Eyed Joe?

The solution code is in the included jupyter notebook.

Lastly, an answer to the final question is...

The optimal value of x changes from its maximum of 3 to it's minimum of 0. This makes sense because for the first objective function increasing x by 1 unit increases the objective value by 4 units. For the second objective function increasing x by 1 unit reduces the objective value by 4 units. Since the goal is to maximize the objective value, in the first problem we would like to make x as large as possible and in the second we would like to make it as small as possible.

Problem 3: The World Food Program

We consider a case study on the United Nations World Food Program (WFP). The case concerns an optimization model for the food assistance response by the WFP in which the food basket to be delivered, the sourcing plan, the delivery plan, and the transfer modality of a recovery operation are included. The case is from the civil war in Syria in 2017.

The problem consists of collecting food from suppliers in the food industry and delivering it to food relief agencies that serve individuals in need, of which an example is illustrated in Figure 1.

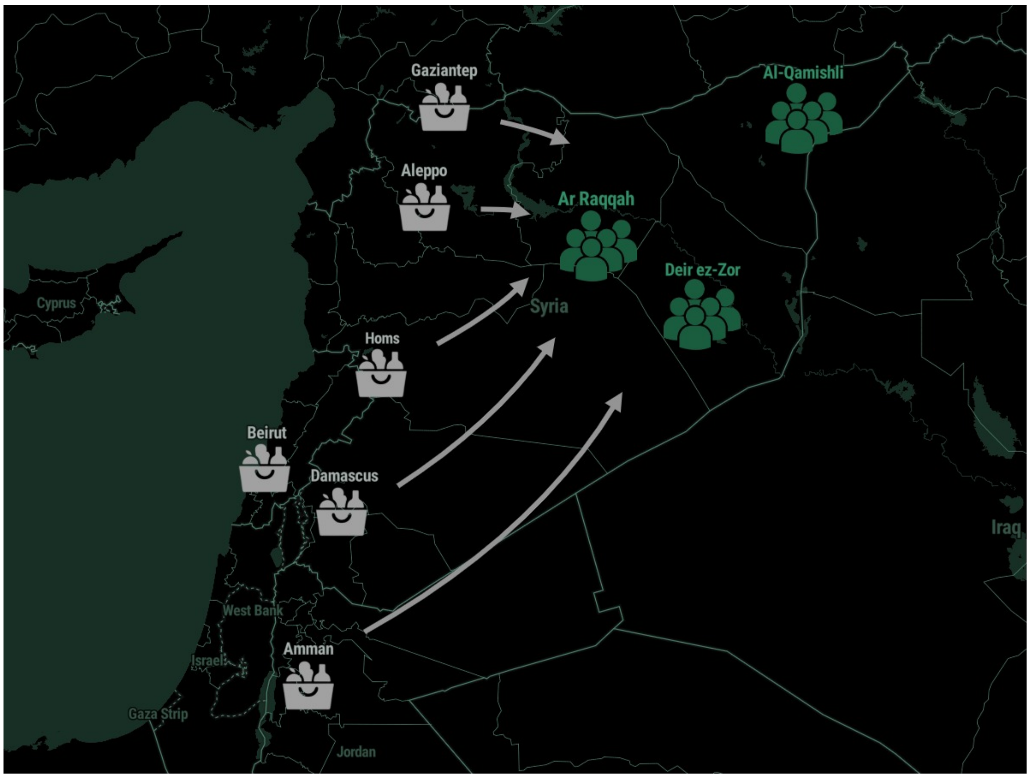


Figure 1: An example of the food collection and distribution problem in Syria, to serve the hungry population in Ar Raqqah, Al-Qamishli and Deir ez-Zor. There are regional suppliers as well as international suppliers from Lebanon, Jordan and Turkey.

The supply chain is simple: there are several international, regional and local locations where one can buy food products. The network consists of direct links between these locations to the locations of the beneficiaries.

The corresponding data are:

$\mathcal{N}_\mathcal{S}$	=	Set of suppliers (Set)
$\mathcal{N}_\mathcal{D}$	=	Set of demand locations (Set)
$\mathcal{N}_\mathcal{C}$	=	Set of commodities (Set)
p^P_{ik}	=	Procurement costs at location i for commodity k (Parameter)
p^T_{ijk}	=	Costs of transporting from location i to location j of commodity k (Parameter)
α	=	10^4, Conversion rate from metric tonnes to 100 grams (Parameter)
d_i	=	Number of beneficiaries at location i (Parameter)
$days$	=	Number of days in the time period (Parameter)
$nutval_{kl}$	=	Amount of nutritional value in commodity k of nutrient l (Parameter)
$nutreq_l$	=	Nutritional requirement per person per day for nutrient l (Parameter)

Consider the following optimization variables

F_{ijk}	$=$	Amount of commodity k shipped from location i to location j **in metric tons** (Variable)
R_k	$=$	Ration of commodity k in the daily food basket **in 100 grams** (Variable).

The linear optimization problem for this case is:

$$
\begin{aligned}
\text{(M)} \quad \min_{F_{ijk}, R_k} \quad & \sum_{i \,\in\, \mathcal{N_S}, j \in \mathcal{N_D}, k \in \mathcal{N_C}} \left(p^P_{ik} F_{ijk} + p^T_{ijk} F_{ijk} \right) && && (1) \\
\text{s.t.} \quad & \sum_{j \in \mathcal{N_S}} \alpha F_{jik} = d_i days R_k && \forall i \in \mathcal{N_D},\ \forall k && (2) \\
& \sum_k nutval_{kl} R_k \geq nutreq_l && \forall\ l && (3) \\
& F_{ijk}, R_k \geq 0 && \forall i, j, k, && (4)
\end{aligned}
$$

(a)

Give an interpretation of the objective and each of the constraints of the linear optimization problem.

(b)

We add the following constraint to model (M). What are we trying to achieve?

$$\sum_{i \,\in\, \mathcal{N_{SR}}, j, k} \left(p^P_{ik} F_{ijk} + p^T_{ijk} F_{ijk} \right) \geq 0.5 \sum_{i \,\in\, \mathcal{N_S}, j, k} \left(p^P_{ik} F_{ijk} + p^T_{ijk} F_{ijk} \right)$$

where $\mathcal{N_{SR}}$ is the set of suppliers that are regional.

Solution

(a)

In the objective, we minimize the sum of the procurement costs and the transportation costs for acquiring a commodity k from supplier i and transporting it to location j over all commodities, suppliers and demand locations.

Balance constraints (2) equalize the arriving commodity flow at the beneficiaries (from all suppliers) to the demanded rations per commodity across the time period of interest for all demand locations and commodities.

Constraints (3) ensure that the supplied nutrients meet a minimal requirement for each nutrient l by summing the nutritional contribution of each ration of commodity.

Lastly, the non-negativity constraints (4) ensure the decision variables are non-negative as it doesn't make sense for amounts and rations of commodities to be negative.

(b)

With the added constraint we require that 50% of the total cost be spent regionally and bolster the local economy.