**Vaccine Allocation Software Overview** 5/13/22

Author: Abraham Holleran

github: https://github.com/Stonepaw90/international-vaccine-allocation

**Data files**

Passengers\_2020.csv

Description: airline passengers by country

Source: web (ICAO)

Pop.csv

Description: country populations since 1820

Source: web (Our World in Data)

Vaccinations.csv

Description: vaccinations by country by day

Format: CSV

Source: web (Our World in Data)

Owid-covid-data.zip

Description: zipped file of various covid-related information by country by day.

Source: web (Our World in Data)

vax\_budget.csv

Description: Total vaccines administered by day

Source: created by Data Manipulation.Rmd

Mid\_vaccine\_params.csv

Description: Parameters for each of 166 countries

Source: created by Data Manipulation.Rmd

Four\_mid\_seir\_data.py

Description: list of 4 area aggregated parameters

Source: copied down from Data Manipulation.Rmd printed output

Vaccine Allocation parameters.xlcs

Description: model inputs not by area. Also known as Table 1.

Source: edited

Four\_mid\_params.py

Description: Model inputs not by area for four areas. Copied from Vaccine Allocation Parameters but slightly adjusted.

Source: edited

Mid\_params\_real.py

Description: Model inputs not by area for the 166 area simulation. Copied from Vaccine allocation parameters.

Source: edited

User\_input\_params.py

Description: code that allows the user to input parameters for the four area case. You can edit T (the time horizon), the Vaccine Budget, and the subset of the four areas you want to use.

User\_input\_params\_real.py

Description: code that allows the user to input parameters for the 166 country case

State history CSVs (global and four areas)

Description: state variables by day. Global csvs are used in simulation\_pipeline\_global.R and four area csv are used in simulation\_pipeline.R

Source: global\_seir.py

Plots

Description: Plots of state variables over time

Source: simulation\_pipeline\_global.R or simulation\_pipeline.R

**Programs**

Data Manipulation.Rmd

Description: Country preprocessor. combines by country data files, aggregates countries into areas. Finds global vaccination budget by day. Prints parameters for 4 areas, saves parameters for 166 countries into a .csv.

Language: R (Markdown)

Inputs: Passengers\_2020, pop, vaccinations, owid-covid-data CSV files

Outputs: vax\_budget.csv, mid\_vaccine\_params.csv, information written in four\_mid\_seir\_data.py

Mid\_process\_df.py

Description: Opens the 166 country .csv files for use in the global\_seir.py file

Inputs: vax\_budget.csv, mid\_vaccine\_params.csv

Outputs: file variables in python which are used in mid\_seir\_data\_real.py to access area parameters

Mid\_seir\_data\_real.py

Description: Loads vaccine parameters into variables, using the file that was opened in mid\_process\_df.py

Inputs: mid\_process\_df.py

Outputs: variables for global\_seir.py

Global\_seir.py

Description: simulates SEIR equations. If optimizing over vaccine allocation, repeatedly simulates and calls Gurobi linear program solver.

Inputs: mid\_process\_df.py, mid\_seir\_data\_real.py, mid\_params\_real.py, user\_input\_params\_real.py, four\_mid\_params.py, four\_mid\_seir\_data.py, user\_input\_params.py

Outputs: optimal vaccine allocation, State history as a .csv

Simulation\_pipeline\_global.R and simulation\_pipeline.R

Description: plots state variables over time

Inputs: State history DATA, day of variant, lag parameter

Outputs: plots

**Global\_seir.py**

This python file is at the center of the whole process and since the code is complicated, with few comments, this is the documentation to help you understand what happens.

Terminal:

As a user of this file, you will use the terminal to specify how the run goes.

First, enter whether you want to do the four-area aggregated run. This is done by typing the “y” key at the first prompt then pressing enter. Otherwise, type ‘n’ to use the data from all 166 countries. This takes about 20 minutes, so make sure you’re ready for the run to happen. If you’re testing something, you can go into the global\_seir.py file and make “small\_run” True. This runs everything with 12 countries. (You can set these countries in global\_seir.py)

If you want to run the code on a subset of the four countries, you can edit that in user\_input\_params.

Next, you’ll be prompted for an Alpha value. For the global runs, we use a modest .6. For the four area runs, we use a more dynamic 0.7.

Next, you’ll be prompted for the type of run. If you want a policy restriction, enter a number between 0 and 1. I type “.75” for p\_k = 0.75. If you want unrestrained, type ‘n’. If you want the big M method to encourage vaccinations, type ‘M’.

If you put a number in (for a policy constraint) you can enter ‘y’ for a strict policy that ensures exactly p\_k B(t) vaccines are administered at time t to the donor area.

Then sit back and watch the code work. The simulate function prints some output (some for debugging) and the optimize interface gives some output. The model will initialize with a few runs of simulate (to zone-in on a t\_N value), then it will switch between simulation and optimization. After convergence, a while-loop ends, and the final output is printed:

The vaccinations for each area by day

The total vaccines given by day

The total vaccinations given over the time horizon

The initial ‘naïve’ vaccine allocation death count

The objective value (deaths, except in the big M case, where it’s deaths minus about 1)

The total deaths over all areas

The objective values for each iteration (used to check convergence). Rarely, a value from a previous iteration will be significantly better than the final value.

The vaccine allocation totals for each iteration

The naïve vaccination total

Total deaths in the area (the same as the objective value, except in the big M case)

The day the variant occurs

The name of the CSV that is printed to

Policy description (that was input earlier)

Duration of the run (duration starts after the user enters the inputs)

Code:

First, the code gets user input and uses that to decide which files to import. It sets up the small run if that’s the case.

In the simulate\_covid function, a vaccine allocation is the input, and all the simulate data is the output. The function first initializes many dictionaries. It sets the values for future time steps at 0 and it will later overwrite these. It sets the time = 0 values to their correct values.

It finds the alpha value in the area that’s assumed to be the most infected area, or the area that’s calculated to be the most infected (cumulatively) between time 0 and time T\_N.

Next it calculates the infections for each area, so that the travel calculations in the next step have values for the infections of other countries.

For each time step, and then for each area, the next time step is calculated. The information for time step t is plugged into a formula that gives the information for time step t + 1.

Negative values are to be avoided, so the simulate\_covid function tells the user if there are negative values.

This function outputs the number of infected individuals, the simulated number of deaths, the alpha value (calculated from the most\_infected\_area), t\_N, the area simulated as the most infected, the susceptible populations, and the lowest value of any state variable. You don’t want the lowest value negative.

To initialize the problem, the initial number of vaccines is distributed proportionally to population.

The program runs the simulate\_covid function 3 times, so it can zone-in on a proper value for t\_N. It then runs two more times, so there’s room here to remove those 3 runs. Please test this.

The run parameters for the optimization function are set. Some parameters depend on the run type. The exploration tolerance is higher than in the Bertsimas paper, all things considered. If these values are lower, you get infeasible linear programs.

A while loop is run to simulate then optimize. Model variables and auxiliary variables are initialized, then the objective vales and constraints are added depending on the run type.

The program runs, then various values are stopped and printed. The string “var.varname” is checked to see if it’s a vaccine variable. It would be easier to call the object “vax\_variables” and vax\_variables[ar, i].x to get the value, but I didn’t realize you could do this when I made this part.

The program stores values and checks for loops. To check for loops, it checks for values that are very close to each other. If these are found, it terminates early. This loop clause is quite important for some runs.

More information is gathered for printing via the string names. The program saves the state variables and the vaccine variables to a csv.

Finally, the output information in the “terminal” section is printed out.

Diagram of SEIR OPT overview

Diagram

Description automatically generated