# GTAP in R workshop

Thiago Simonato    Uris Lantz Baldos

2024-03-10

## Workshop Roadmap

# 1 Workshop Overview

This workshop is designed to introduce a couple of tools to work with GTAP models in R. Our session, spanning 120 minutes, will cover a comprehensive range of topics, from the basic installation of the packages to utilizing its features. This workshop is structured to provide a comprehensive overview of the strategies used to interpret and solve the GTAP version 7[1] model, with a primary focus on practical, hands-on experience. Among the packages used in the workshop is the HARr package, which serves as a bridge between the databases and the R language by reading and writing *.har* files adopted in the GEMPACK suite. The TabloToR package that runs GTAP simulations as it interprets the model in GEMPACK-style TABLO *.tab* files and solves it within the R environment[2]. Other data visualization packages are also used as an example of an application for building automated analysis in R.

Table 1: Workshop Agenda

| Topic | Duration |
|---|---|
| Workshop Overview | 5 minutes |
| Introduction | 5 minutes |
| A Brief Overview of Data Structures in R | 5 minutes |
| Setting up | 5 minutes |
| Hands-On Session | 30 minutes |
| Q&A and Discussion | 15 minutes |
| Extra contents | - |

# 2 Introduction

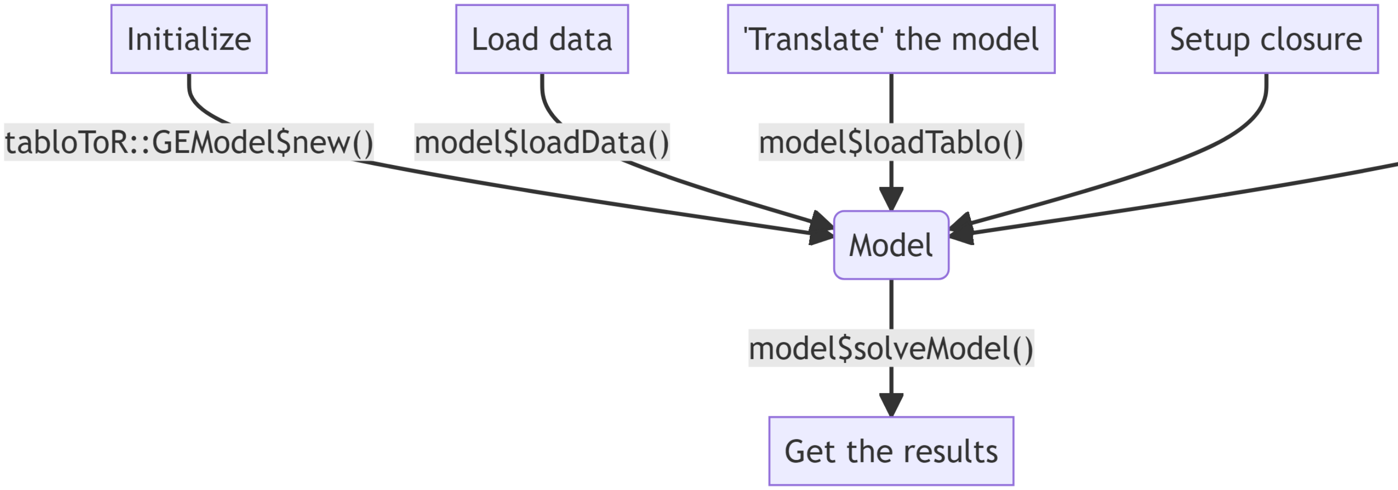
Note

- Welcome and workshop objectives
- Brief `base R` data structure description
- Overview of `TabloToR`
- Is this completely free?
- Applications
- Limitations

[1] Corong, E. L., Hertel, T. W., McDougall, R., Tsigas, M. E., & Van Der Mensbrugghe, D. (2017). The standard GTAP model, version 7. Journal of Global Economic Analysis, 2(1), 1-119.

[2] M. Ivanic, The GTAP model in R, 2020, 23rd Annual Conference on Global Economic Analysis.

The diagram below illustrates the R package workflow for modeling. It starts with initializing the model, followed by configuring data loading, model translation from TABLO to R, closure setup, and shock application. The process concludes with running the model to obtain results.




## 3 Setting Up

To run the workshop exercises, you will need to have *R* installed on your computer. You can download R from the Comprehensive R Archive Network (CRAN). We also highly recommend installing *RStudio*, a powerful and user-friendly integrated development environment for R. RStudio can be downloaded from the official RStudio website.

Additionally, the `TabloToR` package is required for the workshop. If it is not already installed on your system, you can install it using the following command in R:

```r
if (!require("devtools", quietly = TRUE)) {
  install.packages("devtools")
}
if (!requireNamespace("tabloToR", quietly = TRUE)) {
  devtools::install_github('tsimonato/MTED-TabloToR', force = TRUE)
}
```

Set working directory to the workshop folder:

## 3.1 In Rstudio

The command below sets the Rstudio working directory to the folder where the .R file you are editing is located (only works if you are using Rstudio):

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
getwd()
```

## 3.2 Manual

When setting the working directory manually in R, remember that R does not accept the single backslash `\` in file paths. Instead, you should use either double backslashes `\\` or forward slashes `/`.

```
setwd("PATH TO WORKSHOP FOLDER HERE")
getwd()
```

> Caution
>
> - We will use the `package::function()` synthax in the code examples. Therefore, `library("package")` commands will not be necessary.
> - Check if your current working directory is correct.

## 3.3 A Brief Overview of Data Structures in R

R provides several data structures for handling data. Among the most commonly used are Data.frames, Arrays, and Lists. Each structure has its unique features and use cases.

## 3.4 Data.frames

A Data.frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. It's similar to a spreadsheet or SQL table and is particularly well-suited for data manipulation and analysis tasks.

Example of a Data.frame:

```
df <- data.frame(
  Commodities = c("Wheat", "Corn", "Wheat", "Corn"),
  Activities = c("Farming", "Farming", "Harvesting", "Harvesting"),
  Margins = c("Low", "High", "Low", "High"),
  Values = c(1, 2, 3, 4)
```

```
)
df
```

.

Subsetting in a Data.frame can be done to select specific rows, columns, or both:

#### 3.4.1 Filter by row

Here, we select rows where the `Values` column is greater than 2

```
df[df$Values > 2, ]
```

#### 3.4.2 Filter by column

Here, we select the 'Commodities' column

```
df$Commodities
```

#### 3.4.3 Filter by rows and columns

Here, we select rows where 'Values' is greater than 2 and only the 'Commodities' and 'Activities' columns

```
df[df$Values > 2, c("Commodities", "Activities")]
```

### 3.5 Arrays

An Array in R is a multi-dimensional data structure that can store data in more than two dimensions. While arrays can have any number of dimensions, a two-dimensional array is known as a matrix. Arrays are useful for performing mathematical operations on multi-dimensional data.

Example of an Array:

```
# Assuming each combination of categories has a unique value
array_data <- array(df$Values, dim = c(2, 2, 2))
dimnames(array_data) <- list(
  unique(df$Commodities),
  unique(df$Activities),
  unique(df$Margins)
)
```

```
array_data

# Subsetting data from an Array
array_subset <- array_data["Wheat", "Farming", ]
array_subset
```

## 3.6 Lists

A List is a collection of elements that can be of different types and structures. You can think of a list as a container that holds a variety of objects, which can be accessed and manipulated individually.

Example of a List:

```
list_data <- list(
  Data_Frame = df,
  Array = array_data,
  Details = list(
    Commodities = unique(df$Commodities),
    Activities = unique(df$Activities),
    Margins = unique(df$Margins)
  )
)
list_data
```

You can subset elements of a list using the `$`. For nested lists, you can chain these operators to access deeper levels.

### 3.6.1 Select a Data.frame within the list

```
list_data$Data_Frame
```

### 3.6.2 Select a specific column in the Data.frame

```
list_data$Data_Frame$Commodities
```

### 3.6.3 Select a specific value in the Array

```
list_data$Array["Wheat", "Farming", "Low"]
```

# 4 Hands-On Session

In this section, our attention will be centered on the steps evolved to run **GTAPv7** simulations in R. We will delve into some key tasks: learning how to effectively import and examine .har files in R, interpreting models presented in .tab format within the R environment, executing simulations, and exploring the resulting data.

## 4.1 Step 1: Set up the GTAP database

GEMPACK utilizes custom .har binary data files. The R package HARr enables R to directly read these .har files, specifically headers with character, integer, and dense real matrices. It converts the data in these headers, including set information, into a list of arrays, in which each array represents an Header of the .har file.

In this step, you'll write a script to read the GTAP model's database files and explore specific data within them. After loading the files, extract and examine the header `vdfb` (Domestic purchases, by firms, at basic prices) from the appropriate file. Explore the structure and contents of this header.

> **Tasks**
>
> 1. Load the .har files using the HARr package.
> 2. From the loaded files, extract the header `vmfp`(Imported purchases, by firms, at producers prices), then subset acts to `food`.
> 3. Convert `vmfp` to a data.frame class.
> 4. Analyze the extracted `vdfb` data. You can make a chart if you want it.

### 4.1.1 Task 1: Loading the GTAP Database

In this task, you will learn how to load the GTAP model's database files into R using the `HARr` package. This step is crucial as it sets the foundation for all subsequent analyses. You'll focus on how to read .har files and understand their structure within the R environment.

> **Caution**
>
> The name of each list must be the same as that specified in the database model, preferably in lowercase. See below that the names in the .tab file and the previously generated lists are the same.
> *GTAPv7.tab*, line 99:
>
> ```
> ...
> File
> ```

```
    GTAPSETS # file with set specification #;
File
    GTAPDATA # file containing all Base Data #;
File
    GTAPPARM # file containing behavioral parameters #;
...
```

> **Tip**
>
> Use the HARr package to load the .har files named *SETS.HAR*, *Default.prm*, and *Basedata.har* into R. Store the read files in a list named data, with keys `gtapsets`, `gtapdata`, and `gtapparm` respectively.

```r
# You need to get the data files .har
database = list(
  gtapsets = HARr::read_har('model\\SETS.HAR'),
  gtapdata = HARr::read_har('model\\Basedata.har'),
  gtapparm = HARr::read_har('model\\Default.prm')
)
```

### 4.1.2 Task 2: Extracting and Filtering Data

This task involves extracting specific data from the loaded GTAP database, with a focus on the `vmfp` header. You will learn how to filter this data for particular regions and activities. This exercise will assist you in becoming familiar with the format and structure of the data utilized in these packages.

> **Tip**
>
> - Use R's traditional sublist selector`$` to subset lists and [,,] to subset the array in its 3 dimensions.
> - Note that all characters in the database are in lower case, including the name of the headers.

Extracting the `vmfp` header:

```r
vmfp <- database$gtapdata$vmfp
vmfp
```

Filtering for *food* in *acts*:

> **Tip**
>
> To filter an array, it helps to know the number and order of its dimensions, in the case of vmfp: `[comm,acts,reg]`.

```
length(vmfp)                                                    ①
dim(vmfp)                                                       ②
dimnames(vmfp)                                                  ③
# Subset a list using $ and a array []
vmfp_food <- database$gtapdata$vmfp[,"food",]
vmfp_food
```

① Returns the total number of elements (values) in the array.
② Returns the dimensions of the array, showing how many elements are in each dimension.
③ Return the names of each dimensions and its elements. It's useful for understanding what each dimension represents.

### 4.1.3 Task 3: Converting Data for Analysis

Here, you will convert the extracted array data into a `data.frame` format using base R. This conversion is essential for making the data more accessible and easier to analyze using various R functions and packages like `data.table`, `dplyr` and `tidyr`.

```
vmfp_df <- as.data.frame.table(vmfp)
vmfp_df
```

> **Caution**
>
> Arrays converted to data.frame through `as.data.frame.table()` function have *Freq* as the name of the column of numerical values. Keep this in mind when using this type of conversion.

### 4.1.4 Task 4: Exploring the data

There are many options for exploring these databases in R. This can be useful for building a descriptive analysis of the database, identifying patterns, etc.

> **Tip**
>
> Perhaps a bar chart using `barplot()` function would be a good option in this case.

2 dimensions static bar chart example:

```
barplot(
  vmfp_food,
  legend.text = rownames(vmfp_food),
  main= "Imported purchases, by firms, at producers prices (million US$)"
)
```

## 4.2 Step 2: Read the model

In this step, we will convert the **GTAPv7** model from the TABLO language into a format that R can understand. This is achieved through a process that interprets and translates different components of the model, statements like *Read*, *Set*, *Coefficient*, *Variable*, *Formula*, *Equation*, *Parameter*, *Mapping*, etc. Each category of commands is handled in a specific way to ensure they are correctly represented in R.

The correct labeling of each statement into its appropriate group is essential for the subsequent definition and construction of the matrix that will be inverted during the simulation. This precise organization is key to maintaining the integrity of the model's structure and calculations as it transitions into the R environment.

> **Tasks**
>
> 1. Set the path to the model's TABLO file.
> 2. Initialize a GEMModel object.
> 3. Assign to this object the path to the model.
> 4. Assign the GTAPv7 database to the model.

```
tabloPath <- "model\\GTAPv7.tab"                    ①

model <- tabloToR::GEModel$new()                    ②

model$loadTablo(tabloPath)                          ③

model$loadData(database)                            ④
```

① The location path for the **GTAPv7** model's TABLO file.
② Create a new instance of the *GEModel* class.
③ This step involves translating the model's equations, formulas, mappings, and other commands into a syntax understandable by R. The translated model is then stored within the `model` object, which was initialized in the previous step.
④ Load the `database` read in the previews step.

### 4.3 Step 3: Setup the closure and shocks

In this step, we focus on setting up the **closure** conditions and applying specific **shocks** to the model. Closure, what can be seen as the decision about which variables are exogenous and which are endogenous, is a critical aspect that determines its operational framework, influencing how the model reaches equilibrium and responds to external changes. It essentially sets the 'rules of the game' for the model, defining what is fixed and what is flexible within its economic environment. Once the closure are defined, we define the vectors of shocks, these are changes to the model's inputs that simulate different scenarios, such as economic or policy changes.

> Tasks
>
> 1. Defining exogenous variables.
> 2. Setting all exogenous variables to zero.
> 3. Apply an chosk of 5% increase in productivity for the `food` activity in the `ssa` and `eu` regions.

#### 4.3.1 Task 1: Identifying Exogenous Variables

In this task, you will create a character vector composed of exogenous variables. These are variables that are determined outside the model and are inputs to the model's system.

> Tip
>
> When defining exogenous variables, it's important to ensure that they align with the variables used in your specific model version (GTAPv7 in this case). This list forms the basis for setting up the model's initial conditions and applying shocks later.

```
exogenous_variables <- c(
  'afall', 'afcom', 'afeall', 'afecom', 'afereg', 'afesec', 'afreg', 'afsec',
  'aintall', 'aintreg', 'aintsec', 'ams', 'aoall', 'aoreg', 'aosec', 'atall',
  'atd', 'atf', 'atm', 'ats', 'au', 'avaall', 'avareg', 'avasec', 'cgdslack',
  'dpgov', 'dppriv', 'dpsave', 'endwslack', 'incomeslack', 'pfactwld', 'pop',
  'profitslack', 'psaveslack', 'qe', 'qesf', 'tfd', 'tfe', 'tfm', 'tgd',
  'tgm', 'tid', 'tim', 'tinc', 'tm', 'tms', 'to', 'tpdall', 'tpmall', 'tpreg',
  'tradslack', 'tx', 'txs'
)
```

### 4.3.2 Task 2: Initializing Model Variables

Here, you'll assign the value `0` to each of these variables in `model$variableValues`. This step is crucial for preparing the model's baseline before applying any shocks.

> **Tip**
>
> Using a for() loop to iterate over the list of exogenous variables is an efficient way to assign values to them.

```
for (var in exogenous_variables) {
  model$variableValues[[var]][] <- 0
}
```

### 4.3.3 Task 3: Applying Shocks to the Model

In this task, you will apply a specific shock to the model. For example, a 5% increase in the total productivity change variable (`aoall`) for the `food` activity in the `ssa` and `eu` regions.

> **Tip**
>
> You can adapt using an similar approach to this previous example: `list_data$Array["Wheat", "Farming", "Low"]`

```
model$variableValues$aoall["food", c("ssa", "eu")] = 10
model$variableValues$aoall
```

```
       reg
acts     ssa eu row
  food    10 10   0
  mnfcs    0  0   0
  svces    0  0   0
```

## 4.4 Run the model

At this stage of the process, we are ready to execute the model, bringing together all the elements we've prepared: the input database, closure conditions, and the shocks we've defined. This step involves solving the model's matrix, which is a representation of the equations from the model. The model's execution not only solves the matrix but also carries out the calculations defined in the model's formulas. Additionally, for GTAPv7, post-simulation processes are conducted, which involve further calculations based on the model's output.

Running with 1 iteration and 1 step (*Johansen* method).

```
model$solveModel(
  iter = 1,
  steps = 1
)
```

Running with 3 iteration and multiple steps.

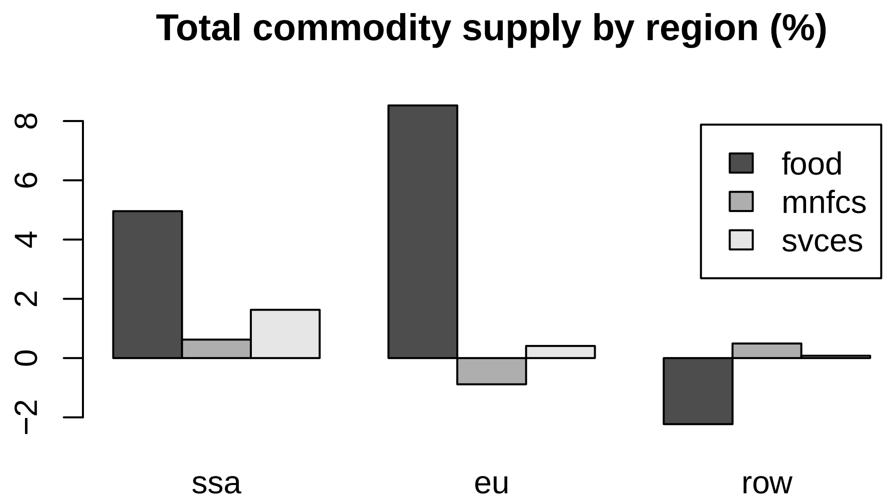
```
model$solveModel(
  iter = 3,
  steps = c(1, 3)
)
```

Results for percentage change in total commodity supply by region (`qc`):

```
outputData$qc
```

```
       reg
comm          ssa         eu         row
  food   4.9577805  8.5266022 -2.23012678
  mnfcs 0.6236865 -0.8850531  0.49274290
  svces 1.6287317  0.4097268  0.07828056
```

```
barplot(
  outputData$qc,
  beside = TRUE,
  legend.text = rownames(outputData$qc),
  main= "Total commodity supply by region (%)"
)
```




## 5 Extra content: Post Sim analisys

## 6 Q&A and Discussion

Note

- Open floor for questions
- Group discussion on applications