

The GTAP model in Julia

Maros Ivanic

I introduce a formulation of the GTAP version 7 model in an open-source language, Julia, that closely follows the specification of the model in GEMPACK, including equation and variable names. Unlike the linearized GEMPACK version, my formulation is in levels. I show that my model produces nearly identical results with differences at or beyond the fifth significant digit. Because the model is expressed in levels, it is able to address a wider range of policy questions, especially those that are not suitable for perturbation assessment, such as simulations involving parameter changes or new value flows. My formulation of the GTAP model is quite different from the earlier formulation in levels in GAMS (Mensbrugghe 2018), commercial software, which does not follow the nomenclature of the GTAP model in GEMPACK and also requires a software license, making it more difficult for the GEMPACK users to transition to.

Keywords: GTAP, Julia, JuMP, CGE

Acknowledgments: I would like to thank Jayson Beckman, Eric Davis and Noe Nava for their excellent comments on an early draft of this work.

Disclaimer: The findings and conclusions in this publication are those of the author and should not be construed to represent any official USDA or U.S. Government determination or policy. This research was supported in part by the U.S. Department of Agriculture, Economic Research Service.

Introduction

It is quite surprising that despite its wide use in quantifiable global policy analysis documented by about forty-two thousand mentions on Google Scholar, the GTAP model has been remarkably difficult to run for uninitiated members of the research community. If someone with skills in general computing, e.g., R, Python or Julia, who has just learned of this great model would decide to try it, they would likely soon find themselves on a fairly arduous journey of finding the correct model among many resources available on the GTAP website, purchasing and learning a math solving system, such as GEMPACK or GAMS, and then obtaining the necessary data to allow the model to run, either by purchase or by downloading any of the older databases available for free.

Even though a part of the difficulty in getting involved with the GTAP model can be attributed to the fact that the most up-to-date data supporting the model are only available for purchase, the incompatibility of its most common implementations, i.e., in GEMPACK or GAMS, with the modern computing paradigm poses an even greater barrier for modern researchers interested in exploring or using the model.

To understand why it is difficult to run the GTAP model, which is nominally open source, on one's computer, it is important to understand that open-source software solutions are typically executed within open-source computing frameworks such as Python, R, and Julia. The requirement to install commercial, stand-alone software that operates on a small number of operating systems to run GTAP presents an important barrier to anyone interested in exploring it.

The benefit of true open-source software solutions is that they typically run within existing open-source frameworks and therefore they need not be installed as a separate application on the operating system, which is an important benefit for those researchers without administrative access to their computers or those who do not feel comfortable installing applications from sources that they may not know well.

An added benefit of using open-source frameworks is that the distribution of software is achieved using standard package distribution, which reduces the need to find and download the latest version of the software manually. Typically, there is a repository of software solutions for each framework that is vetted, trusted and easily accessible. The fact that open-source frameworks allow for easy package access also encourages the software authors to integrate it within the same framework by utilizing existing packages in its development.

Finally, because the software only needs to run within the computing framework, it can be used on any operating system for which this framework is available. In today's world, researchers use a variety of operating systems, and it is desirable that the developers of useful packages, e.g., a new CGE model, or estimation technique, do not need to worry much about making their work compatible with all possible computer configurations.^{1 2}

¹Because of the interest of the open-source community to integrate each other's solutions, there is a strong push for good practices and conventions to be applied, resulting in typically much more easily to read, better organized, and more aesthetically pleasing code, another benefit of open-source solutions.

²As of January 2025, there have been about 45 million downloads of Julia and over 10,000 registered packages. Combining Julia users and developers with a 31,000+ strong GTAP network of researchers could greatly benefit both groups.

It is very obvious that the GTAP model, in terms of its software implementation in either GEMPACK or GAMS is lacking at nearly every aspect of modern software development and distribution described above: the code required for the execution of the model needs to be looked up and downloaded manually from the GTAP Center’s website (or other websites that need to be identified) and then compiled using a commercial software solution that is a standalone application, likely requiring administrative access to the operating system.³ In addition, it requires one to obtain a license, a substantial cost with the exception of the smallest models. The user also needs to become familiar at minimum with one stand-alone software that reads and aggregates the data, another one that executes the model, and possibly others to view and analyze the results. The software required to execute the model is additionally only available for a limited number of operating systems, or Windows alone, further limiting its availability to the potential users. Many of the same obstacles that apply to running the model also apply to viewing the model’s data and outputs.

In this work, I present a very different workflow for the standard GTAP model (Corong et al. 2017) that places the development and execution of the GTAP model into the standard framework. Specifically, I present a version of the model in Julia, an open-source computational framework, in the form of a downloadable package, with accompanying tools for data aggregation. I demonstrate that this version of the model produces the same results as the model executed in GEMPACK (Horridge et al. 2019), and it produces higher accuracy results at a comparable execution time, or even shorter time when taking accuracy into consideration. In the rest of the paper, I describe the translated model and explain how it can be accessed and executed. I conclude by demonstrating the equivalency of its results for three sample scenarios by comparing the results obtained in Julia with those of the same model executed in GEMPACK.

Executing GTAP model v 7 in Julia

In this section, I walk the reader through the steps necessary to complete a GTAP simulation in Julia, including data aggregation, running the model, calculating welfare change and real GDP growth, and viewing the results. Unlike the GEMPACK version of the model, the entire process is completed within Julia—there is no other software that needs to be downloaded and installed. To follow the examples presented, the user only needs to download and install Julia and several packages, including JuMP (Lubin et al. 2023) and Ipopt (Wächter and Biegler 2006), on their computer.

Data aggregation

Data aggregation represents the first step in a typical GTAP model analysis. The reason why data are aggregated is to allow for the model to run in a timely fashion within the limits of one’s computer. To

³The compilation of the model is more cumbersome in GEMPACK, where the user needs to manually compile the human-readable model into either an EXE file or a GemSim format. GAMS compiles the code once the user runs it. However, the greatest disadvantage of both systems over modern systems is the lack of interactive execution where the user may run code by line and evaluate variables and other objects as needed.

understand the size problem, consider the fact that the latest available GTAP database at the time of this writing, version 11, contains 160 regions and 65 sectors. Just to describe the bilateral trade of this many elements requires 1.7 million equations and variables. Adding just three distinct margins to each trade flow would add another 5 million variables and equations, quickly exhausting the capacity of most computers.

Data aggregation in GTAP has traditionally been performed by stand-alone applications such as GTA-PAgg or FlexAgg, which require the user to specify the groupings of commodities, factors and regions using a graphical user interface (GUI) in GTA-PAgg or a script file in FlexAgg. In the case of a script, the input file is in a custom format that essentially needs to be produced by hand. The script is then processed by a Fortran code, written in TABLO language, that produces the aggregated database. It is important to note that both the disaggregated database input and output files are in a proprietary, Fortran-based HAR format that until recently could not be opened by any generally available data processing software.⁴

The inability of most users to explore the distributed data without a lot of investment, plus the fact that the aggregation code is not visibly published in a common public repository, e.g., GitHub, has likely contributed to the slow pace of improvement to the aggregation script and the persistence of errors in it. For example, even though version 7 of the GTAP model introduced region-specific trade substitution elasticities, for a number of years, they were not properly generated by the aggregation script; instead, the aggregation script would produce region-generic elasticities suitable for version 6.2 of the model and then use these values to create a version 7 database. While the aggregation script has been amended to produce data for version 7 of the model, it still, in some instances, produces erroneous region-generic elasticities.

In my Julia package, I provide for data aggregation entirely within Julia. Because Julia is a language that allows vectorization and user functions, the entire aggregation script involves about 100 lines of code that is easy to read. Moreover, this code is published on GitHub to enable the users to quickly point out errors and suggest fixes. To allow users to generate results comparable to what would be obtained using GEMPACK, I also include a function that allows users to apply the GEMPACK approach that incorrectly also turns the region-specific elasticities into region-generic ones. I do not, however, endorse its use in work destined for publication.

Model

The GTAP model is typically executed in GEMPACK, GAMS and, in recent years for smaller models, in R (Ivanic 2023). Both GEMPACK and GAMS are commercial computational frameworks, while R is an open-source one. Because the open-source version in R is far slower than the its commercial counterparts, it is probably best suited for educational work or analyses that do not require much regional and sectoral detail.

⁴The availability of the R package HARr which can read and write HAR data files and read SL4 solution files, removed most of these barriers for the version of the model executed in the GEMPACK software.

The GTAP model has been historically run in GEMPACK, a software specifically formulated to solve large (at the time), linear CGE models. The software, written in Fortran, provides several key features supporting CGE analysis, including a succinct formulation of the models in the TABLO language, a high level language specifically designed for describing CGE models, which it then translates to Fortran and executes. Unlike Fortran, which is a low-level computer language, TABLO is quite readable and able to define the basic building blocks of a CGE model, such as sets, variables, coefficients and equations. This allowed modelers to focus on economic theory rather than the modalities of the operating and file systems.

GEMPACK solutions are derived from linear approximations and this puts some mathematical constraints on what can be modeled. First, no zero value can ever change in the GEMPACK model, as any percentage change applied to zero results in zero. This naturally prevents the GTAP model formulated in GEMPACK from addressing policy questions that involve an introduction of a new market, commodity or a trade flow.^{5, 6} Second, due to its linear nature, all equations in GEMPACK must be in the form of parameter/coefficient $\times$ percent change in variable, where percent changes in variables are multiplied by the parameter or coefficient values. Because a zero vector (i.e., no change to any endogenous variables) is the solution to the model, changes in parameters do not cause any change in the model results as the zero vector remains the solution for any parameter/coefficient values. This prevents a whole class of policy simulations involving changes in fixed shares, preferences, elasticities, etc. For example, in the standard formulation of the GTAP model, a region with a negative savings rate must stay that way forever.

The users of GEMPACK know that and do not generally expect that the linearized equations hold for the solved values. GAMS, on the other hand is a much more general computational system which is intended to solve the system of equations exactly. The difference between the approach of GEMPACK and GAMS can be illustrated with a simple linearized equation $a = b + c$ to represent the relationship $A = B \times C$, where a , b , and c are percent change variables to the levels of the values A , B and C . GEMPACK interprets this equation to mean that for infinitesimally small changes, the percent change in a is the sum of the percentage changes in b and c . However, for large changes, I would not expect the equation to hold. In fact, for $b = 10\%$, $c = 10\%$, the solution is $a = 21\%$.

While GEMPACK users are perfectly fine with a solution that may equate $21\% = 10\% + 10\%$ because they know that relationships expressed in linear equations only hold only for small changes, GAMS imposes the requirement that equation solutions be mathematically correct. It is therefore necessary to write the underlying levels equation in GAMS, for example as $A \times (1 + a) = B \times (1 + b) \times C \times (1 + c)$ which would yield the solution above, even if the representation is a bit more complex.

Both GEMPACK and GAMS have enabled numerous users to employ the GTAP model, using either percentage changes or levels. However, both tools have two important drawbacks which are connected

⁵Conversely, it is also not typically possible to bring any variable to zero either—no matter how much relative price grows, a CES formulation requires that some, infinitesimal trade takes place.

⁶Modelers have been partly able to avoid the problem of zero flows, by replacing zeros with small numbers (e.g., one dollar for the total flow) which then allows increasing the flow further; however, that typically requires excessively large (often in excess of 1,000 percent) increases in model variables (e.g., Beckman and Arita 2016), which are hard to defend or explain.

to each other: their commercial nature means that the users have to pay for a license and are unable to contribute directly to the software's improvement and development. This greatly slows down the rate of updates as they need to be produced by fewer people. This then leads to the second drawback of technological outdatedness and the resulting lack of integration.

The commercial status of GEMPACK and GAMS which require a payment for a license to solve the GTAP model of any useful size also reduces the quality of research done using this software: review and replication of results is difficult for those without the license. Even though trial licenses are sometimes available, they are typically very limiting.

The technological lag is another important drawback of both GEMPACK and GAMS. Both pieces of software maintain a fairly antiquated approach to computing, largely following the steps a researcher might have taken fifty years ago: formulate the model, put together the data, submit the task to be executed to a mainframe computer and collect the results once available. Because mainframe computers could not work in an interactive mode, this workflow made perfect sense in 1980's. The key limitation of this approach is, of course, that any results which were not included in the requested output are lost as soon as the mainframe finished the job. However, with the advance of personal computers available at researchers' desks, it has long stopped making sense to follow this approach. Modern software has instead evolved to be interactive, allowing researchers to probe their mathematical and statistical models in real time, and thus removing the need to re-run entire models to implement any change.

The lag in adopting modern technological trends by GEMPACK and GAMS has also largely contributed to their diminished ability to integrate with other software and commonly used tools. For example, both pieces of software are built around custom file formats (e.g., HAR and GDX) and offer very limited options to convert data in and out of those formats, frequently requiring a license to use those tools. This means that it is typically not an easy task to incorporate GAMS or GEMPACK solutions into standard workflows, e.g., to produce automatic outputs and visualizations using common scripting languages, such as R, Python, and Julia.

In my implementation of the GTAP model in Julia, I follow the approach adopted by Mensbrugghe (2018) in creating a levels version of the model in GAMS. This is mainly because linearized solutions are non-standard and more restrictive than levels formulations which reflect the actual equations (rather than their linear approximations) and allows for parameter changes in the model in addition to the changes in exogenous variables.

To make the model in Julia as easy to comprehend by existing users as possible, I follow as closely as possible the existing variable and equation names. Of course, the levels formulation requires the addition of additional parameters and equations to assure proper calibration of parameters.

Similarity between linearized and levels equations

Most of the equations translated to Julia can be easily traced to those in GEMPACK. For example, equation `e_qxs` in Julia is a levels version of the equation in GEMPACK:
In GEMPACK:

```

Equation E_qxs
# regional demand for disaggregated imported commodities by source #
(all,c,COMM)(all,s,REG)(all,d,REG)
  qxs(c,s,d)
    = -ams(c,s,d) + qms(c,d)
    - ESUBM(c,d) * [pmds(c,s,d) - ams(c,s,d) - pms(c,d)];

```

In Julia:

```

e_qxs[c=comm, r=reg],
  log.(Array(qxs[c, :, r])[δ_qxs[c, :, r]]) .==
  log.( ces(qms[c, r], Vector(pmds[c, :, r])[δ_qxs[c, :, r]],
    Vector(α_qxs[c, :, r])[δ_qxs[c, :, r]], esubm[c, r], γ_qxs[c, r])

```

Of course, levels equation require additional information for this function, including the distributional parameters α_{qxs} and the scaling parameter γ_{qxs} , both of which are calculated during the calibration process, and a binary switch for any missing flows, δ_{qxs} , which is calculated based on the initial data.

Results

The processing of the results of a GTAP model in both GAMS and GEMPACK requires the use of additional commercial software. This is mainly because both are non-interactive systems—the user submits all inputs required for the model run and collects all specified outputs after the model completes its execution and saves the results. Because many variables in the GTAP model are multidimensional matrices, special tools are needed to view the data on two dimensional screens. These tools often come as separate applications, e.g., ViewHAR, ViewSol, requiring additional installation and instruction for use. For example, it is not possible to inspect an equation in GEMPACK or GAMS directly with the values currently solved in the model; instead specialized software/work is required to achieve this (e.g., AnalyseGE).

In my implementation, I intentionally do not produce any specialized software for viewing the results. Instead, I rely on the standard methods of viewing complex objects in Julia using common functions and methods. Because Julia is an interactive framework, it is possible to view, output, or visualize variables and equations in real time before and after the model completes.

The interactive nature of Julia presents an important improvement over the workflows of GEMPACK or GAMS where only the variables that are output during the run are available and everything else is lost once the model has completed. In Julia, after the model has solved, all equations and variables are available in memory and can be inspected with ease.

Sample workflow

In this section, I present sample workflows of completing an analysis using my formulation GTAP model in Julia. For comparison, I present in Figure 1 a comparison of the workflows of the same task in both GEMPACK and Julia. As the figure shows, while the workflows are similar, there are some crucial differences. First, Julia workflow does not involve obtaining any licenses. Second, the Julia workflow adds an additional step of calibrating the data, which is needed because the model is expressed in levels. While this extra step takes some time, it only needs to be done once. Finally, unlike the workflow in GEMPACK, the one in Julia involves no other software—everything from data aggregation to viewing the results is done in the same software and the same environment.

I next walk the reader through the steps of working with the GTAP model in Julia and provide examples.

Installation

To install GTAP v7 in Julia, install package `GlobalTradeAnalysisProjectModelV7` from the Julia registry.



Installation of the package

```
using Pkg
Pkg.add("GlobalTradeAnalysisProjectModelV7")
```

Preparing the simulation data

The first step for any analysis in `GlobalTradeAnalysisProjectModelV7.jl` package is to obtain GTAP data in the required form, i.e., a dictionary of arrays for the data, parameters and sets. A separate package `HeaderArrayFiles.jl` allows the data to be read in HAR format; however, it is not required to use this particular package. It is also not required, even though it is quite convenient at this point, to use the data provided by the GTAP Center in the HAR format.

For those users who are not familiar with GTAP data, or find the process difficult to follow, the function `get_sample_data()` provides a sample aggregation of the freely available version 9 of the GTAP database aggregated into seven regions, six sectors and five factors. This should help users avoid undergoing the multi-step process to obtain the data from the GTAP Center's website.



Obtaining sample data

The user can run the function `get_sample_data()` without any arguments. The function returns a named tuple including `hData`, a dictionary of all required data (e.g., to produce initial values for

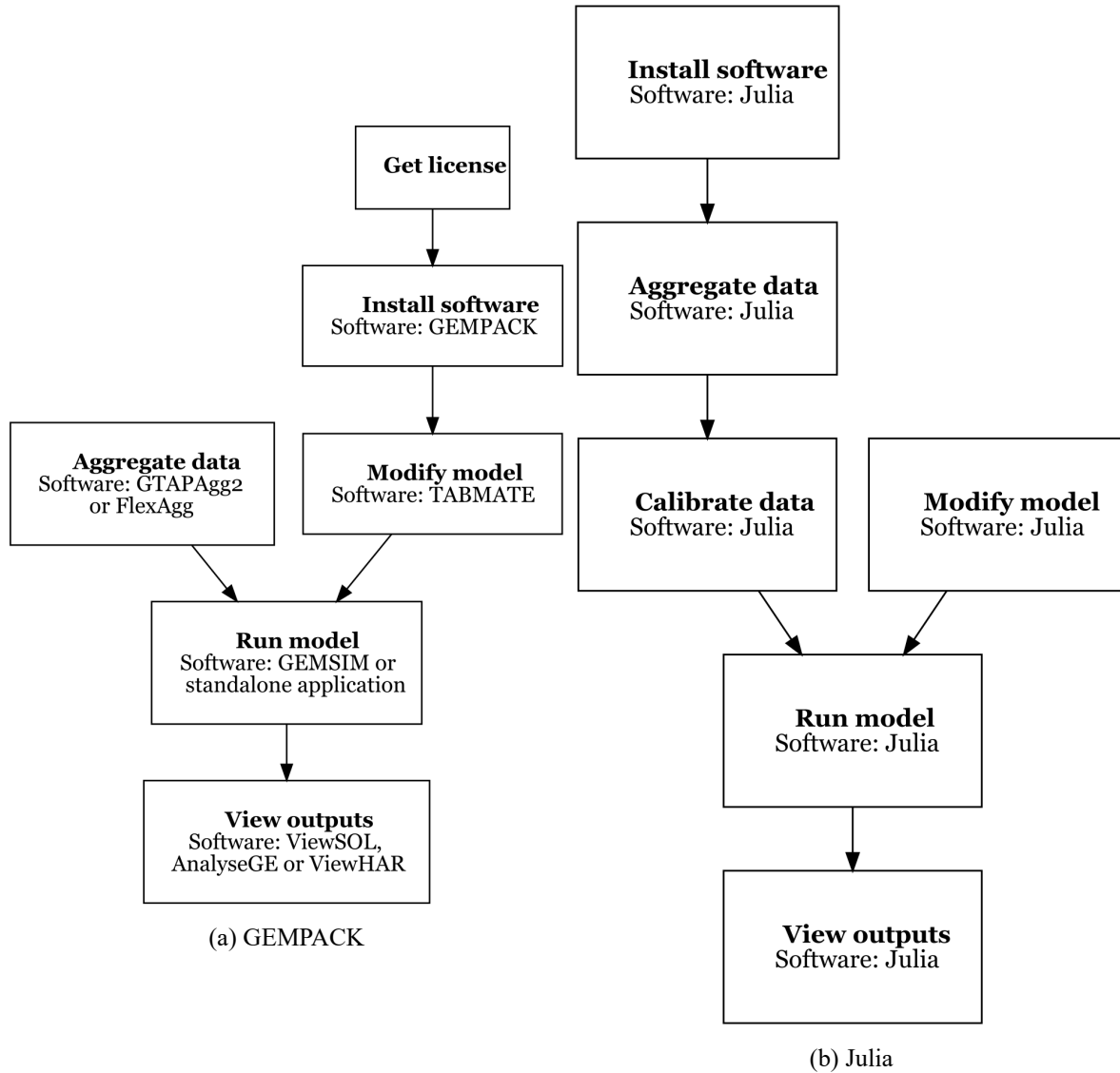


Figure 1: Comparison of workflows in GEMPACK versus Julia

all model variables and calibrated parameters), `hParameters`, a dictionary of all parameters, and `hSets`, a dictionary of the sets used in the model.

```
(; hData, hParameters, hSets) = get_sample_data()
```

This named tuple is structurally the same as the result of the aggregation described below. If the user intends to use the sample data, he or she can skip the aggregation section entirely.

Data aggregation

Data aggregation using the tools provided by the Centre of Policy Studies, e.g., GTAPAgg or FlexAgg is either not replicable (GTAPAgg) or prone to error (FlexAgg): GTAPAgg is a GUI that requires great attention from the user for precise clicking in order to aggregate the data as needed. On the other hand, FlexAgg operates off of a custom script file, as opposed to a standard data format, where a lot of data is positional, meaning that a simple careless transposition of two lines may produce an unexpected aggregation without any notification of the problem.

In my Julia package, I treat aggregation as a simple summation of data and weighted summation of parameter values. The user only provides to function `aggregate_data()` mapping vectors for endowments, commodities and regions in the form of named vectors. Please note that, unlike in FlexAgg, the aggregation input is an actual Julia code and therefore the input is generally not compromised if the lines are exchanged. Also, because I am using a named vector where each element has its own name, the mapping instruction may include actual element names, e.g., `regMap["usa"] = "United States"` to make the original region labeled `usa` map to a new region labeled `United States`, further reducing any unintentional errors.

As I mentioned earlier, there is a bug in the current aggregation software provided by the GTAP Center, which makes some of the region-specific parameters region-generic ones (e.g., `ESBV`, `ESBD`, `ESBM`). To allow for identically aggregated parameters in comparison exercises, I also include the function `aggregate_data_legacy()` which implements the same error and thus produces parameters identical to those generated by either GTAPAgg2 or FlexAgg.

Defining a data aggregation

The first step is to create named vectors for regions, commodities and endowments, which map each original name to the new name. The initial values of those vectors will be one-to-one mappings, where each original element will be mapped to itself.

```
regMap = NamedArray(sets["reg"], sets["reg"])
comMap = NamedArray(sets["comm"], sets["comm"])
endMap = NamedArray(sets["endw"], sets["endw"])
```

In the next steps, I can assign different names to individual, or ranges or original elements, using positions, ranges of positions or enumerated names:

```
regMap[1:3] .= "oceania"
regMap[4:27] .= "asia"
regMap[28:55] .= "americas"
regMap[56:82] .= "eu"
regMap[83:98] .= "other europe"
regMap[99:121] .= "mena"
regMap[122:160] .= "subsaharan africa"

comMap[1:8] .= "crops"
comMap[9:12] .= "animals"
comMap[13:18] .= "extract"
comMap[19:26] .= "processed food"
comMap[27:45] .= "manuf"
comMap[46:65] .= "svces"

endMap[:] .= "other"
endMap[1:1] .= "land"
endMap[[2,5]] .= "skilled labor"
endMap[[3,4,6]] .= "unskilled labor"
endMap["capital"] = "capital"
```

Finally, I can use function `aggregate_data()` to aggregate the data into named tuple that includes all data, parameters and sets:

```
(; hData, hParameters, hSets) =
    aggregate_data(hData=data, hParameters=parameters, hSets=sets,
                  comMap=comMap, regMap=regMap, endMap=endMap)
```

Data cleaning

The GTAP database sometimes contains values which are extremely close to zero, which are an artifact of the way are data processed. To allow for easy numeric solutions, it is important to make those almost zeros actual structural zeros. For sure, the value of imports of coal from Honduras to Mozambique was not a three-millionth of a US cent, as reported in version 11 of the standard GTAP database.

Structural zeros represent a problem for some models when they result in a division/log of zero. This is the reason they are often replaced with small values. In my implementation of the model in Julia, I simply exclude the structural zeros from the equations. However, because the model is expressed

in levels, I am able to turn structural zeros into actual values, which means that the model is able to simulate a change from an initial zero flow. An example of this approach would be to allow for a currently missing bilateral trade flow by changing the distribution parameter of the Armington preference function from 0 to a positive value.

Removing fictitious trade

```
# Remove all trade that is less than 100 US dollars
valid_trade = hData["vxsb"].>1e-4
for k in ["vcif", "vmsb", "vfob", "vxsb"]
    hData[k][.!valid_trade].=0
end
```

Initial model

The initial model is constructed with function `generate_initial_model()`. This function returns an object, `model_container`, that includes the model, i.e., equations and variables (field `model`), parameters (field `parameters`), sets (field `sets`), the default closure with the specifications which variables are fixed in the model (field `fixed`) and the initial data (field `data`) populated with the starting values for all variables in the model. Reasonable starting values are needed for each variable for the model the model to solve efficiently.

Modification of the model

The model container makes available to the user the actual JuMP model as well as all of the required data. This allows the experienced user to access and modify the model as needed, using standard JuMP commands. The user is thus able to remove and add variables and equations. Naturally, some changes to the model may require changes to the calibration procedure as it assumes the structure of the model as contained in GTAP v 7.

Unlike the case of a linearized model solved in GEMPACK where a vector of zeros is always a starting solution, in the case of levels equations, the solution is non-trivial and it typically does not include many zeros as most of the variables, e.g., prices and quantities, need to be positive. To help the solver find the solution, we identify a set of non-trivial initial values for each variable near the expected solution values.

Generating the initial model container

```
# Starting data and parameters
mc = generate_initial_model(hSets=hSets, hData=hData, hParameters=hParameters)
```

Note that after setting up the initial model with the starting values, the model is unlikely to be in a solved state, as the initial values for all unknown variables are very simple guesses. However, all of the values which are known because they are provided in the GTAP database such as tax rates, values of trade, and others are correct. For this reason the data object of the `model_container` is useful, as we can use it in calibration.

💡 Data for calibration

```
# Save the start data as they represent the original GTAP values for the
# known flows and rates
start_data = deepcopy(mc.data)
```

After saving the starting data, which contain the original GTAP flows and rates, it is important to solve the initial model, using function `run_model!(model_container)` before proceeding to the calibration so that calibration can start from an actual solution. Please note the exclamation point after the function name: this is a convention in Julia requiring that functions that change the values of the argument be marked so. After running the model, the values in the field `data` are populated with the solved values.

💡 Solving the initial model and updating values

```
# Solve the model and update data
run_model!(mc)
```

Model calibration

Unlike the linearized version of the GTAP model implemented in GEMPACK, my version in Julia is in levels, which means that similarly to the levels model representation in GAMS (Mensbrugghe 2018) additional distributional and scaling parameters need to be specified in order to generate the desired starting values of the model.

The target values of the calibration are the values recorded in the standard GTAP database. It is important to note that there are no quantities or prices in the database. The targets for calibration are therefore only the value flows, e.g., value of trade CIF, found in the database.

Calibration is done by the same model that is used for simulations, except we use a different closure, i.e. a different dictionary of which values are fixed (`fixed` field) and we specify the values of the variables that should be fixed. While in a simulation, the calibration parameters are fixed and the values of flows are endogenous, to perform the calibration, the values are fixed while the calibration parameters are made endogenous. This means that during calibration, the model solves for the parameters subject to the initial value flows.

For convenience, I have included function `generate_calibration_inputs()` which generates the calibration closure (field `fixed`) and calibration data (field `data`) which blends the initial solution with the desired levels for the variables.

💡 Generating calibrated data

To produce the calibrated values, the user needs to provide the `start_data` data object that has the desired values. Also provided are: `data`, a data object with a solved, uncalibrated model; `sets` a data object with the set definitions; `parameters`, a data object with all parameter values; and `fixed` which has the default closure.

```
# Get the calibration closure and data
(;fixed_calibration, data_calibration) =
  generate_calibration_inputs(mc, start_data)
```

Before running the calibration, it is useful to make a copy of the default closure so that we can use it for actual simulations later:

```
fixed_default = deepcopy(mc.fixed)
```

Then we can run the calibration by replacing the closure and data and running the model using function `run_model!()` which is described below:

```
# Replace the solved data with the target data
mc.data = data_calibration

# Replace the standard closure with the one for calibration
mc.fixed = fixed_calibration

# Solve the model
run_model!(mc)
```

After completing the calibration, it is useful to make a copy of the calibrated data, as they are likely to be the user's starting point for each simulation:

```
calibrated_data = deepcopy(mc.data)
```

Finally, we can replace the calibration closure with the standard one:

```
mc.fixed = fixed_default
```

Running simulations in the model

As we saw in the section on calibration, we run the model using function `run_model!(model_container)` which executes several internal functions. First, the values from the `data` object are loaded as initial values of the model. Second, the closure is imposed on the model based on the information in object `fixed` which is a dictionary where each element of `data` which should be fixed is listed as a named array of the same size, with Boolean values indicating whether a particular value is fixed or not. Those variables that are not fixed, i.e., endogenous, need not be specified in object `fixed`. However, those variable which have at least one component fixed must be included with those elements that are not fixed set to `false`. As the third step, the function finds the solution of the model. Finally, the function reads all solution values from the model and updates object `data`.

It is the user's responsibility to assure that the model remains square with the selection of the closure, i.e., the number of free variables must equal the number of constraints. Unlike in GEMPACK, where only a square closure can be mathematically executed and therefore a wrong closure results in error, in GAMS and Julia/JuMP it is important to pay attention to the standard output of the solver, which indicates both numbers. In case of a wrong closure, Julia/JuMP would attempt to find a solution but it is likely that the solution may not exist or may not be unique. Also, looking for a solution in a wrong closure is likely to result in much longer run times and greater demands on computer's memory—the solver may relax some of the constraints, increasing the size of the free variable space—, giving the user an opportunity to notice the problem.

The `run_model!()` function also accepts three optional parameters. The first, `max_iter` limits the number of solution iterations. The default value is 50.⁷ The second, `constr_viol_tol` has a default value of `1e-5` that defines the level at which constraints are considered to be satisfied. Finally, `bound_push` defines how close can the starting values come to the boundaries of the variables. It is recommended that this be as close to zero as possible for a typical GTAP database where some flows are extremely close to zero and it is fine for variables to be close to zero as well.

The `run_model!()` function does not return any values; instead, it modifies the model container, e.g., it will modify the data associated with the model by replacing all values with the model's solutions. It is important to remember that once the simulation has completed, the `data` object in the container has been updated, and therefore the user should make copies of the solution data that are of interest as each further run will replace the previous solved state.

Running a simulation

To run a simulation, the user only needs to update the model container with the correct data (object `data`) and the appropriate closure (`fixed`). For example, if I would like to increase the population of `oceania` by 10 percent, I would need to do the following steps:

⁷For most model and calibration runs, the model should quickly converge and it is unlikely that more than 50 iterations are needed; in those cases, where the model appears not to converge, it is a better approach to split the shock into components so that the solution is not too far from the initial state.

```

# Load the calibrated data
mc.data = deepcopy(calibrated_data)

# Load the default closure
mc.fixed = deepcopy(default_fixed)

# Change the population in oceania
mc.data["pop"]["oceania"] .= mc.data["pop"]["oceania"] * 1.1

# Run the model
run_model!(mc)

```

Viewing the solution

Viewing of the results of the model run is best done interactively because once the model has finished running, all results are available in memory for viewing or further processing, e.g., saving, visualizing or publishing. This applies not only to the values of the variables, but also equations.

Within Julia, a modern, interactive system, there is little need for helper tools, such as AnalyseGE or ViewHAR, whose only job is to display the arrays of values such as those produced by the model run. Such tools made sense in the old days of mainframe computers where the user would submit a job and only collect the results without any interaction with the model. Because GEMPACK and GAMS adopted this paradigm of no interaction, special tools for post-processing were needed. As we explain above, modern software tends to be much more interactive, allowing the view solutions in memory.

Because the model in Julia is a mutable (modifiable) object that changes with each solution, I highly recommend saving the results of a simulation in a separate object so that the user can look at as many solutions as needed.

Viewing the results in Julia

If I am, for example, interested in a percentage change to the consumer price index, I could easily calculate it based on the original and updated values.

First, I would save the results of my simulation:

```
sim_data = deepcopy(mc.data)
```

Then I can compare the solution to the original state with ease:

```
round.(
  ((sim_data["ppriv"]./calibrated_data["ppriv"])[:] .- 1) .* 100;
  digits = 2
)
```

Some key differences from the version of GTAP version 7 in GEMPACK

Even though the version of the GTAP version 7 model in Julia produces nearly identical results to those generated by GEMPACK, there are some differences in implementation that I discuss here in more detail. Most of the differences relate to those variables that measure a change between simulations. Because the levels model is not calculating changes in variables, but rather the solution values, percentage changes are best calculated outside the model. Julia allows for multiple model solutions in memory, making such calculations very practical for all variables included in the model. Some other variables typically calculated in the linearized model, such as change in welfare or real GDP; however, are a bit more involved.

Welfare calculations

In the case of equivalent variation, for example, it is necessary to solve for income using one solution's prices to reach another solution's utility. The first difference between the levels and change formulation of the model is that the levels formulation cannot do EV calculations in the model. Again, this is due to the fact that variables from two solutions—(1) the state of the world before the shock and (2) the resulting state of the world after the shock—need to be used to calculate the equivalent variation between the solutions. While EV calculation cannot be a part of the levels formulation of the model, it is relatively trivial to calculate it after running the model based on the pre-simulation and post-simulation prices and utility.

For the user's convenience, I include in the package the function `calculate_ev` which calculates the income needed to achieve the new level of utility with original prices. Essentially this function solves just the utility part of the model, identifying the level of income y to reach the desired value of utility u , with the original prices for savings ($psave$), household consumption (ppa) and government consumption (pga).

Calculating equivalent variation

Assuming my initial state of the model is represented by the data dictionary `calibrated_data` while the scenario state is represented by the data, I can use function `calculate_ev()` to calculate the equivalent variation:

```
ev = calculate_ev(
    sets=sets,
    data0=calibrated_data,
    data1=data,
    parameters=parameters
)
```

I can then view the value of EV for each region, by comparing the original income to the income required, at original price, to reach the new utility level:

```
ev .- calibrated_data["y"]
```

Real GDP index

In the case of real GDP index, it is necessary to evaluate the quantity of GDP in one solution using prices from another solution. Similarly to the calculation of EV, it is not convenient to define the real GDP index in the model as its calculation mixes values of different solutions—original and updated quantities and original prices—and is also best calculated after the model is solved. In the case of real GDP, I evaluate the quantities of the components of the GDP at pre-simulation prices. For the user's convenience I provide function `calculate_gdp()` that calculates the value of GDP at different solution's prices.

It is important to note that in the GEMPACK version of the GTAP model, variable `qgdp` does not represent the change in GDP at pre-simulation prices because the equation includes GDP weights that are updated with price changes. Instead of a price deflated GDP growth, `qgdp` in GEMPACK measures the average of GDP growth at pre-simulation and post-simulation prices.

Calculating change in real GDP

Assuming my initial state of the model is represented by the data dictionary `calibrated_data` while the scenario state is represented by the `data`, I can use function `calculate_gdp` to calculate the value of GDP at different price vectors:

```
qgdp1 = calculate_gdp(sets=sets, data0=calibrated_data, data1=data)
qgdp0 = calculate_gdp(sets=sets, data0=calibrated_data, data1=calibrated_data)
```

I can then view the change in the value of GDP at original prices:

```
(qgdp1 ./ qgdp1 .- 1) .* 100
```

Convenience indices

The second difference is that I do not calculate many of the index variables which are sometimes used, but these can be easily calculated after the model is solved (e.g., `pwu` measuring world price index for commodities, etc.) Understandably, in GEMPACK and GAMS, it is fairly difficult to calculate a new variable based on the solution values and for that reason the GTAP model in GEMPACK contains a wide range of price, quantity and value indices, but this is really not necessary in Julia, where it is easy to calculate any index based on the solution results.

Comparison of model results

In this section, I perform a few sample simulations using GTAP v7 in Julia and compare the results with those obtained by the GEMPACK version. In all examples, I use GTAP database 11 and aggregate it into 15 regions, 6 commodities and 5 factors shown in Table 1, Table 2, and 3. The closure is long-run, with capital and labor mobile and land sluggish. The investment funds are allocated proportionally ($RORDELTA = 0$). Finally, instead of using the default numeraire of `pfactwld`, I choose the consumer price of the first commodity in the first region (crops in oceania) to be the numeraire. Even though the choice of numeraire is irrelevant for the analysis results, GEMPACK's limited precision in calculating `pfactwld` means that all prices would be slightly shifted by a tiny factor of about 0.0001.

Table 1: Regional aggregation

Aggregated region	Included GTAP regions
oceania	aus, nzl, xoc
easia	chn, hkg, jpn, kor, mng, twm, xea
seasia	brn, khm, idn, lao, mys, phl, sgp, tha, vnm, xse
sasia	afg, bgd, ind, npl, pak, lka, xsa
namerica	can, usa, mex, xna
samerica	arg, bol, bra, chl, col, ecu, pry, per, ury, ven, xsm
carib	cri, gtm, hnd, nic, pan, slv, xca, dom, hti, jam, pri, tto, xcb
eu	aut, bel, bgr, hrv, cyp, cze, dnk, est, fin, fra, deu, grc, hun, irl, ita, lva, ltu, lux, mlt, nld, pol, prt, rou, svk, svn, esp, swe
oeurope	gbr, che, nor, xef, alb, srb, blr, rus, ukr, xee, xer, kaz, kgz, tjk, uzb, xsu
wasia	arm, aze, geo, bhr, irn, irq, isr, jor, kwt, lbn, omn, pse, qat, sau, syr, tur, are, xws
mena	dza, egypt, mar, tun, xnf
wafrica	ben, bfa, cmr, civ, gha, gin, mli, ner, nga, sen, tgo, xwf
scafrica	caf, tcd, cog, cod, gnq, gab, xac
eafrica	com, eth, ken, mdg, mwi, mus, moz, rwa, sdn, tza, uga, zmb, zwe, xec
safrica	bwa, swz, nam, zaf, xsc, xtw

Table 2: Commodity aggregation

Aggregated commodity	Included GTAP commodities
crops	pdr, wht, gro, v_f, osd, c_b, pfb, ocr
animals	ctl, oap, rmk, wol
extract	frs, fsh, coa, oil, gas, oxt
food	cmt, omt, vol, mil, pcr, sgr, ofd, b_t
manuf	tex, wap, lea, lum, ppp, p_c, chm, bph, rpp, nmm, i_s, nfm, fmp, ele, eeq, ome, mvh, otn, omf
svces	ely, gdt, wtr, cns, trd, afs, otp, wtp, atp, whs, cmn, ofi, ins, rsa, obs, ros, osg, edu, hht, dwe

Table 3: Endowment aggregation

Aggregated endowment	Included GTAP endowments
land	land
sklabor	tech_aspros, off_mgr_pros
unsklabor	clerks, service_shop, ag_othlowsk
capital	capital
other	natlres

Increasing tariffs

In this scenario I double the power of tariff on imports of crops and food from mena and safrica to eu. I calculate the model results in GEMPACK as well as Julia and compare the results.

Because I can use the standard closure, the only thing I need to change is the value of tariff tms on crops and food from mena and safrica to eu:

```
# Change the power of tariff in the model's data (model container = mc)
mc.data["tms"][["crops", "food"], ["mena", "safrica"], "eu"] .=
    mc.data["tms"][["crops", "food"], ["mena", "safrica"], "eu"] * 2

# Run the model
run_model!(mc)
```

An equivalent piece of GEMPACK code in the CMF statement is:

Table 4: Imports into eu (qxs[:, :, eu]) under the tariff increase scenario (percent changes)—calculated in GEMPACK

	crops	animals	extract	food	manuf	svces
oceania	4.645720	1.131616	-0.062836	1.465049	-0.020744	-0.007522
asia	4.846148	1.251953	-0.070315	1.522812	0.010741	0.017948
seasia	4.643625	1.127508	-0.062502	1.443421	0.001047	0.010836
sasia	4.831641	1.238156	-0.065924	1.513173	0.004723	0.013956
namerica	4.742795	1.190955	-0.075817	1.495095	-0.006687	0.006514
samerica	4.603964	1.082161	-0.067983	1.409869	-0.080180	-0.045359
carib	4.201101	0.862345	-0.069890	1.307597	-0.074903	-0.040736
eu	2.894137	0.085623	-0.050559	0.804992	-0.062655	-0.033006
oeurope	4.755982	1.143054	-0.028126	1.473276	-0.019779	-0.005061
wasia	4.818307	1.243936	-0.056017	1.538462	0.020322	0.021320
mena	-95.686043	11.231926	0.017857	-96.293419	2.774672	2.069345
wafrica	4.113976	0.718111	-0.099015	1.350469	-0.154081	-0.082714
scafrica	4.802666	1.242593	-0.074465	1.514383	0.016312	0.016752
eafrica	4.758133	1.251944	-0.135192	1.550390	0.034350	0.022665
safrica	-95.868500	11.918674	0.061139	-96.283157	1.656899	1.419624

```
xset targetcom(crops, food);
xsubset targetcom is subset of comm;
xset targetreg(mena, safrica);
xsubset targetreg is subset of reg;
shock tms(targetcom, targetreg, "eu") = uniform 100;
```

First, I show the impact of the simulation on the volume of trade to the EU in Table 4 and Table 5. In Table 6, I show the differences between the solutions. As Table 6 documents, the differences between solutions are negligible and both solutions are identical to five significant figures.

I next move to the changes in consumer price, which are shown in Table 7 and Table 8. In Table 9 I show the differences between the two solutions, again, finding that any differences are negligible.

Finally, I report the equivalent variation for the simulation in GEMPACK and Julia in Table 10 as well as, visually, in Figure 2. Once again, the results are very similar. The reason for less accuracy is that EV reports the change in national incomes and the starting values are in billions of dollars. Thus, interacting such large values with percent change variables where there are tiny variations may produce noticeable difference in the reported values, even though the difference is still relatively small.⁸

⁸For example, in the case of wasia the initial income is \$589,225 million and a change of 1.7 million or 2.4 million is still within an accuracy beyond the fifth significant figure.

Table 5: Imports into eu (qxs[:, :, eu]) under the tariff increase scenario (percent changes)—calculated in Julia

	crops	animals	extract	food	manuf	svces
oceania	4.645723	1.131616	-0.062836	1.465049	-0.020744	-0.007522
easia	4.846149	1.251953	-0.070315	1.522812	0.010740	0.017947
seasia	4.643625	1.127508	-0.062502	1.443421	0.001047	0.010835
sasia	4.831643	1.238156	-0.065924	1.513173	0.004723	0.013956
namerica	4.742797	1.190955	-0.075817	1.495096	-0.006688	0.006514
samerica	4.603967	1.082160	-0.067983	1.409869	-0.080181	-0.045360
carib	4.201103	0.862345	-0.069890	1.307597	-0.074904	-0.040737
eu	2.894137	0.085624	-0.050558	0.804992	-0.062654	-0.033006
oeurope	4.755984	1.143054	-0.028126	1.473276	-0.019779	-0.005061
wasia	4.818309	1.243936	-0.056017	1.538462	0.020322	0.021320
mena	-95.686042	11.231937	0.017858	-96.293415	2.774676	2.069349
wafrica	4.113972	0.718107	-0.099014	1.350469	-0.154081	-0.082715
scafrica	4.802668	1.242593	-0.074465	1.514383	0.016311	0.016751
eafrica	4.758135	1.251944	-0.135192	1.550390	0.034350	0.022664
safrica	-95.868501	11.918666	0.061138	-96.283163	1.656898	1.419624

Table 6: Imports into eu (qxs[:, :, eu]) under the tariff increase scenario (percent changes)—difference between solutions

	crops	animals	extract	food	manuf	svces
oceania	-0.000002	0.000000	-0.000000	-0.000000	0.000000	0.000000
easia	-0.000001	-0.000000	-0.000000	-0.000000	0.000001	0.000000
seasia	-0.000000	0.000000	-0.000000	0.000000	0.000000	0.000000
sasia	-0.000002	0.000000	-0.000000	-0.000000	0.000000	0.000000
namerica	-0.000002	0.000000	-0.000000	-0.000000	0.000000	0.000000
samerica	-0.000003	0.000000	-0.000000	-0.000000	0.000001	0.000000
carib	-0.000001	-0.000000	0.000000	0.000000	0.000001	0.000000
eu	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
oeurope	-0.000002	-0.000000	-0.000000	-0.000000	0.000000	0.000000
wasia	-0.000002	-0.000000	-0.000000	-0.000000	0.000000	0.000000
mena	-0.000000	-0.000011	-0.000001	-0.000004	-0.000005	-0.000004
wafrica	0.000004	0.000004	-0.000000	0.000001	0.000000	0.000000
scafrica	-0.000002	0.000000	-0.000000	0.000000	0.000000	0.000000
eafrica	-0.000002	0.000000	-0.000000	-0.000000	0.000000	0.000000
safrica	0.000002	0.000007	0.000000	0.000006	0.000001	0.000000

Table 7: Change in consumer prices ppa under the tariff increase scenario (percent changes)—calculated in GEMPACK

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.012226	-0.055975	-0.031178	-0.051528	-0.047361
easia	-0.040069	-0.043484	-0.055476	-0.045789	-0.053872	-0.053749
seasia	-0.005397	-0.007877	-0.055986	-0.031084	-0.052723	-0.052043
sasia	-0.038710	-0.039500	-0.055660	-0.045461	-0.053297	-0.052851
namerica	-0.016168	-0.024345	-0.054936	-0.040041	-0.051843	-0.050810
samerica	0.011225	0.000701	-0.055528	-0.024102	-0.043830	-0.037590
carib	0.088688	0.054960	-0.055376	-0.005694	-0.047031	-0.039396
eu	0.725898	0.298214	-0.057513	0.176557	-0.047976	-0.040758
oeurope	-0.029478	-0.005673	-0.058465	-0.020318	-0.050107	-0.047872
wasia	-0.050307	-0.036652	-0.056626	-0.049112	-0.055899	-0.054855
mena	-2.879804	-2.419444	-0.062222	-0.778171	-0.328138	-0.566043
wafrica	0.104869	0.089115	-0.052910	-0.015849	-0.045489	-0.029381
scafrica	-0.035745	-0.041412	-0.054923	-0.051704	-0.062594	-0.054128
eafrica	-0.033494	-0.046364	-0.049988	-0.067354	-0.067534	-0.056751
safrica	-2.301165	-2.537153	-0.065760	-0.854684	-0.174701	-0.401317

Table 8: Change in consumer prices ppa under the tariff increase scenario (percent changes)—calculated in Julia

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.012226	-0.055975	-0.031178	-0.051528	-0.047361
easia	-0.040069	-0.043484	-0.055476	-0.045789	-0.053872	-0.053749
seasia	-0.005397	-0.007877	-0.055986	-0.031084	-0.052723	-0.052043
sasia	-0.038710	-0.039500	-0.055660	-0.045461	-0.053297	-0.052851
namerica	-0.016168	-0.024345	-0.054936	-0.040041	-0.051843	-0.050810
samerica	0.011225	0.000701	-0.055528	-0.024102	-0.043830	-0.037590
carib	0.088688	0.054960	-0.055376	-0.005694	-0.047031	-0.039397
eu	0.725898	0.298214	-0.057513	0.176557	-0.047976	-0.040758
oeurope	-0.029478	-0.005674	-0.058465	-0.020318	-0.050108	-0.047872
wasia	-0.050307	-0.036652	-0.056626	-0.049112	-0.055899	-0.054855
mena	-2.879808	-2.419447	-0.062222	-0.778173	-0.328139	-0.566044
wafrica	0.104870	0.089116	-0.052910	-0.015849	-0.045489	-0.029381
scafrica	-0.035745	-0.041412	-0.054923	-0.051704	-0.062594	-0.054128
eafrica	-0.033494	-0.046364	-0.049988	-0.067354	-0.067535	-0.056751
safrica	-2.301164	-2.537152	-0.065760	-0.854684	-0.174701	-0.401318

Table 9: Change in consumer prices ppa under the tariff increase scenario (percent changes)—
difference between solutions

	crops	animals	extract	food	manuf	svces
oceania	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
easia	0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000
seasia	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
sasia	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
namerica	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
samerica	-0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000
carib	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
eu	-0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000
oeurope	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
wasia	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
mena	0.000003	0.000003	0.000000	0.000001	0.000001	0.000001
wafrica	-0.000001	-0.000001	0.000000	-0.000000	0.000000	0.000000
scafrica	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
eafrica	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
safrica	-0.000001	-0.000000	0.000000	0.000000	0.000000	0.000000

Table 10: Equivalent variation under the tariff increase scenario

	GEMPACK	Julia	Difference
oceania	11.2	11.1	0.1
easia	-57.5	-57.4	-0.2
seasia	11.5	12.0	-0.6
sasia	10.1	10.9	-0.8
namerica	-4.5	-5.1	0.6
samerica	89.3	89.2	0.1
carib	33.9	33.9	0.0
eu	-2,221.6	-2,235.2	13.5
oeurope	-89.6	-90.3	0.7
wasia	1.7	2.4	-0.7
mena	-999.7	-1,003.8	4.2
wafrica	56.1	57.0	-0.9
scafrica	11.0	11.2	-0.2
eafrica	44.5	45.0	-0.5
safrica	-371.4	-371.7	0.4

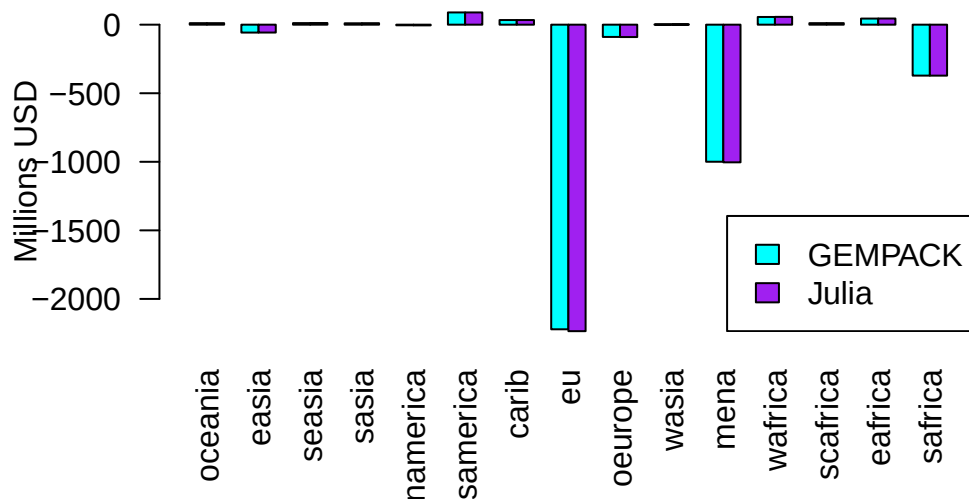


Figure 2: Equivalent variation of the tariff exercise

Implementing a technological shift

In this scenario, I implement a technological shift by shifting value-added productivity by 100 percent for service in Africa (regions wafrika, scafrica, eafrika, safrica):

```
target_regions = ["wafrika", "scafrica", "eafrika", "safrica"]
mc.data["y_qintva"][["svces"],target_regions]
    . = mc.data["y_qintva"][["svces"],target_regions] * 2
run_model!(mc)
```

An equivalent piece of the shock definition in GEMPACK is:

```
xset targetreg(wafrika, scafrica, eafrika, safrica);
xsubset targetreg is subset of reg;
shock aoall("svces", targetreg) = uniform 100;
```

I report the equivalent variation of the simulation in GEMPACK and Julia in Figure 3. It appears that even though both systems give very similar values for EV, some small differences are visible, again for the reason of multiplying tiny difference with huge income values.

⁹As mentioned in the text, the GDP index as reported in GEMPACK is the average of GDP growth at pre- and post-simulation pricess

Table 11: Change in consumer prices ppa under the technical change scenario— calculated in GEM-PACK

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.369135	0.495397	-0.931947	-1.088207	-1.305855
easia	-0.610777	-0.743825	0.056401	-0.922292	-1.077218	-1.234110
seasia	0.612364	-0.049209	0.365496	-0.591506	-1.156791	-1.547579
sasia	-0.251822	-0.472595	0.028674	-0.866392	-1.037379	-1.462329
namerica	-0.153189	-0.736558	0.291343	-1.164021	-1.236869	-1.493577
samerica	-0.191707	-0.315003	0.438658	-0.699029	-0.905499	-0.960589
carib	-0.516726	-0.826659	0.269928	-1.324964	-1.477581	-1.896633
eu	0.341384	-0.278079	0.473798	-1.146043	-1.386452	-1.652400
oeurope	0.187300	-0.404351	0.490894	-1.171005	-1.318076	-1.737743
wasia	-0.337988	-0.490333	0.496373	-0.913665	-1.097375	-1.400724
mena	0.160159	-0.120766	0.528209	-0.858505	-1.138458	-1.351908
wafrica	26.450390	34.909569	3.911168	9.161580	2.256921	-43.544868
scafrica	50.023254	39.620789	1.821708	18.499454	1.156086	-36.783501
eafrica	30.449890	37.149746	14.983858	13.336100	-4.463117	-43.306210
safrica	11.567289	25.396723	4.714528	8.119794	-2.121912	-44.794109

Table 12: Change in consumer prices ppa under the technical change scenario— calculated in Julia

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.369134	0.495396	-0.931945	-1.088206	-1.305852
easia	-0.610777	-0.743825	0.056400	-0.922291	-1.077217	-1.234109
seasia	0.612363	-0.049209	0.365495	-0.591505	-1.156789	-1.547576
sasia	-0.251821	-0.472595	0.028674	-0.866391	-1.037377	-1.462327
namerica	-0.153189	-0.736557	0.291342	-1.164018	-1.236867	-1.493574
samerica	-0.191706	-0.315001	0.438657	-0.699027	-0.905497	-0.960586
carib	-0.516725	-0.826657	0.269926	-1.324961	-1.477577	-1.896628
eu	0.341382	-0.278079	0.473797	-1.146041	-1.386450	-1.652397
oeurope	0.187299	-0.404350	0.490893	-1.171002	-1.318073	-1.737739
wasia	-0.337986	-0.490331	0.496372	-0.913662	-1.097373	-1.400721
mena	0.160158	-0.120766	0.528208	-0.858504	-1.138457	-1.351906
wafrica	26.450341	34.909545	3.911155	9.161718	2.256981	-43.544762
scafrica	50.023263	39.620800	1.821703	18.499462	1.156092	-36.783502
eafrica	30.449882	37.149746	14.983867	13.336098	-4.463109	-43.306213
safrica	11.567293	25.396730	4.714526	8.119797	-2.121904	-44.794119

Table 13: Change in consumer prices ppa under the technical change scenario—difference between solutions

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.000001	0.000001	-0.000002	-0.000001	-0.000003
easia	-0.000000	0.000000	0.000000	-0.000001	-0.000001	-0.000001
seasia	0.000001	-0.000000	0.000001	-0.000001	-0.000002	-0.000003
sasia	-0.000000	-0.000000	0.000000	-0.000001	-0.000002	-0.000002
namerica	-0.000000	-0.000001	0.000001	-0.000002	-0.000002	-0.000003
samerica	-0.000001	-0.000001	0.000001	-0.000002	-0.000002	-0.000003
carib	-0.000001	-0.000002	0.000002	-0.000004	-0.000003	-0.000006
eu	0.000002	0.000000	0.000001	-0.000002	-0.000002	-0.000002
oeurope	0.000001	-0.000001	0.000001	-0.000003	-0.000002	-0.000004
wasia	-0.000002	-0.000002	0.000001	-0.000003	-0.000002	-0.000003
mena	0.000001	-0.000000	0.000002	-0.000002	-0.000002	-0.000002
wafrica	0.000049	0.000024	0.000012	-0.000138	-0.000060	-0.000106
scafrica	-0.000008	-0.000012	0.000005	-0.000008	-0.000006	0.000001
eafrica	0.000009	0.000000	-0.000008	0.000002	-0.000007	0.000003
safrica	-0.000004	-0.000007	0.000002	-0.000003	-0.000008	0.000010

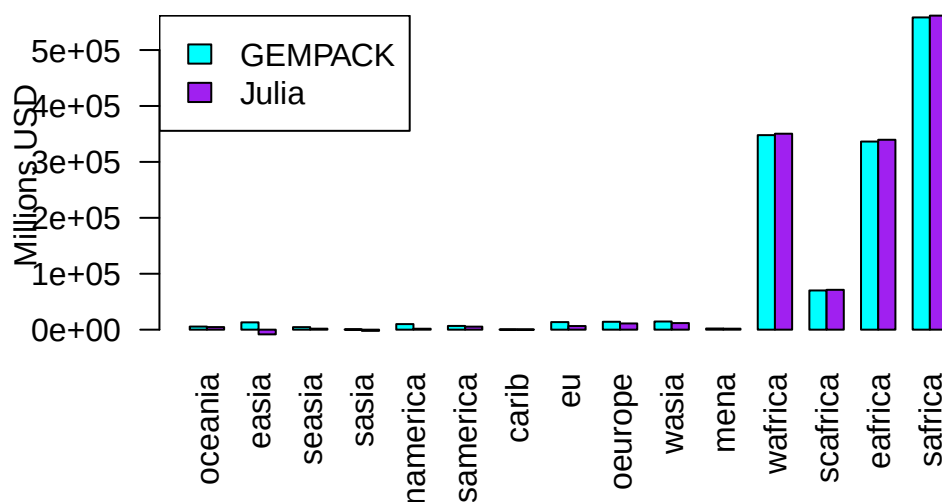


Figure 3: Equivalent variation of the technical change exercise

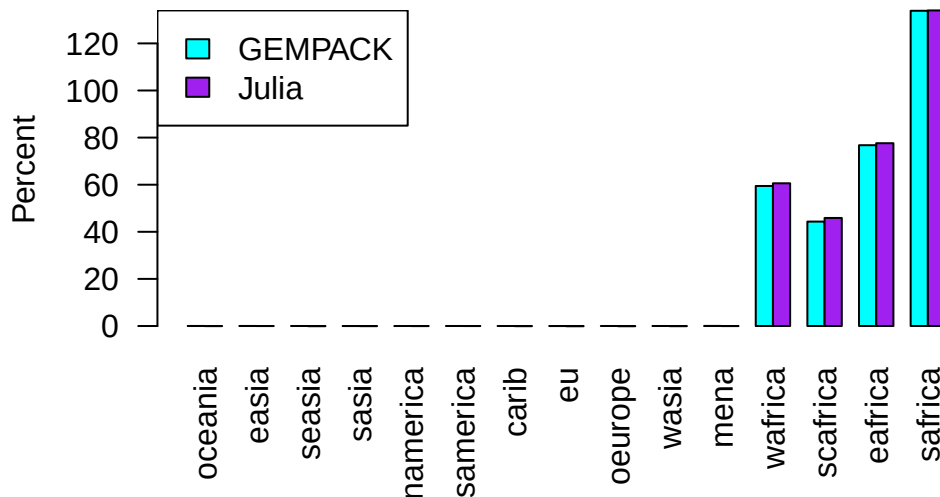


Figure 4: GDP quantity index change of the technical change exercise⁹

Reducing output through an output tax

In this scenario, I demonstrate that I am able to swap variables, i.e., turn an exogenous variable endogenous while another endogenous variable becomes exogenous, very easily and solve the model to find an output tax that brings an ordinarily endogenous output level to a target level. I reduce the output of extract in eu by five percent (variable qc) by targeting output tax (variable to).

The only thing that I need to do to implement this scenario is the change the closure and provide the desired level of qc:

```
# The default `fixed` dictionary has the standard closure; I need to change it
# Make the element in output tax "to" for extract produced by extract activity
# in eu false to make it endogenous
mc.fixed["to"]["extract","extract","eu"] = false

# Create a new array of false values, to make all qc's endogenous so that I
# can pass it to the model
mc.fixed["qc"]=NamedArray(falses(size(mc.data["qc"])), names(mc.data["qc"]))

# Fix "qc" for extract in eu only
mc.fixed["qc"]["extract","eu"] = true
```

Table 14: Calculated consumer price ppa changes in the output tax scenario—GEMPACK

	crops	animals	extract	food	manuf	svces
oceania	0.000000	0.004430	0.255693	0.016901	0.028409	0.030468
easia	-0.015587	-0.016801	0.189305	-0.006583	0.010101	-0.012059
seasia	-0.014324	-0.015040	0.230023	-0.004830	0.014362	-0.009360
sasia	-0.034315	-0.038660	0.184554	-0.027335	0.017251	-0.026056
namerica	-0.001822	-0.010031	0.223453	-0.006629	0.012107	-0.009442
samerica	0.000208	0.002701	0.261641	0.014428	0.032069	0.013798
carib	-0.013927	-0.013585	0.244534	-0.007857	0.008372	-0.013154
eu	-0.040020	-0.050384	0.733545	-0.053589	-0.014226	-0.078133
oeurope	-0.020752	-0.018545	0.386123	-0.012924	0.007496	-0.016316
wasia	-0.004583	-0.001910	0.303559	0.001380	0.033445	0.012052
mena	-0.000512	0.007102	0.397443	0.012158	0.047412	0.021394
wafrica	0.021530	0.029061	0.327594	0.038725	0.035195	0.041888
scafrica	0.122876	0.099745	0.268964	0.069461	0.055719	0.060312
eafrica	0.000786	0.005506	0.197519	0.008320	0.023858	0.013873
safrica	-0.009242	-0.018318	0.228729	0.001678	0.039614	-0.001191

```
## Reduce output by 10 percent
mc.data["qc"]["extract","eu"] := mc.data["qc"]["extract","eu"] * 0.95

# Perform the simulation
run_model!(mc)
```

An equivalent swap and shock definition in GEMPACK is:

```
swap qc("extract","eu") = to("extract","extract","eu");
shock qc("extract","eu") = -5;
```

I report the obtained changes in consumer prices in Table 14 for GEMPACK, in Table 15 for Julia and all differences in Table 16. Clearly, even in this case, the difference between the price impacts are negligible, beyond the third decimal.

Changes to the model

In GEMPACK or GAMS, it is not possible to extend an existing model as these systems were not meant to be interactive. To make a change to the model in both GEMAPCK, the modeler needs to write

Table 15: Calculated consumer price ppa changes in the output tax scenario—Julia

	crops	animals	extract	food	manuf	svces
oceania	0.000000	0.004430	0.255693	0.016901	0.028409	0.030468
easia	-0.015587	-0.016801	0.189305	-0.006583	0.010101	-0.012059
seasia	-0.014324	-0.015040	0.230023	-0.004830	0.014362	-0.009360
sasia	-0.034315	-0.038660	0.184554	-0.027335	0.017251	-0.026056
namerica	-0.001822	-0.010031	0.223453	-0.006629	0.012107	-0.009442
samerica	0.000208	0.002701	0.261641	0.014428	0.032069	0.013798
carib	-0.013927	-0.013585	0.244534	-0.007857	0.008372	-0.013154
eu	-0.040020	-0.050384	0.733546	-0.053589	-0.014226	-0.078133
oeurope	-0.020752	-0.018545	0.386123	-0.012924	0.007496	-0.016316
wasia	-0.004583	-0.001910	0.303559	0.001380	0.033445	0.012052
mena	-0.000512	0.007102	0.397443	0.012158	0.047412	0.021394
wafrica	0.021530	0.029061	0.327594	0.038725	0.035195	0.041888
scafrica	0.122876	0.099745	0.268965	0.069461	0.055719	0.060312
eafrica	0.000786	0.005506	0.197519	0.008320	0.023858	0.013873
safrica	-0.009241	-0.018318	0.228729	0.001678	0.039614	-0.001191

Table 16: Calculated consumer price ppa changes in the output tax scenario—differences

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
easia	0.000000	0.000000	-0.000000	0.000000	-0.000000	0.000000
seasia	0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.000000
sasia	0.000000	0.000000	-0.000000	0.000000	-0.000000	0.000000
namerica	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
samerica	-0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.000000
carib	0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.000000
eu	0.000000	0.000000	-0.000000	0.000000	-0.000000	0.000000
oeurope	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
wasia	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
mena	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
wafrica	0.000000	0.000000	-0.000000	-0.000000	-0.000000	-0.000000
scafrica	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
eafrica	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
safrica	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000

a completely new model with no persistent connection to the one on which it is based.¹⁰ In GAMS, it is also not possible to modify a model, but it is possible to add new equations and define a new model using the equations in the same script. However, this is not a practical way of creating links between model and it is rarely used.

Extending models in Julia is far more tractable as the Julia language and the associated math modeling packages expect an interactive engagement with the model. It is therefore possible to add new variables and equations to the model and even delete them, using the standard functions within package JuMP.

The feature of JuMP models in Julia that they can be further extended opens important new possibilities for developing new versions of GTAP models in the future where they can maintain a link to the canonical version 7. A simple addition or modification of a few equations in a model in Julia does not require to document an entire model; instead, the author of the extended model would only need to describe the added or modified equations as the rest of the model would clearly come from the original version.

💡 Making land supply endogenous

Let me demonstrate how a simple modification to the model can be achieved in Julia. Let's modify the assumption of the standard model that land supply is exogenous by replacing this assumption with a constant elasticity supply equation:

$$qe_{land} = \alpha_{land} \times \left(\frac{pe_{land}}{ppriv} \right)^{\sigma_{land}}$$

If this is the only change that we would like to do, all that is necessary is to:

Add new variables to the model—one for the scaling parameter and one for the elasticity:

```
reg= mc.sets["reg"]
@variables(mc.model,
    begin
        land_scale[reg]
        land_elasticity[reg]
    end
)
```

Add the equation:

¹⁰Naturally, most of the extended model is likely to be a copy of the original model; however, without a careful comparison of the source codes, it is not possible to see how the model was extended.

```

@constraints(mc.model,
    begin
        e_qo_land[r=reg], log(mc.model[:qe]["land", r]) ==
            log(land_scale[r] *
                (mc.model[:pe]["land",r] /
                 mc.model[:ppriv][r]) ^ land_elasticity[r])
    end
)

```

Next, I calibrate my model. For the sake of exercise, let me assume that the land supply elasticity is -1:

```

# Calibrate the new parameters
mc.data["land_elasticity"] = -1 .* NamedArray(ones(length(reg)), reg)

```

Then I calculate the calibrated value of the scaling factor:

```

mc.data["land_scale"] =
    mc.data["qe"]["land",:] ./ ((mc.data["pe"]["land",:] ./
                                mc.data["ppriv"]).^mc.data["land_elasticity"])

```

Finally, I make qe of land endogenous, and make my parameters exogenous:

```

mc.fixed["land_elasticity"] =
    NamedArray(trues(size(mc.data["land_elasticity"])), reg)
mc.fixed["land_scale"] = NamedArray(trues(size(mc.data["land_scale"])), reg)
mc.fixed["qe"]["land",:] = false

```

And I can run the modified model:

```

run_model!(mc)

```

Run time comparison

A comparison of run times between GEMPACK and Julia is not completely straightforward as both systems require slightly different steps and operate at different levels of precision. In general, GEMPACK produces solutions faster than Julia with several important caveats. First, GEMPACK is not capable of assuring precision beyond the sixth decimal for all variables. In each sample solution presented above, there were observed several variables with 0 significant figure accuracy. Second, larger shocks tend to increase the computing time in GEMPACK much more than in Julia because GEMPACK must trace

out every solution's curve from the starting, while Julia—working in levels—may be able to find the solution in a smaller number of iterations. This is also true when adding another small shock to an existing simulation where the model in Julia is likely to reach the updated solution in fewer iterations because it starts from the previous solution.

Conclusion

In this paper, I demonstrate that it is possible to solve the standard GTAP version 7 model in Julia with virtually identical results to those obtained in GEMPACK. All of the presented percent change results match to the fifth significant digit between the two solutions for the three simple scenarios presented in the paper. For change variables, such as welfare, which involve the multiplication of a percent change variable by a value flow, the difference relative to the size of the flow are also equivalent beyond the fifth significant figure.

Even though for the set of presented scenarios the model in Julia produces the same results as GEMPACK, my version of the model is expressed in levels which makes it suitable for analysis of additional types of policies that cannot be addressed by the linearized version of the model in GEMPACK such as those that involve creation of new value flows or changes in parameters.

The GTAP model in Julia described in this paper is not the first GTAP model expressed in levels; however, it is novel in maintaining variable and equation names found in the original GTEMPACK formulation. This makes it easier for anyone who is familiar with the GEMPACK version to use my levels version in Julia.

By enabling the GTAP model to be run in Julia I believe that I have reduced two important obstacles in using the model by researchers around the world. First, I have eliminated the cost of licenses for either GEMPACK or GAMS since Julia is open-source. Second, by moving to Julia—a modern general-purpose computing language—I have enabled the GTAP model to be integrated into modern research workflows.

I demonstrate that performing GTAP analysis in Julia is much more straightforward than in GEMAPCK as there is no need to install additional stand-alone programs—all of the steps that need to be taken to preform an analysis are done in Julia using a single package which performs data aggregation, calibration of the model parameters, solving the model, and viewing the results of a simulation.

Finally, the code of my package is available on GitHub, and it is therefore transparent to researchers who may at all times contribute to its development. I hope that this level of public oversight can lead to preventing errors in the model and faster development, building a stronger technical foundation for GTAP-based policy analysis in the future.

References

- Beckman, Jayson, and Shawn Arita. 2016. “Modeling the Interplay Between Sanitary and Phytosanitary Measures and Tariff-Rate Quotas Under Partial Trade Liberalization.” *American Journal of Agricultural Economics* 99 (4): 1078–95. <https://doi.org/10.1093/ajae/aaw056>.
- Corong, Erwin, Hertel Thomas, McDougall Robert, Marinos Tsigas, and Dominique van der Mensbrugghe. 2017. “The Standard GTAP Model, Version 7.” *Journal of Global Economic Analysis* 2 (1): 1–119. <https://doi.org/10.21642/jgea.020101af>.
- Horridge, Mark, Michael Jerie, Dean Mustakinov, and Florian Schiffmann. 2019. “GEMPACK manual.” *Centre of Policy Studies/IMPACT Centre Working Papers Gpman, Victoria University, Centre of Policy Studies/IMPACT Centre*.
- Ivanic, Maros. 2023. “GEMPACK Simulations in r: Ademonstration of Running the GTAP Model and Processing Its Results Entirely in r Using Packages HARr and tabloToR.” *Journal of Global Economic Analysis* 8 (1): 1–20. <https://doi.org/10.21642/jgea.080101af>.
- Lubin, Miles, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. 2023. “JuMP 1.0: Recent Improvements to a Modeling Language for Mathematical Optimization.” *Mathematical Programming Computation*. <https://doi.org/10.1007/s12532-023-00239-3>.
- Mensbrugghe, Dominique van der. 2018. “The Standard GTAP Model in GAMS, Version 7.” *Journal of Global Economic Analysis* 3 (1): 1–83. <https://doi.org/10.21642/jgea.030101af>.
- Wächter, Andreas, and Lorenz T Biegler. 2006. “On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming.” *Mathematical Programming* 106: 25–57.