Journal of Global Economic Analysis, Volume 10 (2025), No. 1, pp. 175-217.

The GTAP v7 model in Julia

BY MAROS IVANIC [a]

In this work, I introduce a formulation of the GTAP version 7 model (Corong et al., 2017) in an open-source algebraic modeling language, JuMP (Lubin et al., 2023), implemented in Julia (Bezanson et al., 2017), that closely follows the specification of the model in GEMPACK (Horridge et al., 2019), including equation and variable names. Unlike the linearized GEMPACK version, my formulation is in levels. However, my formulation of the GTAP model is quite different from the levels formulation in GAMS (Bussieck and Meeraus, 2004) by Mensbrugghe (2018), in following more closely the variable and equation names of the GEMPACK model. I show that my model produces essentially the same results as the GEMPACK model. Because it is expressed in levels, with unabridged functional forms underpinning its behavioral equations (e.g., containing all parameters in the case of CES functions), my model can address a wider range of policy questions, especially those involving parameter changes. Calibrating the model to additional data, e.g., quantities, additionally allows it to be used in scenario analyses involving absolute productivity metrics. As an important benefit, my implementation of the GTAP model using open-source Ipopt solver (Wächter and Biegler, 2006), requires no software license to solve the model.

JEL codes: C63, C68

Keywords: GTAP; Julia; JuMP; Ipopt; CGE

1. Introduction

It is quite surprising that despite its wide use in computable policy analysis — documented by about forty-two thousand mentions on Google Scholar at the time of this writing — the Global Trade Analysis Project (GTAP) model has been remarkably difficult to obtain and run for the uninitiated members of the research community. If someone with skills most commonly encountered in scientific computing, e.g., R, Python, or Julia, had just learned of this great model and decided to try it, they would likely soon find themselves on an unfamiliar and fairly arduous journey taking them through several obstacles, that include finding the correct version of the model among many available on the GTAP website, purchasing and mastering an arcane math-solving system such as General Equilibrium Modelling PACKage (GEMPACK) or General Algebraic Modeling System (GAMS), and then

[a] United States Department of Agriculture, Economic Research Service, 1400 Independence Ave, SW, Washington DC, 20250 (e-mail: maros.ivanic@usda.gov).

obtaining the necessary data to allow the model to run in a non-standard format, such as HAR or GDX, either by purchasing the latest GTAP database or by downloading 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 a far more pronounced 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 software solutions commonly used in economic research are typically executed within open-source computing frameworks such as Python, R, and Julia, with their own unified ecosystem of supporting tools, e.g. to produce charts, plug with a web service, etc. The requirement to install commercial, stand-alone software, that often operates on a limited number of operating systems and may require administrator rights; with its own Integrated Development Environment (IDE) and custom data formats only to run a single model, is unexpected and presents a significant obstacle to anyone interested in exploring the GTAP model from scratch. [1] , [2]

A modern user of computing technology also expects that software is distributed centrally, either through a package repository (e.g., CRAN for R or Julia Repository for Julia) or through a version control system, such as GitHub. Package repositories are especially valuable because, they not only reduce the need to find and download the latest version of the software manually by searching for it, they also often assure that the software is compatible with the user's setup, including other installed packages. The software distributed through package repositories is also vetted so that it can be trusted by users and implemented in their workflows. Having to identify and download the correct version of the GTAP model and the database without any assurance that they would be compatible with each other—a very common occurrence in the fast-paced level of development of the GTAP data and the multitude of GTAP-based models—is a notable problem not only for a seasoned GTAP user but even more so for new users.

The ease of integration is another important feature of modern computing systems, which strive to support exchange of data among subsystems. Modern users of such systems therefore expect that any model as well as the supporting data

[1] For example, GEMPACK is primarily designed to work on Windows computers; while a Linux version can be arranged, the creators of GEMPACK urge users to consider Windows instead. GAMS is available for a number of operating systems, but historically GAMS creators have not been very quick at porting the software to other operating systems; for example, the Mac and Linux versions were only made available in 2005.

[2] Even though the software distributed by GEMPACK or GAMS are stand-alone executable files as it used to be common before the advent of Windows, such files may not be permitted to run under many security policies.

be available in common formats that can be generated or manipulated by other software, particularly the existing open-source solutions. In this regard, the availability of the standard GTAP model in a custom and commercial TABLO format, and the GTAP database in two custom and commercial formats, HAR and GDX, put additional burden on the potential user of the GTAP model to figure out the ways to integrate the model in their work, including making any changes to the underlying data and the model, and using the model output in any subsequent post-processing or publication.

Finally, open-source computing systems no longer come without integrated development environments (IDEs) or graphical user interfaces (GUIs), reflecting the great advances of general-use environments that can be easily adapted to handle different languages and outputs. Nowadays, users of computing systems expect to be able to use their favorite editor and development environment with different software solutions. Forcing a user to use dated GUIs or IDEs such as TABMATE or GAMS Studio with an extremely limited in functionality compared to modern IDEs, is another hurdle in front of a potential user of the GTAP model.

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 standard computing framework—specifically Julia, to allow modern researchers to work with the GTAP model in an expected manner. This means that the model, sample data, and the necessary tools are available as a package in the Julia repository. The package is also tested for several versions of Julia and the acceptable versions of all required packages are explicitly declared. In addition to the package code, my GitHub repository contains online documentation of the package, allowing any user to follow it to perform a Computable General Equilibrium (CGE) simulation using the model.

The GTAP model version 7 included in the package is translated from the GEMPACK version, enabling the users to reference the existing rich documentation of the model available on the GTAP Center's website and elsewhere. Naturally, some differences from the original formulation exist, mostly in the parts of the model where the vectorization feature of Julia made the model more succinct by combining analogous parts of the Constant Elasticity of Substitution (CES) function.[3] Other differences relate to the calculation of changes in the levels formulation, such as Equivalent Variation (EV) or real Gross Domestic Product (GDP) change, which require a comparison of two separate solutions.

I first demonstrate that this version of the model produces the same results as the model executed in GEMPACK (Horridge et al., 2019) for three scenarios in-

[3] For example, in the standard GTAP model activity, output is produced using a CES production function of two inputs, a combined intermediate input and a combined value-added input, expressed by two equations in GEMPACK. Because both inputs are determined by the same CES function, in the Julia/JuMP representation they are are expressed by a single two-element vector equation.

volving changes in tariffs, a technical change, and a change in output tax. Perhaps even more importantly, by executing example scenarios completely in Julia, I show that the GTAP model may be easily integrated in other open-source workflows: the model and the data may be read and modified within Julia and any outputs may be directly processed by the same system or saved in any standard format for further processing by other computing technologies.

In the rest of the paper, I describe the translated model and explain in detail how it can be accessed and executed. I then demonstrate 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. I also discuss the range of applications of the model in Julia for future research before I offer a few brief concluding remarks.

2. Performing a GTAP analysis entirely in Julia

In this section, I discuss in detail the steps necessary to complete a GTAP simulation entirely in Julia. This includes data aggregation, specifying the closure, specifying the shocks, running the model, and performing post-simulation analyses, including calculating welfare change and real GDP growth. To help current GEMPACK users understand the motivation for these steps and to note any important differences from the formulations by Corong et al. (2017) in GEMPACK, I first discuss the implementation in GEMPACK and then contrast it with the implementation in Julia. In addition to technical detail, I provide examples that may be followed by the reader interested in replicating those steps.

As a visual help, in Figure 1, I present a comparison of the workflows of performing a policy analysis using the GTAP model in both GEMPACK and Julia. Note that the workflow includes the optional steps of modifying the model, which is only needed if the user needs to make changes to the model. 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.

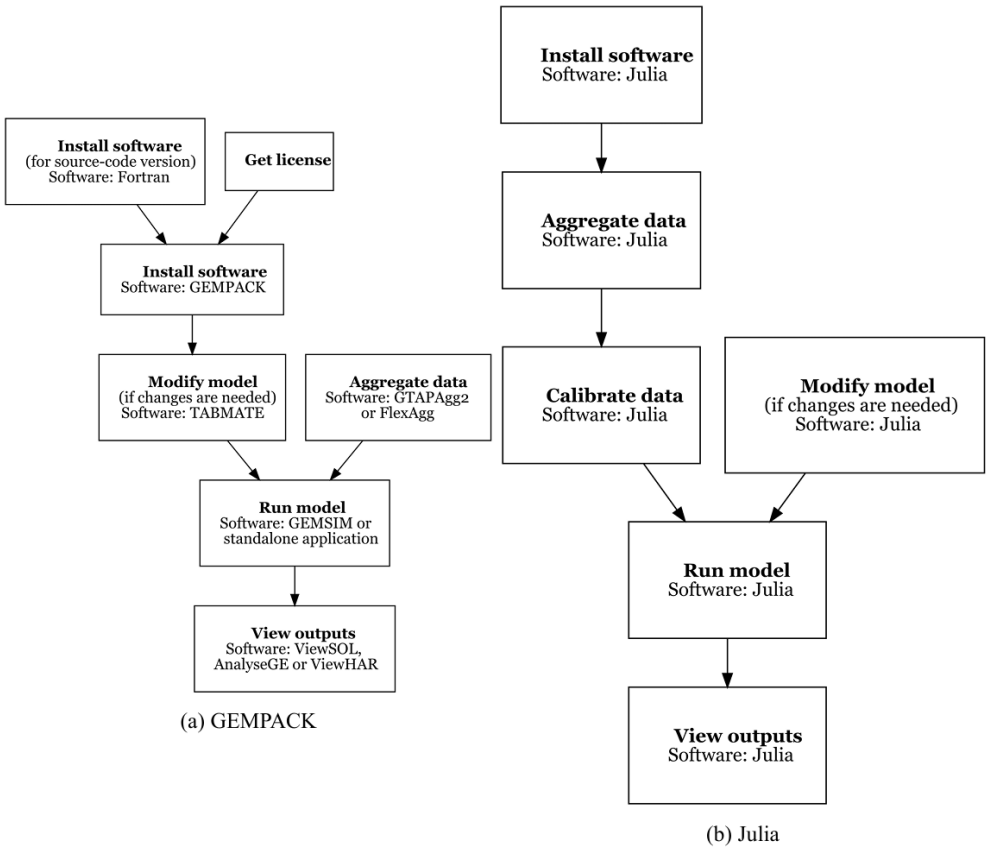

Figure 1. Comparison of workflows in GEMPACK versus Julia.

Source: Author calculations.

I would like to stress that all work done in Julia may be easily scripted and either executed as a whole or by line in Julia's interactive (REPL) implementation. The latter allows the user to work through the analysis and review the results at each step of the process. This is particularly useful in debugging the model or making changes to the model variables in real time. Furthermore, the possibility of performing an entire analysis in a script makes this approach particularly useful to support replicable research, one where each step of the analysis is transparent and open to review.

Because open-source software is typically developed on a continual basis, I would like to encourage interested readers to follow the development of the package on GitHub https://mivanic.github.io/GlobalTradeAnalysisProjectModelV7.jl. This repository contains up-to-date code along with the documentation of the package, including working examples that can be easily copied and executed. Also, this

is the place where users may easily report any bugs, propose fixes or ask for new features.

2.1 Required software

To execute a GTAP analysis in source-code GEMPACK, one first needs to obtain and install Fortran which is the low-level language in which GEMPACK models are executed. The user may use free or commercial versions of Fortran. With Fortran installed, the researcher would proceed to install and build GEMPACK, a suite of tools that allow the creation and execution of models, viewing and editing of the data, and a viewer for the model results. Alternatively, one can use a executable-image version of GEMPACK, which does not require a Fortran compiler. A researcher may also use a GUI-based stand-alone application RunGTAP, which facilitates an execution of the GTAP model simulation with the help of walk-through tabs, subject to some limitations. The user is required to obtain a license for GEMPACK in order to execute any but the smallest models.

The GEMPACK suite of software represents stand-alone applications which are invoked directly by the user in the operating system. While some of the tools, e.g., VieHAR, ViewSOL, AnalyseGE, and TABMATE, contain a graphical user interface, others such as executable models or GemSIM are intended to run interactively with user's input or through command files (e.g., cmf files). Reflecting the time when GEMPACK was created over forty years ago, GEMPACK does not offer a single scripting language; instead, the user is expected to utilize the operating system's fairly limited scripting ability, e.g., using DOS/Windows batch files, allowing for only simple execution flows supported with rudimentary operation system variables.

2.1.1 Installation of Julia and the required packages

Unlike the GEMPACK version of the model, the entire process of completing a simulation 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 package JuMP (Lubin et al., 2023), which allows for the models to be written out succinctly and solved by a solver, and Ipopt (Wächter and Biegler, 2006) a powerful open-source solver, on their computer.[4]

A user may install Julia free of charge from various locations, including Julia's website or Microsoft store. To install JuMP and the Ipopt solver, the user may install these packages (JuMP and Ipopt) along with any required dependencies from the Julia registry. To work with HAR files directly in Julia, the user may optionally

[4] JuMPdoes not require the use of the Ipopt solver and users may utilize other solvers including GAMS or other commercial solvers as they may be more efficient for large models. For comparison of different algebraic modeling systems, I point the reader to Jusevičius, Oberdieck, and Paulavičius (2021).

install package `HeaderArrayFiles`. To install the version of the GTAP model presented in this paper, the user may install package `GlobalTradeAnalysisProjectModelV7` from the Julia repository.

Installation of the Julia package representing GTAP model version 7

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

2.2 Required data

After obtaining the software, the next step for any analysis using the GTAP model is to obtain the GTAP database in the required form, i.e., a dictionary of arrays for the data, parameters and sets. Historically, the GTAP database has been provided by the GTAP Center in the form of a series of HAR files using a Fortran format designed to work with data writing sequentially, e.g., magnetic tapes. Nowadays, the GTAP Center also offers the data in the GDX format to be used with GAMS.

Instead of containing a single global Social Accounting Matrix, the GTAP database contains derivative data specifically designed to support the model, for example, providing the required value flows for all markets included in the model, such as household consumption, trade values at Cost, Insurance, and Freight (CIF) valuation, savings, and many others.

The GTAP database is normally downloaded from the GTAP Center's website. The Center currently offers the last two versions of the database for purchase—versions 11 and 10 representing reference years 2017 and 2014, while older versions are available for free download from the archive available on the website.

It is also important to note that for GEMPACK use, the GTAP Center provides the data in two formats: one for the use with GTAPAgg2 aggregation software, and one to be used with its more modern version FlexAgg. However, the data contained in both versions are identical.

2.2.1 Preparing the data in Julia

To work with the GTAP data as published on the GTAP Center's website, the user needs to first download the data. A separate package `HeaderArrayFiles.jl` is available, which allows the HAR data to be read in the HAR format directly into Julia; however, the user is free to use other means of importing the data into Julia.

For those users who are not familiar with GTAP data, or would like to test the model quickly without obtaining the data from the GTAP Center's website, the function `get_sample_data()` included in the package provides a sample aggregation of the freely available version 9 of the GTAP database aggregated into seven regions, six sectors, and five factors.

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 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 may skip the aggregation section entirely.

2.3 Data aggregation

Data aggregation represents an important step in a typical GTAP model analysis. The reason why the GTAP data are normally aggregated is to allow for the model to run in a timely fashion within the limits of one's computer. To 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 GTAPAgg or FlexAgg, which require the user to specify the groupings of commodities, factors and regions using a GUI in GTAPAgg 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, which 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.[5]

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 of the aggregation script and the persistence of errors in it.[6] For example,

[5] The 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.

[6] To view the data in a HAR format—before HARr package became available that allowed for a native import of the data into R — one would need to download a separate data viewer, ViewHAR, and learn how to use it. The situation with GAMS GDX file format is essentially the same, except there is no package available to read the data in R without a

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.

2.3.1 Aggregating GTAP data in Julia

In my Julia package, I treat aggregation as a simple summation of data and weighted summation of parameter values. The user only provides the function `aggregate_data()`, mapping vectors for endowments, commodities, and regions in the form of named vectors. Unlike in FlexAgg, where mappings between regions, commodities, and factors are simple vectors, in Julia the aggregation input is an actual Julia code, allowing, for example, the mapping instruction to include actual element names, e.g., `regMap["usa"] = "United States"` to make the original region labeled `usa` map into a new region labeled '`United States`', thereby reducing any unintentional errors and allowing for the aggregation code to be easily recycled, including in future versions of the GTAP database.

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. I do not, however, endorse its use in work destined for publication. Finally, as a reminder, the aggregation code for my Julia package is published on GitHub and therefore available to all users to quickly point out errors and suggest fixes in my work.

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:

GAMS installation. Having to install additional software and possibly pay a license simply to view the data would be quite an investment for a proficient data user who expects to be able to view and process complex data in the industry-standard formats.

```
regMap[["aus","nzl","xoc"]] .= "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"
```

The use of enumerated elements, i.e., writing `regMap[["aus","nzl","xoc"]] .= "oceania"` instead of `regMap[1:3] .= "oceania"` in the aggregation script is a particularly error-proof way of aggregating, as it does not depend on the positions of the elements in the disaggregated database. This means that when new regions are added and the positions of other regions is changed, for example as happened most recently with the addition of Mali and Haiti in the GTAP database version 11, positional aggregations would become invalid. On the other hand, an aggregation based on enumerated elements would generally be unaffected as long as the new regions are added to the aggregation.

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)
```

2.3.2 Data cleaning

The GTAP database sometimes contains values which are extremely close to zero, which are an artifact of the way data are 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 or they may be omitted from the model. In my implementation of the model in Julia, I exclude the structural zeros from the equations. To reduce the size of the model, optionally, I recommend removing any value less than 1 US dollar (i.e., 1e-6 of the units used in the GTAP database, which is in millions of US dollars) because the lower bound on quantities is 1e-9 by default. Removing such small values will not create any noticeable imbalance in the GTAP data base as the level of balance is in the order of millions of US dollars. Of course, if the user of the model would like

to maintain smaller values in the data, the bounds on quantities may be lowered further.

Removing fictitious trade

```
#Remove all trade that is less than 1 US dollar
valid_trade = hData["vxsb"].>1e-6
for k in ["vcif","vmsb","vfob","vxsb"]
hData[k][.!valid_trade].=0
end
```

2.4 Model

The model formulation is where most of the differences between the GEMPACK and Julia implementation lie, even though both implement the same GTAP model version 7 (Corong et al., 2017). The model is a general equilibrium model that describes the behavior among many agents in the economy. The agents include the regional household, government, and private household spending, saving, investment, and factor owners. Most of the production behavior is theoretically modeled as a series of nested CES functions, including the distribution of the outputs of an activity and the use of value-added and intermediate inputs in the activity. Similarly, trade is modeled under the Armington assumption as a CES production function where imports are 'produced' using bilateral imports as imports. The CES function is also used to produce an intermediate input mix of imported and domestically produced inputs. A CES (Constant Elasticity of Transformation (CET)) function is used to transform sluggish factors into different activities. Finally, a Constant Difference of Elasticities (CDE) function is used to model the preferences of the consumers. [7]

The GTAP model has been historically run in GEMPACK, a software specifically formulated to solve efficiently large linear systems.[8] Because the percentage change formulation of the GTAP model's CES, CET, and CDE equations only involves linear components, it is extremely suitable for the numerical solution of perturbations from the initial state of the GTAP model.

The GEMPACK suite of software, written in Fortran, provides several key fea-

[7] The reason for the widespread use of CES/CET functions is that even though their formulations are fairly complex and involve as many parameters as commodities plus an elasticity parameter (i.e., one independent scaling parameter, n-1 independent distributional parameters, and an elasticity parameter), they collapse to very simple linear relationships of percentage changes in variables with a single elasticity parameter, a very useful feature in the early days of computing.

[8] 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 its commercial counterparts, it is probably best suited for educational work or analyses that do not require much regional and sectoral detail.

tures 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. Other supporting software included in the suite allow to view and plot the results.

Even though GEMPACK is only able to solve the linearized version of the GTAP model, it is important to note that GEMPACK is capable of approximating the solution to the underlying non-linear system. This is achieved in GEMPACK by utilizing multistep solution methods (e.g., Gragg or Midpoint) which break the solution interval into many smaller subintervals, update the values of coefficients after each subinterval, and extrapolate the likely solution from multiple parallel multistep solutions using the Richardson extrapolation technique (Richardson, 1911).

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.[9,10] To change the shares of trade flows substantially between any levels, it is necessary to modify the distribution parameters of the underlying CES function in levels. Second, due to its linear nature, all equations in GEMPACK must be in the form of parameter/coefficient × 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. Again, in a levels formulation of the model the underlying parameters are made available and can be changed by the modeler as a part of a simulation.

The GTAP model in GEMPACK is represented by its linearized form, but the

[9] Conversely, it is also not typically possible to bring any quantity variable in a CES function to zero unless the parameters of the function change—no matter how much relative price grows, a CES formulation requires that some, infinitesimal trade takes place.

[10] Modelers 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.

solution to a simulation is, with the exception of a linear Johansen solution, solved non-linearly. This means that the solved values in GEMPACK will not necessarily satisfy the equations in their strict mathematical sense. The users of GEMPACK know that and do not generally expect that the linearized equations hold for the solved values.[11] The key point about the linear formulation of the GTAP model in GEMPACK is that it represents a simplified marginal relationship among variables. For example, where GEMPACK may relate the change in bilateral trade to overall imports and prices as:

$$\hat{q}_{s,i,d} = \hat{Q}_{i,d} + \sigma_{i,d}\left(\hat{p}_{s,i,d} - \hat{P}_{i,d}\right) \tag{1}$$

where the percentage change in imports of i from source region s to destination region d would be related to the change in total imports, the price of total imports and the price of the bilateral flow. The levels formulation captures the actual CES demand relationship with additional parameters whose percentage changes may or may not be included in the equation:

$$q_{s,i,d} = \frac{Q_{i,d}}{\gamma_{i,d}}\left(\frac{\alpha_{s,i,d}\gamma_{i,d}c_{i,d}}{p_{s,i,d}}\right)^{\sigma_{i,d}} \tag{2}$$

with the parameter γ representing scaling factor, α representing the distributional parameters ($\sum_i \alpha_1 = 1$), σ representing the elasticity, and c representing unit costs.

2.4.1 The GTAP model in Julia

In my implementation of the GTAP model in Julia, I follow more closely the approach adopted by van der Mensbrugghe (2018) in creating a levels version of the model in GAMS; however, I represent all functional forms in their full form, e.g., the actual CES function. This is mainly to allow researchers to be able to view the actual parameters of the underlying theoretical structure. The implementation of the GTAP model in Julia that is presented here aims to maintain the connection to the underlying theoretical structure of the GTAP model as closely as possible:

$$qxs_{i,d} = ces(qms_{i,d}, pmds_{c,r}, \alpha\_qxs_{i,d}, esubm_{i,d}, \gamma\_qxs_{i,d}) \tag{3}$$

where $ces()$ is a vector-valued CES demand function that accepts as arguments the single value for total imports $qms_{i,d}$, a vector of import prices from each source $pmds_{c,r}$, a vector of distributional parameters $\alpha\_qxs_{i,d}$, elasticity of substitution

[11] 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 for small changes, other AMLs such as JuMP and GAMS impose the requirement that equation solutions be mathematically correct. In those systems, it is therefore necessary to write the underlying levels, for example as $A \times (1 + a) = B \times (1 + b) \times C \times (1 + c)$ which would yield the solution that GEMPACK produces, even if the representation is a bit more complex.

$esubm_{i,d}$, and a single scaling parameter of the underlying CES function $\gamma\_qxs_{i,d}$.[12] This function returns a vector of import demand quantities $qxs_{i,d}$.

To make the model in Julia as easy to comprehend by existing users as possible, I additionally 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, including the distributional parameters α_qxs and the scaling parameterγ_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.

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/JuMP:

```
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])
```

The initial model is generated 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 of 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 to solve efficiently. The model also contains fields `lower` and `upper`, which contain optional lower and upper bounds on variables; if a bound is included for a variable, that variable is listed in either field with a corresponding value for the bound.

[12] Because the CES function is not defined for all values of σ, the implementation of the function has three different functions to cover the case of (1) $\sigma = 0$ as a Leontief function with distributional parameters allow for different input intensities, (2) $\sigma = 1$ as a Cobb-Douglas function, and (3) CES function for all $\sigma \notin 0, 1$

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 v7.

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 that 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)
```

2.4.2 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 solved data with the target data
mc.data = data_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
```

After calibration is completed, I recommend rebuilding the model using function `rebuild_model!()` which drops all equations that are only used in the calibration model, which results in a smaller model to be solved in actual simulations.

2.5 Specifying the model closure

The GTAP model in GEMPACK is solved as a system of linear equations (with single or multiple steps, depending on the solution method), with exactly as many endogenous variables as there are equations. Following Walras' law (Walras, 1900), one market needs to be omitted, as general equilibrium assures that the omitted market will be in equilibrium as well. The omitted market equation then allows one price to be fixed as a numeraire. In the GEMPACK formulation, the global investment market serves as the omitted market, with the imbalance captured by an endogenous variable `walraslack` which serves as an important check on the validity of the general equilibrium conditions. The price that is most commonly fixed is that of global factors, `pfactwld`.

The GTAP model contains additional exogenous variables which allow the users to change policy variables (e.g., taxes) and some of the model parameters (e.g., technical change). The exogenous variables may be swapped with endogenous variables, for example, to find a tax requiring a target output change. As long as the number of equations equals the number of endogenous variables, the model may, in principle, be solved. The closure is specified within the command file (cmf), which lists the exogenous and endogenous variables, as well as any swaps of the variables.

2.5.1 Specifying the model closure in Julia

The GTAP model in Julia/JuMP is very similar to that formulated in GEMPACK, which also omits the investment market and by default sets one price to one. Unlike the linearized version in GEMPACK, the levels version of the model needs to exclude some of the variables from the model. For example, many activities do not use land or natural resource, making their quantities and prices undefined.

In the linearized version, treating undefined variables as zeros typically does not pose any issues; however, in the levels formulation a zero quantity does not allow for a definite price to be associated with the transaction. To avoid this problem, the levels model in Julia/JuMP contains Boolean indicators of missing markets which remove missing markets from the system and exclude the variables which are undefined, allowing the system of equations to be solved.

The approach of excluding entire equations and the associated variables, distinguishes my approach from that used by Mensbrugghe (2018) in implementing a one solution method for GTAP in GAMS which involves mixed complementarity, allowing for possible zero-quantity markets to be associated with positive prices.

Finally, the user may also specify a flexible closure in Julia by fixing or freeing any variable in the model by listing the variable in the dictionary `fixed`, which is a part of the model container, as an array of the same size as the actual variable; any element whose value is set to true will be treated as an exogenous variable; all other elements and all variables not specified in fixed dictionary are treated as endogenous.

It is important to note, that GTAP in Julia/ JuMP does not require the model to be square, i.e., to have the same number of equations as variables. It is important for the user to check that this condition holds before attempting to solve the model.

2.6 Running a GTAP simulation

The users of GEMPACK have several ways of running a GTAP simulation, either using a command prompt, e.g., executing the model which has been compiled into an executable file, or executing it in GemSim with an associated command cmf file, or a GUI, such a RunGTAP, which executes the GTAP model in the background using the command prompt as a script.

Some of the additional specifications that the user is responsible for, is the selection of the solution method, the solution accuracy, number of intervals, and steps. Because GEMPACK uses a linear version of the model to estimate the underlying non-linear model changes, there are different methods to perform this approximation.

2.6.1 Running GTAP simulations in Julia

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 variables that 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 an imbalanced 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.[13] 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 the starting values can 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 data in 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.

Unlike in the case of GEMPACK, the solution in Julia is exact solution to a nonlinear system and therefore there is less need to specify accuracy level or solution method since the solution method should be the same regardless of which method was used and accuracy near the solution point can typically be higher than is usually needed (e.g., an error less than 1e-6).

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:

```
# 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)
```

2.7 Viewing the results of a simulation

Upon a successful completion, the GTAP model in GEMPACK produces a set of outputs, including the solution file that lists each variable with its solved value (an SL4 file), updated data if specified in the model (e.g., update statements on value flows), calculated coefficients at the start of a simulation (an SLC file), in addition

[13] For 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.

to optional log files that may be useful for debugging purposes, e.g., an accuracy listing file.

The viewing or processing of the results requires the use of additional software included in the suite of programs distributed with the installation of GEMPACK. These tools include a viewer of HAR files, e.g., ViewHAR, to view the original and updated data as well as the coefficients file, and specialized viewers for the model solutions, e.g., ViewSol or AnalyseGE, which process the output files to show the solved variable values either in a simple tabular form or within the original model code.

Because some of the outputs of the GTAP model involve multidimensional matrices, for example, the changes in margin uses in bilateral trade is a four- dimensional matrix; viewing these results is not simple without the use of specialized tools mentioned above that allow the user to slice through arrays to display lower-dimension portions of the arrays or provide simple summations across dimensions, an extremely useful feature implemented in ViewHAR.[14]

2.7.1 Viewing the solution in Julia

In my implementation of the GTAP model in Julia, 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 offers an interactive environment (REPL), it is possible to execute commands interactively before or after the model is executed. Furthermore, the JuMP package offers convenient access to the components of the model,e.g., variables and constraints, which makes it easy 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. This allows the user of the model to make changes to the model variables, e.g., adjust the shocks in real time and easily recalculate the model, starting from the last solution with great ease.

Because Julia offers interactivity, 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. Using standard Julia commands, the user may easily and efficiently not only slice the solution arrays and view the equations with the solved values, but also produce statistics across an array's dimensions, such as summations, medians, or produce any visualizations in real time.

[14] In some cases, even ViewSOL may not be adequate, if the number of dimensions of the variable that the user wishes to view exceeds three.

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)
```

2.8 Solution management

GEMPACK is a suite of independent executable files that perform various tasks related to conducting a GTAP analysis, including data aggregation, model execution, solution viewing and extraction. There is no GEMPACK-specific scripting language to make these programs operational; instead, the system relies on batch files executed directly by the operating system, with limited scripting capabilities. Finally, because there is no run-time environment for GEMPACK applications to store their results, they rely on file exchange as a way of communicating parts of the analysis through the work stream. As a result, a GTAP analysis typically produces a number of solution and data files that need to be managed by the researcher.

2.8.1 Solution management in Julia

Unlike running the GTAP model in GEMPACK, Julia provides an environment in which the workflow may be scripted. Furthermore, as a run-time environment, Julia allows for objects to be saved and retrieved in memory, allowing for the different parts of the simulations to easily access each other's outputs without generating any files. Files are typically only produced when the researcher desires to save any of the outputs for future use. While conducting a GTAP analysis, the researcher would typically use a single model container, an object residing in Julia's environment, and only preserve in memory any data object of any simulation that are needed for further analysis or need to be reported. The researcher may easily saved the results of simulations with convenient object names to support any subsequent analysis and save any output as files as needed.

3. Some key differences in implementation of the GTAP model between GEMPACK and Julia

Even though the Julia version of the GTAP version 7 model implements closely the same model that the GTAP model in GEMPACK represents in the linear form, there are some differences in implementation, which stem from the differences between the frameworks (GEMPACK vs Julia) and the type of the model (linear vs levels), 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, changes to variables between simulations, e.g., welfare, real GDP, 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, even if they require side calculations.

Other differences relate to the inclusion of convenience variables. Because calculating additional variables in GEMPACK is not easy, and any new variable requires recompiling and re-running the model, the GEMPACK version of the model contains many summary variables and indices that are likely not used in all situations, but because adding them later is costly, they are produced with each run. Because Julia provides an interactive environment, such convenience variables are best calculated after the model run using the solved values and are therefore omitted from the model.

3.1 Welfare calculations

In the case of equivalent variation, 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. An EV calculation requires the use of two levels solutions — the state of the world before the shock, and the resulting state of the world after the shock. They are used to calculate the equivalent variation between the solutions as:

$$\mathrm{EV} = e\left(p_0, u_1\right) - e\left(p_0, u_0\right) \tag{4}$$

where $e\left(\right)$ is an expenditure function, p_0 represents initial prices, and u_0 represents initial utility, while u_1 represents the level of utility after the simulation has been concluded.

While EV calculations 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, post-simulation prices, and utility, using the formula above.

For the user's convenience, I include in the package the function `calculate_expenditure` which calculates the income needed to achieve the new level of utility with original prices. Essentially this function just solves 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:

```
needed_expenditure = calculate_expenditure(
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:

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

3.2 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` and the scenario state is represented by the `data`, I can use the 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
```

3.3 Convenience indices

There are several summary variables that are calculated in the standard GTAP model which I have not included in the GTAP model in Julia because they can be easily calculated after the model is solved (e.g., `pwu` measuring world price index for commodities, etc.). Understandably, in GEMPACK 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 to avoid the need of costly compilations and recalculations to produce such summary

statistics. In Julia, it is fairly trivial to calculate any desired summary statistics using the existing solution without the need for repeating any calculations or re-running the model, since all results are available in memory.

4. 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 the 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, twn, 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, egy, 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

Source: Author calculations.

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

Source: Author calculations.

Table 3. Commodity aggregation.

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

Source: Author calculations.

4.1 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:

```
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.

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
easia	4.846148	1.251953	-0.070315	1.522812	0.010741	0.017948
seasia	4.643624	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	4.113975	0.718111	-0.099015	1.350469	-0.154081	-0.082714
scafrica	4.802666	1.242593	-0.074465	1.514383	0.016312	0.016752
eafrica	4.758132	1.251944	-0.135192	1.550390	0.034350	0.022665
safrica	-95.868500	11.918674	0.061139	-96.283157	1.656899	1.419624

Source: Author calculations.

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

Source: Author calculations.

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	-0.000002	-0.000000	-0.000000	-0.000000	0.000001	0.000000
seasia	-0.000001	-0.000000	-0.000000	-0.000000	0.000000	0.000000
sasia	-0.000002	0.000000	-0.000000	-0.000000	0.000000	0.000000
namerica	-0.000003	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	-0.000001	-0.000000	-0.000000	-0.000000	-0.000000	-0.000000
oeurope	-0.000001	-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	0.000003	0.000004	-0.000000	0.000001	0.000001	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

Source: Author calculations.

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.

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	-2.879805	-2.419444	-0.062222	-0.778172	-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

Source: Author calculations.

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

Source: Author calculations.

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

Source: Author calculations.

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.[15]

Table 10. Equivalent variation under the tariff increase scenario.

	GEMPACK	Julia	Difference
oceania	11.2	12.0	-0.8
easia	-57.5	-54.9	-2.6
seasia	11.5	11.1	0.3
sasia	10.1	9.9	0.2
namerica	-4.5	-8.1	3.6
samerica	89.3	86.5	2.8
carib	33.9	33.8	0.1
eu	-2,221.6	-2,230.7	9.1
oeurope	-89.6	-88.5	-1.0
wasia	1.7	2.3	-0.7
mena	-999.7	-1,005.2	5.5
wafrica	56.1	54.7	1.4
scafrica	11.0	11.0	0.0
eafrica	44.5	44.2	0.3
safrica	-371.4	-365.9	-5.4

Source: Author calculations.

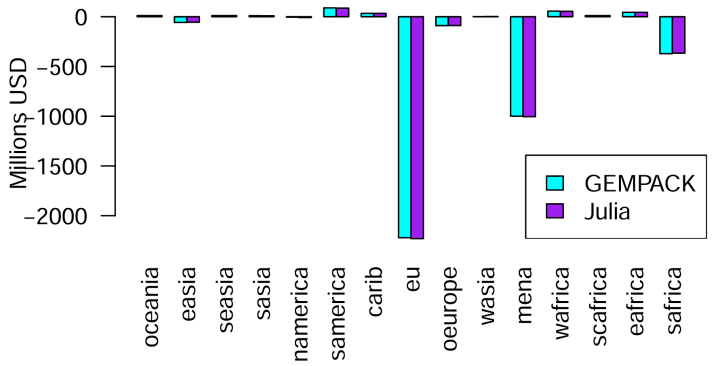

Figure 2. Equivalent variation of the tariff exercise.

Source: Author calculations.

[15] For 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.

4.2 Implementing a technological shift

In this scenario, I implement a technological shift by shifting value- added productivity by 100 percent for the service sector in Africa (regions `wafrica`, `scafrica`, `eafrica`, `safrica`):

```
target_regions=["wafrica", "scafrica", "eafrica","safrica"]
mc.data["γ_qintva"}][["svces"],target_regions]
.= mc.data["γ_qintva"][["svces"],target_regions] *2
run_model!(mc)
```

An equivalent piece of the shock definition in GEMPACK is:

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

A change in consumer prices related to the experiment are given in Table 11 for GEMPACK, Table 12 for Julia, and Table 13 shows the differences.

Table 11. Change in consumer prices `ppa` under the technical change scenario—calculated in GEMPACK.

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.369134	0.495397	-0.931946	-1.088207	-1.305854
easia	-0.610778	-0.743826	0.056401	-0.922293	-1.077219	-1.234111
seasia	0.612364	-0.049209	0.365497	-0.591506	-1.156791	-1.547579
sasia	-0.251821	-0.472595	0.028675	-0.866392	-1.037379	-1.462329
namerica	-0.153190	-0.736559	0.291343	-1.164022	-1.236870	-1.493578
samerica	-0.191706	-0.315001	0.438659	-0.699027	-0.905498	-0.960587
carib	-0.516725	-0.826658	0.269929	-1.324963	-1.477580	-1.896632
eu	0.341384	-0.278078	0.473798	-1.146042	-1.386451	-1.652398
oeurope	0.187301	-0.404349	0.490894	-1.171003	-1.318075	-1.737741
wasia	-0.337987	-0.490332	0.496373	-0.913664	-1.097374	-1.400723
mena	0.160158	-0.120767	0.528209	-0.858506	-1.138459	-1.351909
wafrica	26.450392	34.909550	3.911167	9.161584	2.256924	-43.544861
scafrica	50.023266	39.620800	1.821708	18.499454	1.156091	-36.783497
eafrica	30.449869	37.149734	14.983857	13.336094	-4.463110	-43.306206
safrica	11.567291	25.396729	4.714525	8.119807	-2.121907	-44.794113

Source: Author calculations.

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

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

Source: Author calculations.

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

	crops	animals	extract	food	manuf	svces
oceania	0.000000	-0.000000	0.000002	-0.000002	-0.000001	-0.000002
easia	-0.000001	-0.000001	0.000001	-0.000001	-0.000001	-0.000002
seasia	0.000001	-0.000000	0.000002	-0.000001	-0.000002	-0.000003
sasia	0.000000	-0.000000	0.000001	-0.000001	-0.000001	-0.000002
namerica	-0.000001	-0.000002	0.000001	-0.000004	-0.000003	-0.000005
samerica	-0.000000	-0.000000	0.000002	-0.000001	-0.000001	-0.000001
carib	-0.000000	-0.000001	0.000002	-0.000003	-0.000003	-0.000004
eu	0.000002	0.000002	0.000002	-0.000001	-0.000001	-0.000001
oeurope	0.000002	0.000001	0.000002	-0.000001	-0.000002	-0.000002
wasia	-0.000001	-0.000001	0.000002	-0.000002	-0.000002	-0.000003
mena	-0.000000	-0.000001	0.000002	-0.000002	-0.000002	-0.000003
wafrica	0.000050	0.000004	0.000012	-0.000134	-0.000057	-0.000098
scafrica	0.000003	-0.000000	0.000006	-0.000008	-0.000001	0.000005
eafrica	-0.000012	-0.000011	-0.000009	-0.000004	-0.000001	0.000007
safrica	-0.000002	-0.000002	-0.000000	0.000010	-0.000002	0.000006

Source: Author calculations.

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 differences with huge income values.

Finally, Figure 4 shows the comparison of the changes in the GDP quantity in-

dex. Again, the differences between the results obtained in GEMPACK and Julia are tiny.

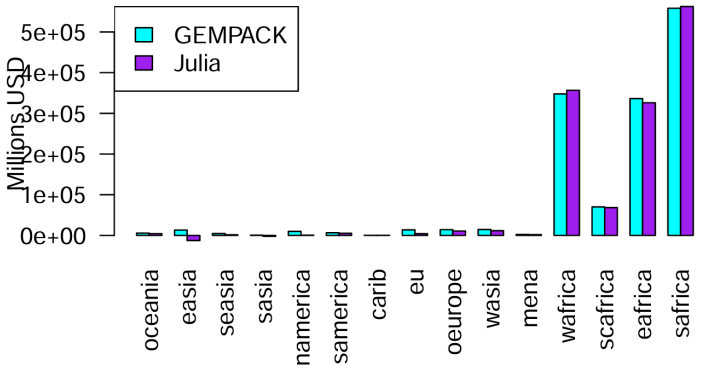

Figure 3. Equivalent variation of the technical change exercise.

Source: Author calculations.

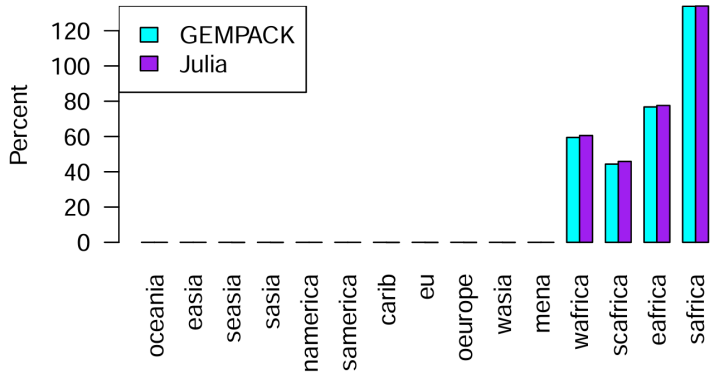

Figure 4. GDP quantity index change of the technical change exercise.[16]

Source: Author calculations.

4.3 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`).

[16] As mentioned in the text, the GDP index as reported in GEMPACK is the average of GDP growth at pre-and post-simulation prices

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
## 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.

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	0.023857	0.013873
safrica	-0.009242	-0.018318	0.228729	0.001678	0.039614	-0.001191

Source: Author calculations.

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

Source: Author calculations.

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

Source: Author calculations.

5. 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 GEMPACK, the modeler needs to write a completely new model with no persistent con-

nection to the one on which it is based.[17] 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 which may be useful in some cases.

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 the package JuMP.

The feature of JuMP models in Julia that they can be further extended opens up important 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 documenting 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, which is a common modification of the GTAP model:

$$qe_{land} = \alpha_{land} \times \left(\frac{pe_{land}}{ppriv} \right)^{\sigma_{land}} \tag{5}$$

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:

```
@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

[17] Naturally, 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.

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)
```

6. Run time comparison

To provide an idea to the reader about the run times of GTAP in Julia, I run a series of complete tariff removal scenarios between two to twelve regions and commodities.[18] For each aggregation size, I ran ten simulations with random aggregations, i.e., for a three-region and four-commodity scenario, I assigned all 160 regions randomly to three aggregated regions, and all 65 commodities randomly to four commodity aggregations. For each run, I recorded the time that it took to calibrate the model and the time that it took to solve the model. I report these times as a contour charts in Figure 5 for the calibration times, and in Figure 6 for the solution times.

As Figure 5 shows, a small 6x6 model may take about 60 seconds to calibrate. Increasing the size of the aggregation to 12 regions and 12 commodities is likely to increase the time to over one hour.

[18] Using a 32-core Intel Xeon Platinum 8710C CPU @ 2.8Ghz with 256 GB of memory.

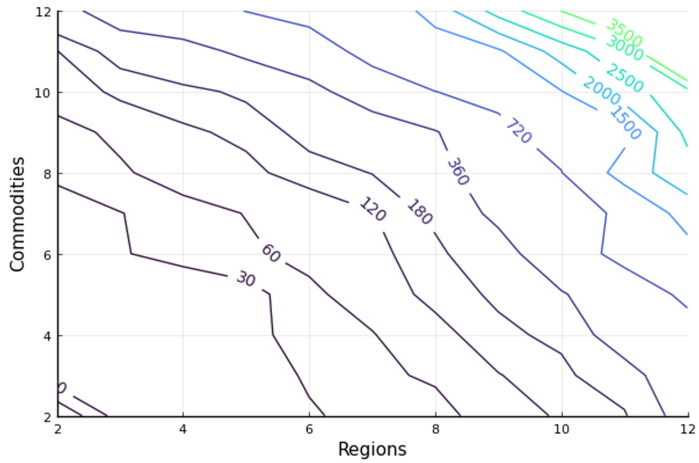

Figure 5. A contour chart of average run times (in seconds) for calibrations for various aggregation sizes.

Source: Author calculations.

Solving the model appears faster as shown in Figure 6, and takes about 30 seconds for a six-region and six-commodity model, and a bit less than an hour for a 12-region and 12-commodity model.

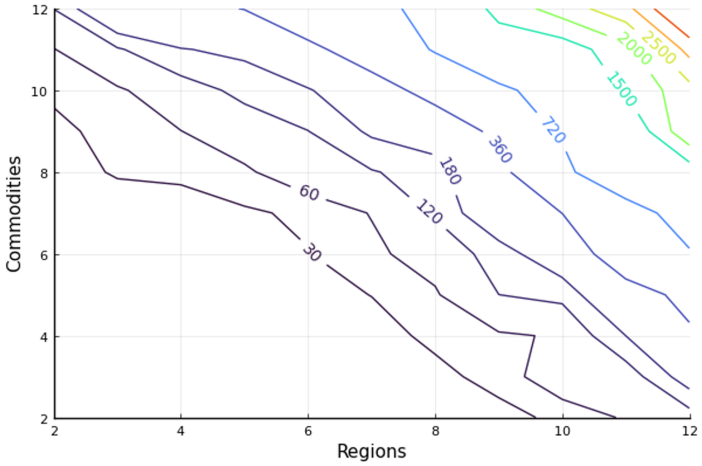

Figure 6. A contour chart of average run times (in seconds) for solving tariff removal for various aggregation sizes.

Source: Author calculations.

7. Discussion of the key benefits of a levels version of the GTAP model in Julia

7.1 Creating new flows

Zero initial flows represent a major problem in the GTAP model in the linearized form: a zero flow cannot become positive because no percentage change applied to it can lift it from the zero value. However, in the levels formulation, a missing flow is a result of a parameter value which may be changed, which allows for a missing flow to be created. For example, we may introduce a missing bilateral trade flow by changing the distributional parameter of the Armington preference function from zero to a positive value.

Naturally, a missing flow means that a market is missing as well—this is represented by the omission of the associated variables from the model, e.g., omitted prices when quantities are missing. To perform a simulation in which a new value flow is introduced, therefore requires that the corresponding parameter be changed from zero and a new model be rebuilt with the required variables included using function `rebuild_model!()`.

7.2 Simulations involving changes in parameters

An important policy question often involves the issue of changing parameters that define the specific supply and demand equations. For example, the parameters of the CDE demand system in the GTAP model could change when preferences of the consumers change, e.g., shift towards healthier diets, etc. Also, production functions may change, perhaps by utilizing input in different proportions or by allowing a greater or smaller degree of substitutability among existing inputs. Finally, trade preferences may change as well - countries may, over time, find imports from different regions more or less desirable.

Changes in parameters cannot be used as a simulation input directly in the GTAP model. As I mentioned earlier, a different parameter is still a part of the initial solution. However, in the cases where the changes in parameters are highly desirable, such as in the case of changed trade preferences and trade efficiency, the GEMPACK model has been modified to allow for that: variable `ams` has been designed to effectuate a combined change in the underlying CES function by increasing the scaling factor and adjusting the distributional parameters. However, changes to the substitution elasticity cannot be implemented in the standard model without considerable additional work.

An obvious benefit of the implementation of the GTAP model in Julia/JuMP is that all parameters are kept separate, allowing the user, for example, to change the elasticity of substitution among imports to simulate its impact on the model, without needing to change anything else.[19] Other applications of this approach

[19] While the levels formulation of the GTAP model in GAMS would normally permit the change of parameters, some particular aspects of its formulation make this type of sim-

may include a direct change in consumer preferences, e.g., to simulate a shift to plant-based diets, or changes to trade preferences without affecting overall trade efficiency.

7.3 Direct comparison of production functions

The standard GTAP model database contains only values for flows; there are no prices and quantities.[20] Naturally, calibrating a levels model to replicate those values produces a set of prices and quantities which give back the target values even though the prices and quantities are meaningless. [21]

An interesting application of the levels GTAP model present here arises in cases when the user is able to supplement the GTAP data with quantities or prices, e.g., by supplying land areas, quantities of agricultural output, or the number of workers in labor force, which would allow for the production functions to be calibrated in absolute terms, allowing for comparison of technologies and efficiency across sectors and regions. For example, by being able to compare the scaling and distributional parameters of the value-added CES function between two regions, the user could run a simulation targeting a particular level of productivity seen in one region in another regions, allowing for targeted technical change scenarios.

7.4 Better alignment of software development with the needs of computable modeling

Both GEMPACK and GAMS have enabled numerous users to employ the GTAP model in their work. However, both tools have two important drawbacks which are connected to each other: they are developed commercially which means that the users have to pay for a license and, at the same time, they 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, leading to technological outdatedness and clumsy integration with existing tools.

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 the source codes of models may be viewed without a license, a license requirement limits the spread of software and the number of researchers familiar with it. Even though trial licenses are sometimes available, they are typically very limiting.

---

ulation less clear. For example, in the case of bilateral imports, the GAMS implementation combines distributional parameters with the elasticity of substitution, requiring that a change in the substitution elasticity be also used to recalculate the combined distributional parameters.

[20] An exception to this is the land-use database which contains the quantities of land use in hectares and crop outputs in tons.

[21] Actually, there is an infinite number of prices and quantities that are consistent with the GTAP value database-setting the initial price to one is a typically a good choice.

While a review of a model can be done without a license - anyone who understands the syntax of GEMPACK, GAMS, or JuMP can review the model implementation - replication imposes a severe constraint on the ability of the researchers to replicate each other's work without incurring a significant financial cost and/or learning a new system. This, of course, has implications on the ability of researchers to publish their work, as replication is often a requirement by peer-reviewed journals.

Finally, it is important to mention the risks associated with commercial software development. If a commercial developer goes out of business, all development necessary ceases as the code stays private. Even worse, the user of the commercial software may not be able to renew their licenses and may remain without a way to run their existing code from one day to another. On the other hand, if the developers of open-source code such as Julia decide to do something else, the code remains available to the public and can be seamlessly developed further.[22] This presents an important guarantee to the users of the software that they will not be left without the ability to run their code in the future.

7.5 Easier integration in research and operational workflows

The formulation of the GTAP model in a modern language, Julia, allows the results to be easily integrated with other tools in Julia. This includes the ability of users to create interfaces to the model to facilitate their use among the users who are not coders, generate visualizations based on the outputs, and use the model in any computational workflows.

An example of such an integration can be demonstrated with the concept of sensitivity analysis, which involves understanding the distribution of the model results due to the uncertain distribution of the underlying parameters or shocks. To produce such a distribution, a researcher needs to run the model a number of times while changing the parameter and shock values in a way consistent with the observed distribution of those variables and parameters.

In GEMPACK, changing the inputs to a model requires changing the shock files and/or parameter files, which requires quite a bit of work without sufficient coding skills. To allow users to run a sensitivity scenario analysis, the creators of GEMPACK provide a specialized program which allows the researcher to run very simplistic scenario analysis for several pre-defined distributions.

Because GTAP in Julia can be directly manipulated within Julia, implementation of a sensitivity analysis is very simple — the researcher only needs to generate the new parameter and shock values - in Julia or elsewhere - and run the model in a loop over those values, without needing any special or additional coding requirements.

[22] For those readers interested in understanding Julia's governance as an open-source project, I recommend reviewing its governance page https://julialang.org/governance

8. 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.

Additionally, the model in Julia is expressed in levels which makes it suitable for analysis of additional types of policies that cannot be easily addressed by the linearized version of the model in GEMPACK such as those that involve creation of new value flows or changes in parameters.[23] The model may also easily be calibrated to additional data, such as quantities, making it useful for additional applications beyond the standard GTAP model.

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 GEMPACK formulation. This makes it easier for anyone who is familiar with the GEMPACK version to use my levels version in Julia. Additionally, the GTAP model presented here maintains all parameters underlying its demand and supply functions, creating a better connection with the theory that underpins the model. This makes the model particularly useful for the purpose of explaining the model to students of CGE modeling.

I demonstrate that performing GTAP analysis using my package allows the entire workflow—including data aggregation, calibration of the model parameters, solving the model, and viewing the results of a simulation - to be executed in Julia without the need of installing additional software. This provides for a far more straightforward analysis which can be easily scripted for the benefits of review and replication.

By enabling the GTAP model to be run in Julia I have reduced two important obstacles in using the model by researchers who are not familiar with GEMPACK or GAMS. 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 accessed by the researchers familiar with the most common computing frameworks, allowing them to integrate the GTAP model into their existing research workflows.

Finally, the code necessary to replicate the results of this work is provided as an attachment to this article. Also the entire Julia package is available on GitHub,

[23] For example, to change the distributional parameters and scaling parameters of trade (`ams`) in GEMPACK to simulate more efficient trade from a particular importer requires not only the reformulation of the trade demand equation, but also the unit cost equation and the calculation of welfare.

and it is therefore fully available to researchers who may at all times contribute to its development either by raising issues or by proposing modifications to the existing code through pull requests. 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.

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.

Acknowledgements

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

References

Beckman, J., and S. 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): 1078–1095. doi:10.1093/ajae/aaw056.

Bezanson, J., A. Edelman, S. Karpinski, and V.B. Shah. 2017. "Julia: A Fresh Approach to Numerical Computing." *SIAM Review*, 59(1): 65–98. doi:10.1137/141000671.

Bussieck, M.R., and A. Meeraus. 2004. "General Algebraic Modeling System (GAMS)." In *Modeling Languages in Mathematical Optimization*. Springer US, pp. 137–157. doi:10.1007/978-1-4613-0215-5_8.

Corong, E., T. Hertel, R. McDougall, M. Tsigas, and D. van der Mensbrugghe. 2017. "The Standard GTAP Model, Version 7." *Journal of Global Economic Analysis*, 2(1): 1–119. doi:10.21642/jgea.020101af.

Horridge, M., M. Jerie, D. Mustakinov, and F. Schiffmann. 2019. "GEMPACK manual." Centre of Policy Studies/IMPACT Centre, Victoria University, Report.

Ivanic, M. 2023. "GEMPACK simulations in R: A demonstration 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. doi:10.21642/jgea.080101af.

Jusevičius, V., R. Oberdieck, and R. Paulavičius. 2021. "Experimental Analysis of Algebraic Modelling Languages for Mathematical Optimization." *Informatica*, pp. 283–304. doi:10.15388/21-infor447.

Lubin, M., O. Dowson, J.D. Garcia, J. Huchette, B. Legat, and J.P. Vielma. 2023. "JuMP 1.0: Recent Improvements to a Modeling Language for

Mathematical Optimization." *Mathematical Programming Computation*, pp. . doi:10.1007/s12532-023-00239-3.

Richardson, L.F. 1911. "The Approximate Arithmetical Solution by Finite Differences of Physical Problems Involving Differential Equations, with an Application to the Stresses in a Masonry Dam." *Philosophical Transactions of the Royal Society A*, 210: 307–357.

van der Mensbrugghe, D. 2018. "The Standard GTAP Model in GAMS, Version 7." *Journal of Global Economic Analysis*, 3(1): 1–83. doi:10.21642/jgea.030101af.

Walras, L. 1900. *Éléments d'économie Politique Pure: Ou, Théorie de La Richesse Sociale*. F. Rouge.

Wächter, A., and L.T. Biegler. 2006. "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming." *Mathematical Programming*, 106: 25–57.