gegravity: General Equilibrium Gravity Modeling in Python

Peter R. Herman

ECONOMICS WORKING PAPER SERIES

Working Paper 2021–04–B

U.S. INTERNATIONAL TRADE COMMISSION
500 E Street SW
Washington, DC 20436
April 2021

gegravity: General Equilibrium Gravity Modeling in Python
Peter R. Herman
Office of Economics Working Paper 2021–04–B

Abstract

The `gegravity` Python package is a collection of tools for analyzing general equilibrium, structural gravity models of international trade. It provides a framework for estimating structural gravity models, simulating counterfactual trade experiments, and conducting Monte Carlo simulations to derive measures of statistical precision for model estimates. The package is based on prominent models used in the literature and aims to make structural gravity modeling more readily accessible to researchers and policy analysts.

Peter R. Herman
Research Division
Office of Economics
U.S. International Trade Commission
peter.herman@usitc.gov

Contents

1 Introduction **4**

2 Quick start **5**
2.1 Installation 5
2.2 Simple example 5

3 Theory and implementation **8**
3.1 Theory 8
3.2 Technical implementation 10
3.3 Package tools 11
3.4 Python dependencies 12

4 The main model: OneSectorGE **12**
4.1 Model inputs 12
4.1.1 Baseline data 12
4.1.2 Trade cost estimates 14
4.1.3 Elasticity of substitution 14
4.2 Counterfactual experiment 14
4.3 Helping the model solve 15
4.3.1 Rescaling the OMR terms 15
4.3.2 Checking inputs 16
4.3.3 Other solver options 16
4.4 Model results 17
4.5 Post-estimation analysis 17
4.5.1 Calculating counterfactual levels 18
4.5.2 Composition of trade 18
4.5.3 Weighted cost shocks 18

5 Monte Carlo analysis **19**
5.1 The MonteCarloGE model 19
5.2 Monte Carlo example 20

6 Conclusion **22**

1 Introduction

This paper describes the `gegravity` Python package, which is a set of tools used to estimate general equilibrium (GE) structural gravity models and simulate counterfactual experiments. The package is based on the well established version of the gravity model described by Yotov et al. (2016). It implements the structural GE gravity model in a general, robust, and easy to use manner in an effort to make GE gravity modeling more accessible to researchers and policy analysts. The package is publicly available and free to use although I ask that users please cite this paper.

The package provides several useful tools for structural gravity modeling. First, it computes theory consistent estimates of the structural multilateral resistance terms of Anderson and van Wincoop (2003) from standard econometric gravity results. Second, it simulates GE effects from counterfactual experiments such as new trade agreements or changes to other types of trade costs. The model can be flexibly used to study aggregate or sector level trade as well as many different types of trade costs and policies. The use of structural gravity models to conduct counterfactual policy experiments is a recent but growing trend. Researchers and policy institutes have used similar models to estimate the effects of a wide range of economic policies, including free trade agreements (Anderson and Yotov, 2016; Baier et al., 2019), Brexit (Brakman et al., 2018), China's Belt and Road Initiative (Kohl, 2019), language diversity (Gurevich et al., 2021), food safety requirements (Zongo and Larue, 2019), and maximum residue limits for pesticides (U.S. International Trade Commission, 2021).

Third, the package contains tools for conducting Monte Carlo simulations that provide a means to compute standard errors and other measures of statistical precision for the GE model results. The package's `MonteCarloGE` model generates a sample of trade cost parameters derived from an econometrically estimated gravity model and simulates a GE gravity model for each sample. This creates a sample distribution of GE results from which to produce various sample statistics. Ultimately, the Monte Carlo tools allow users to translate the inherent statistical imprecision in econometric gravity estimates of factors like distance or preferential trade agreements into corresponding measures of statistical accuracy in the GE results.

The `gegravity` package is one of many recent software tools for conducting gravity analysis. Most of these tools aid in the robust and fast econometric estimation of gravity models, especially those with high dimensional fixed effects. In Stata, these packages include `ppml` (Santos Silva and Tenreyro, 2015), `ppmlhdfe` (Correia et al., 2019), and `ppml_panel_sg` (Larch et al., 2019). In R, there is the similar `R_glmhdfe` package (Hinz et al., 2019). A few other packages have provided more comprehensive gravity modeling tools, such as the `gravity` package in R (Woelwer et al., 2020) and the `gme` package in Python (Herman et al., 2018). Unlike most of these tools, the `gegravity` package is not primarily an econometric package. Instead, it solves GE gravity models and conducts counterfactual experiments, often using econometrically estimated parameter values produced from these other tools.

To my knowledge, the only other GE gravity programming tools available are the .do files accompanying the methodological work of Yotov et al. (2016) and Anderson et al. (2018) and the `GE_gravity` Stata package of Zylkin (2019). The two methodological works describe GE gravity models and a "GEPPML" approach for implementing them (the model and implementation are described in more detail in section 3). The accompanying .do files provide example implementations and could be modified for use in other applications. The `GE_gravity` package is a more general implementation of a similar gravity model in Stata.

The `gegravity` Python package follows this work and offers several notable advantages. First, it is implemented in Python and therefore represents a free and flexible alternative to using Stata. Second, it is a general implementation that can be readily applied to different baseline data sets and counterfactual experiments with ease. Third, it is a parsimonious implementation, allowing users to conduct complex analyses with a limited number of commands. Fourth, it contains a a collection of other tools to help users understand, analyze, and extend their analysis. Similarly, it can be used in conjunction with the vast number of

Python libraries available.

The remainder of this paper is structured as follows. Section 2 provides a "quick start" guide to installing and using the main features of the package. Section 3 details the underlying structural model and the technical details of how it was implemented in the package. Section 3 describes the main model in the package—`OneSectorGE`—and its input requirements, use, outputs, and extensions. Section 5 describes the Monte Carlo gravity model. Finally, section 6 concludes.

In addition to the content in this paper, up-to-date technical documentation for the package as well as additional examples can be found at the packages's webpage, https://peter-herman.github.io/gegravity/.

2 Quick start

2.1 Installation

The `gegravity` package is available from the conventional Python package repository, pypi.org. It can be installed using the following pip command.

```
pip install gegravity
```

This will install the `gegravity` package as well as any necessary dependencies if they are not already installed.

2.2 Simple example

The following is a simple example of how to conduct a GE gravity analysis using the `gegravity` package. A copy of the code containing all commands in these examples can be found at https://gist.github.com/peter-herman/faeea8ec032c4c2c13bcbc9c400cca9b.

The first step in setting up a `gegravity` analysis is to load a few needed packages. The first is the popular `pandas` data manipulation package (McKinney, 2010), the second is the econometric gravity package `gme` (Herman et al., 2018), and the third is the `gegravity` package.

```
import pandas as pd
import gme as gme
import gegravity as ge
```

Next, load the data needed to both estimate trade costs using an econometric gravity model and parameterize the baseline GE gravity model. This example uses a simple cross-section dataset containing trade flows between 30 countries in 2006, output and expenditure values for each country, and a collection of typical trade cost variables such as distance and preferential trade agreements (PTAs).[1] A copy of this example dataset can be downloaded from https://gist.github.com/peter-herman/13b056e52105008c53faa482db67ed4a.

```
gravity_data_location = "C://sample_data_set.csv"
grav_data = pd.read_csv(gravity_data_location)
```

[1]The trade, output, and expenditure data was derived from the example data provided by Yotov et al. (2016) (https://vi.unctad.org/tpa/web/vol2/vol2home.html). The gravity variables were sourced from the Dynamic Gravity dataset of Gurevich and Herman (2018) (https://www.usitc.gov/data/gravity/dgd.htm).

The `gegravity` package relies heavily on the `gme` package, which contains a collection of gravity modeling tools.[2] These tools perform typical gravity estimation tasks and provide a gravity model structure that is used by the `gegravity` package. Using the `gme` package, we can structure the gravity data, define an econometric gravity model, estimate the model using Poisson Pseudo Maximum Likelihood (PPML), and store all relevant inputs and outputs in a single convenient Python object that is used by `gegravity`.

Begin by defining the estimation data structure using the loaded data. This requires the user to specify the columns in which certain variables can be found such as the importer, exporter, and year identifiers as well as the trade flows. After the data structure is defined, create and estimate an econometric gravity model. Doing so requires the user to specify the data structure to use; the column to use as the "left hand side" (LHS), dependent variable; the columns to use as "right hand side" (RHS), independent variables; and the type of fixed effects to add to the model (importer and exporter, in this example). Once defined, the model can be estimated and results printed to the console. Table 1 presents an abbreviated table of those results.[3]

```
# Define GME Estimation Data
gme_data = gme.EstimationData(grav_data,
                imp_var_name="importer",
                exp_var_name="exporter",
                year_var_name = "year",
                trade_var_name="trade")
# Create Gravity Model
gme_model = gme.EstimationModel(gme_data,
                        lhs_var="trade",
                        rhs_var=["pta", "contiguity", "common_language",
                            "lndist", "international"],
                        fixed_effects=[["exporter"], ["importer"]])
# Estimate gravity model with PPML
gme_model.estimate()
# Print results table
gme_model.results_dict["all"].summary()
```

With the gravity model econometrically estimated, which provides the basis for constructing bilateral trade costs, we can define the `gegravity` GE model. The GE model, `OneSectorGE`, utilizes the information that is already stored in the `EstimationModel`, which includes the estimating data, trade cost parameter estimates, and other information like column identifiers. Given that, defining the GE model requires only the specification of a few more inputs such as the year to use (the GE model is static and based on a single year), the columns containing output and expenditures, a reference importer to use (see section 3.2 for details), and an elasticity of substitution (`sigma`).

```
ge_model = ge.OneSectorGE(gme_model, year = "2006",
                          expend_var_name = "E",
                          output_var_name = "Y",
                          reference_importer = "DEU",
                          sigma = 5)
```

The next step is to build the baseline model. This step constructs some needed parameters from the input data and, most importantly, estimates the baseline outward (OMR) and inward (IMR) multilateral resistance terms. In building the baseline, an important model argument is the OMR rescaling factor, which can help the model solve numerically. This rescale factor is discussed more extensively in section 4.3.1. As the model solves, some basic

[2] For details, see https://www.usitc.gov/data/gravity/gme_docs/.

[3] A similarly formatted table of results can be created using the `gme.EstimationModel.format_regression_table()` method.

Table 1: Gravity econometric estimates

Variable	(1)
common_language	0.039
	(0.084)
contiguity	0.885***
	(0.131)
international	-3.422***
	(0.215)
ldist	-0.376***
	(0.072)
pta	0.482***
	(0.109)
AIC	3229449.511
BIC	3215497.239
Obs.	870

This table presents the econometric estimates from the example in section 2.2. Robust standard errors in parentheses. $^{***}p < 0.01$, $^{**}p < 0.05$, $^{*}p < 0.10$.

Table 2: Baseline multilateral resistance estimates

	OMR	IMR
AUS	3.58	1.42
AUT	3.41	1.22
BEL	2.93	1.05
BRA	3.59	1.29
CAN	3.31	1.34
CHE	3.18	1.14
CHN	3.12	0.97
DEU	2.92	1.00
DNK	3.54	1.28
ESP	3.21	1.22
FIN	3.69	1.32
FRA	3.11	1.13

This table presents the baseline multilateral resistance term estimate for a subset of countries.

feedback is printed to the screen indicating if there were any issues with computing the baseline model. After the model solves and the baseline is constructed, the baseline IMR and OMR terms can be retrieved from the model. A subset of these terms are presented in table 2.

```
ge_model.build_baseline(omr_rescale=10)
print(ge_model.baseline_mr)
```

With the baseline solved, the model can be used to conduct counterfactual policy experiments. An experiment modifies some of the trade cost measures (e.g. distance, contiguity, common language, pta, or international border) for certain countries and solves a counterfactual version of the model based on these alternative trade costs. In this example, we simulate the effects of a hypothetical Canada-Japan preferential trade agreement by setting the variable "pta" equal to 1 for Canadian exports to Japan and Japanese exports to Canada. This change is made in a copy of the baseline dataset, which is then supplied to the model as the "experiment dataset".[4]

```
# Get a copy of the data
exp_data = ge_model.baseline_data.copy()
# Modify the desired cost measures
exp_data.loc[(exp_data["importer"]=="CAN") &
             (exp_data["exporter"]=="JPN"), "pta"] = 1
exp_data.loc[(exp_data["importer"]=="JPN") &
             (exp_data["exporter"]=="CAN"), "pta"] = 1
# Supply the counterfactual data to the model
ge_model.define_experiment(exp_data)
```

With the experiment defined, the counterfactual model can be solved. As with the baseline, feedback will be printed to the console as the model solves and will indicate any issues with the solution.

```
ge_model.simulate()
```

[4]Making a "deep" `copy()` here is important as it insures that the baseline data is not modified too. A deep copy creates a new data object instead of just a reference to the original.

Finally, with the model solved, a wide variety of results can be returned from the model. At the country level, the model determines percentage change in factors such as total imports and exports, factory gate prices, real GDP, terms of trade, and the multilateral resistances.[5] It also produces counterfactual bilateral trade flows between each country pair and the estimated percentage change from the baseline. A more detailed description of the types of results that are produced can be found in section 4.4. Finally, users can output all the results and model diagnostics to a collection csv files for storage and analysis outside of Python.

```
country_results = ge_model.country_results
bilateral_results = ge_model.bilateral_trade_results
ge_model.export_results(directory="C://examples//",
                        name="CAN_JPN_PTA_experiment")
```

For illustration, a subset of the results contained in `ge_model.country_results` is presented in table 3. The results demonstrate the types of information that can be gleaned from GE gravity analysis. The largest effects of the hypothetical policy change are for Canada and Japan, as would be expected. Both countries experience lower prices, higher GDP, and greater trade. For example, under the counterfactual experiment, Canada's exports increase by 1.6 percent and real GDP increases by 0.4 percent. The rest of the world experiences much smaller impacts that depend primarily on their relationships with the Canada and Japan. For example, the United States and Mexico both experience relatively large changes, which can be explained by the close economic ties with both Japan and—in particular—Canada.

3 Theory and implementation

3.1 Theory

The `OneSectorGE` model in the `gegravity` package replicates the structural model of Yotov et al. (2016). That model is based on the earlier demand-side, constant elasticity of substitution (CES)-Armington structural gravity model of Anderson and van Wincoop (2003). As shown by Arkolakis et al. (2012), the structural gravity model can be derived from a wide range of different trade models such as the canonical supply-side, Ricardian version of Eaton and Kortum (2002). Thus, this particular version of the model can be considered reflective of a much more general class of trade models. For the sake of parsimony, I keep the theoretical discussion short as the `gegravity` package is merely an implementation of an existing model and makes no theoretical contributions of its own. For more details and discussion of the model, I refer the reader to Yotov et al. (2016) and Anderson et al. (2018).

The model system takes the following form:

$$X_{ij} = \frac{Y_i E_j}{Y} \left(\frac{\tau_{ij}}{\Pi_i P_j} \right)^{1-\sigma}, \tag{1}$$

$$\Pi_i^{1-\sigma} = \sum_j \left(\frac{\tau_{ij}}{P_j} \right)^{1-\sigma} \frac{E_j}{Y}, \tag{2}$$

$$P_j^{1-\sigma} = \sum_i \left(\frac{\tau_{ij}}{\Pi_i} \right)^{1-\sigma} \frac{Y_i}{Y}, \tag{3}$$

$$p_i = \left(\frac{Y_i}{Y} \right)^{\frac{1}{1-\sigma}} \frac{1}{\gamma_i \Pi_i}, \tag{4}$$

[5] Specifically, each change is calculated as 100×[counterfactual - baseline]/baseline.

Table 3: `gegravity` modeled effects of a Canada-Japan trade agreement

	Factory gate price	IMR	OMR	Real GDP	Foreign exports	Foreign imports
AUS	0.004	0.011	-0.004	-0.007	-0.088	-0.035
AUT	-0.003	0.002	0.003	-0.005	-0.035	-0.027
BEL	-0.002	0.000	0.002	-0.003	-0.038	-0.031
BRA	-0.006	0.000	0.006	-0.006	-0.080	-0.067
CAN	-0.134	-0.566	0.134	0.435	1.635	1.564
CHE	-0.003	0.001	0.003	-0.004	-0.035	-0.028
CHN	0.009	0.011	-0.009	-0.003	-0.036	-0.067
DEU	-0.002	0.000	0.002	-0.002	-0.036	-0.035
DNK	-0.004	0.003	0.004	-0.007	-0.040	-0.029
ESP	-0.002	0.002	0.002	-0.004	-0.053	-0.034
FIN	-0.004	0.004	0.004	-0.008	-0.045	-0.035
FRA	-0.002	0.001	0.002	-0.003	-0.040	-0.031
GBR	-0.002	0.002	0.002	-0.003	-0.058	-0.036
HKG	0.019	0.020	-0.019	-0.001	-0.071	0.022
IDN	0.005	0.013	-0.005	-0.008	-0.040	-0.026
IND	0.004	0.009	-0.004	-0.006	-0.050	-0.032
IRL	-0.006	0.001	0.006	-0.007	-0.040	-0.041
ITA	-0.002	0.001	0.002	-0.003	-0.045	-0.037
JPN	0.249	0.192	-0.248	0.056	1.289	2.405
KOR	0.013	0.018	-0.013	-0.004	-0.045	-0.052
MEX	-0.020	-0.007	0.020	-0.013	-0.103	-0.078
MYS	0.011	0.019	-0.011	-0.008	-0.026	-0.018
NLD	-0.002	0.001	0.002	-0.003	-0.039	-0.031
POL	-0.003	0.003	0.003	-0.006	-0.042	-0.030
SGP	0.011	0.013	-0.011	-0.003	-0.038	-0.036
SWE	-0.004	0.003	0.004	-0.007	-0.045	-0.038
THA	0.005	0.012	-0.005	-0.007	-0.036	-0.028
TUR	-0.001	0.005	0.001	-0.006	-0.052	-0.033
USA	-0.041	-0.034	0.041	-0.008	-0.395	-0.193
ZAF	-0.001	0.006	0.001	-0.007	-0.061	-0.040

This table presents the estimated results from the `OneSectorGE` example in section 2.2. All values reflect percentage changes in the counterfactual experiment relative to the baseline model.

$$E_i = \phi_i Y_i = \phi_i p_i Q_i. \tag{5}$$

Equation (1) is a typical gravity equation, which relates bilateral trade (X_{ij}) between exporter i and importer j to exporter output (Y_i), importer expenditures (E_j), global output (Y), bilateral trade costs (τ_{ij}), the elasticity of substitution (σ), and outward and inward multilateral resistances (Π_i and P_j, respectively). The multilateral resistance (MR) terms are defined by equations (2) and (3). These terms can be thought of as aggregate trade cost or price indices for the exporter and importer. Equation (4) defines factory gate prices (p_i), which are determined by output, OMRs, and the CES preference parameter (γ_i). Finally, equation (5) determines expenditures and provides a market clearing condition. Expenditures are determined as a fixed ratio (ϕ_i) of the value of domestic production. Domestic production is defined by the product of output quantity (Q_i) and factory gate prices. In this version of the model, output quantity is fixed/exogenous and all changes in the value of output are captured though the price term.

Throughout, the notation above is used to denote baseline variable values. Counterfactual versions of each variable are denoted with an $*$. For example, counterfactual trade costs are written as τ_{ij}^*.

3.2 Technical implementation

The `gegravity` packages solves the structural gravity model in six main steps, described below. Unlike the implementations described by Yotov et al. (2016) and Anderson et al. (2018), which solve the model via an custom iterative procedure that the authors refer to as "GEPPML", the `gegravity` package solves the model system using standard nonlinear solvers. Specifically, `gegravity` uses the `root` function in the Python `scipy` package (Virtanen et al., 2020).[6] Because of this difference in implementation and other numerical reasons like float precision, model results may not perfectly match between the `gegravity` package and other implementations but should be very similar.

The `gegravity` packages solves the structural model in the following steps.

1. **Initialize model**: The first step defines and initializes the model. This step constructs key parameters and the baseline trade costs from the user-supplied information. Baseline trade costs are computed as $\tau_{ij}^{1-\sigma} = \exp\left(\sum_k \beta_k Z_{ij}^k\right)$, where Z_{ij} is a collection of different trade cost proxies denoted by k—such as distance, trade agreements, and common languages—and β denotes a corresponding coefficient estimate from an econometric gravity model (see section 4.1.2). This step also checks to ensure that certain data is supplied and complete, such as trade flows, parameter values, expenditure values, and output values. This process is completed when the model is first defined via the `OneSectorGE` class.

2. **Solve for baseline MRs**: This step solves for the baseline IMR and OMR terms (P_j and Π_i, respectively). While standard econometric estimations of gravity models typically include controls for MRs in the form of importer(-year) and exporter(-year) fixed effects, these estimates are not equivalent to the important structural MRs. This step constructs the actual baseline structural terms using the baseline trade cost estimates supplied to the model, cost and expenditure data, and the elasticity of substitution. Specifically, it solves the system given by equations (2), and (3) for all Π_i and P_j. This step also calibrates the CES preference parameter (γ_i) based on the assumption that baseline factory gate prices are normalised to 1. This process is completed as part of the `OneSectorGE.build_baseline` method.

3. **Define counterfactual experiment**: This step establishes the counterfactual experiment. The user supplies a counterfactual dataset of cost information (Z_{ij}^*) and

[6] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.root.html

counterfactual trade costs are constructed using the new cost data in conjunction with the original cost parameter estimates ($\tau_{ij}^{*\,1-\sigma} = \exp\left[\sum_k \beta_k Z_{ij}^{*k}\right]$). This process is completed as part of the `OneSectorGE.define_experiment` method.

4. **Solve for "conditional" counterfactual MRs**: This step reconstructs "conditional" MRs for the counterfactual version of the model. "Conditional" refers to the fact that these terms are constructed while holding prices fixed at their baseline values, thereby producing partial equilibrium MRs. While these values can be accessed by users and are included with some of the results, their primary purpose is to provide initial values that aid in solving the full GE model. The model conditional MRs are derived by solving for the MR terms that satisfy equations (2) and (3) with τ_{ij}^* in place of τ_{ij}. This process is completed as part of the `OneSectorGE.simulate` method.

5. **Solve full counterfactual GE model**: This step solves for the full GE model and derives counterfactual IMRs (P_j^*), OMRs (Π_i^*), factory gate prices (p_i^*). Unlike in the previous "conditional" step, these terms are those that jointly solve the system described by equations (2), (3), (4), and (5). Again, the solution to this system system is based on the counterfactual trade costs, τ_{ij}^*. This process is completed as part of the `OneSectorGE.simulate` method.

6. **Construct other results**: Finally, with the model solved for the counterfactual MR terms and factory gate prices, other counterfactual elements are constructed. This step produces a variety of counterfactual outcomes such as bilateral trade flows (X_{ij}^*), outputs (Y_i^*), and expenditures (E_j^*). It also produces a collection of other results such as total counterfactual imports or exports, real GDP, and terms of trade. In most cases, the model also computes percentage changes in these results between the baseline and counterfactual experiment. This process is also completed as part of the `OneSectorGE.simulate` method.

An important caveat to note is that the model system is not uniquely determined on its own. The MR terms need to be pinned to a particular normalizing value because linear transformations of the terms can solve the system. To do so, a "reference importer" must be selected to act as the baseline, normalizing IMR term. The IMR for the reference importer is set equal to 1 in both the baseline and counterfactual experiment. As a result, all remaining IMR and OMR terms are determined in reference to this importer. The model is similarly unable to estimate a counterfactual IMR for the reference importer. Thus, the estimated change in that IMR is necessarily zero and all other counterfactual changes reflect this limitation. For this reason, users should carefully choose the country used as the reference importer. First, it is wise to select an importer with high quality data in order to mitigate the possibility that the remaining MR terms are all calibrated based on poor quality information. Second, because the model cannot estimate counterfactual changes in the reference countries IMR, it is best to choose a country for which the effects of the counterfactual experiment are likely small. Past literature has often used Germany as the reference importer (Yotov et al., 2016; Anderson et al., 2018).

3.3 Package tools

The model implementation is completed via several tools in the `gegravity` package, which are comprised of two main tools and some supporting tools. The two main tools are:

1. **OneSectorGE**: The `OnseSectorGE` class is the main tool of the package and implements the model described in section 3. The `OnseSectorGE` class contains the main methods for solving the baseline model as well as defining and solving the counterfactual model. It also contains several methods for helping diagnose and rectify convergence and other solver issues. The class is detailed in section 4

2. **MonteCarloGE**: The `MonteCarloGE` class is a tool for conducting Monte Carlo counterfactual simulations. It performs multiple simulations of the `OneSectorGE` model using different draws of trade costs. The alternative trade costs are based on the standard errors of the estimates for each of the individual trade cost proxies. The main pruprose of the class is to produce counterfactual results with measures of statistical precision such as standard errors or deviation. The `MonteCarloGE` class is detailed in section 5.

The remaining tools in the package are meant to support one or both of these primary classes. They are discussed as part of the `OneSectorGE` and `MonteCarloGE` descriptions in subsequent sections. Because the `MonteCarloGE` class is an extension of the `OneSectorGE` class, readers ought to initially focus their attention on the `OneSectorGE` class.

3.4 Python dependencies

The `gegravity` package depends heavily on several other Python packages. The `gme` package is used to handle and derive many of the inputs to the `gegravity` model (Herman et al., 2018). The `pandas` package is instrumental in managing the model data throughout (McKinney, 2010). The GE model is solved using the tools of the `scipy` package (Virtanen et al., 2020) and the mathematical tools of the `numpy` package (Harris et al., 2020).

4 The main model: OneSectorGE

The most important component of the `gegravity` package is the `OneSectorGE` class, which is the main GE gravity model. This section describes the features of this model, its input requirements, ways to address solver issues, and the many types of results it produces.

4.1 Model inputs

To perform a GE gravity analysis using the `OneSectorGE` model, users must supply several types of model inputs. These include baseline economic data, trade cost estimates, and other parameter values. Each are detailed in this section. Most of these data must be supplied to the model in the form of a defined `gme EstimationModel` object. However, there are some alternative ways to supply certain inputs, which are discussed below.

4.1.1 Baseline data

The `OneSectorGE` model has several important data requirements:

- **Identifiers**: Each observation in the model baseline data should be identified by three dimensions: an exporter, an importer, and a year.[7] These identifiers should be included in the baseline data input as three columns. All three columns should also be cast as string data for the best performance. Although year is a required identifier, it is not necessary that the data contain multiple years of observations. While additional years can improve the precision of any econometric estimates used in the model, `OneSectorGE` is strictly a static model and requires that a single year be selected for the baseline.

- **Bilateral trade flows**: The model requires data on bilateral trade flows (X_{ij}) between each country in the model. Importantly, the trade data must be *square* and include *intranational* trade flows. Square data refers to the idea that there must be a bilateral trade flow (positive or zero) for every combination of two countries.[8] The

[7] "Year" could represent an alternate time period such as a month, for example, depending on the desired scope of analysis.

[8] For example, a 30 country model should have exactly 900 bilateral trade flows.

data must also include intranational trade flows, which represent domestic shipments within a country (X_{ii}). While sources of intranational trade data are available (discussed below), one common approach is to create intranational flows from trade and production data by subtracting total exports from total production.[9]

- **Bilateral trade cost proxies**: The model constructs baseline and counterfactual bilateral trade costs based on supplied trade cost proxies (Z_{ij}). Throughout the literature, these typically include measures of geographic distance, contiguity, common languages, colonial ties, and preferential trade agreements. However, these cost proxies can be flexibly chosen based on the particular application in order to account for the factors that are most important to the analysis.[10]

- **Output and expenditure values**: The model requires total output and expenditure values for each country in the model. While there exist independent sources for these data, some past gravity research has also constructed these values from bilateral trade when that data is unavailable. In practice, expenditures should be equal to total imports plus intranational trade. Output should be equal to total exports plus intranational trade. However, in most cases, small differences will not prevent the model from solving.

These baseline data should be supplied to the `OneSectorGE` model via the `gme EstimationModel` class, which is a class that organizes and estimates econometric gravity models. To do so, all data should first be combined in a `pandas DataFrame`. Second, that `DataFrame` should be used to define a `gme EstimationData` object, which is a specialized data class for econometric gravity analysis. Third, the `EstimationData` object should be used to define a `gme EstimationModel` object. Finally, the `EstimationModel` can be used as the primary input to the `gegravity` model. Notably, even though the `gme` classes have no explicit support or role for output or expenditure data, these columns should be included in the baseline `DataFrame` used to create the `gme` objects. Additional information about the `gme EstimationData` and `EstimationModel` classes can be found at https://www.usitc.gov/data/gravity/gme_docs/.

The baseline data requirements can be met in a way that is flexible to the needs of the analysis. For example, the model can be used at the aggregate level, using total trade between countries, or at the sector or industry level. Like the theoretical gravity model, the `OneSectorGE` model is fully separable at the sector level. Similarly, the model does not require that a specific set of countries be used. Users should be cautious about omitting important countries but the model should generally be able to run using any set of countries assuming there is consistency in the values of trade, output, and production. That is, total trade, output, and expenditures world wide should be roughly equal and all value accounted for. However, the speed and solvability of the model may be affected by the selection of countries. Modifying the sample of countries can be a fruitful way of diagnosing a model that will not solve.

There are many prominent sources for the type of data used as inputs to the model. Two particularly good sources are the U.S. International Trade Commission's (USITC) Gravity portal and CEPII, which both provide a collection of publicly available, specialized gravity modeling databases that cover most of the basleine data requirements of the model.[11] The USITC's Gravity Portal hosts useful databases such as the International Trade and Production Database for Estimation (ITPD-E), the Dynamic Gravity Dataset (DGD), and the

[9] However, it should be noted this approach can result in some issues. For example, some products may be produced and shipped in different years and therefore appear in different annual reports. Similarly, production and trade reporting agencies may not be perfectly consistent.

[10] In principle, it is also be possible to use country pair (e.g. exporter-importer) fixed effects and their estimates as part of the trade cost component. Doing so would require that the fixed effect data and estimates be included the same way as other cost information rather than as fixed effect arguments in the `gme EstimationModel`.

[11] The USITC Gravity Portal can be found at gravity.usitc.gov. The CEPII databases can be found at http://www.cepii.fr/CEPII/en/bdd_modele/bdd_modele.asp.

Domestic and International Common Language Database (DICL). The ITPD-E database contains both international and intranational trade flows for a large number of countries for the years 2000–2016 (Borchert et al., 2020). The DGD database provides a large collection of trade cost proxies for most countries from 1948–2019 (Gurevich and Herman, 2018). The DICL database provides measures of language similarity, both internationally and intranationally (Gurevich et al., 2021). CEPII provides some similar resources including the TradeProd database of international and intranational trade from 1980–2006 (de Sousa et al., 2012) and the CEPII Gravity database of trade cost proxies (Head et al., 2010). In addition to these curated datasets, users can collect and build their own input data from other sources. For a more extensive discussion of gravity data sources, see Yotov et al. (2016).

4.1.2 Trade cost estimates

The `gegravity` package estimates bilateral trade costs based on typical econometric gravity models. For example, a standard specification might take the following form:

$$X_{ijt} = \exp\left\{Z_{ijt}\boldsymbol{\beta} + \mu_{it} + \nu_{jt}\right\} + \epsilon_{ijt}. \tag{6}$$

The model seeks to estimate trade cost coefficients ($\boldsymbol{\beta}$) associated with a collection of trade cost proxies (Z_{ijt}).[12] In the GE model, baseline trade costs are constructed using the estimates as $\tau_{ijt}^{1-\sigma} = \exp\{Z_{ijt}\hat{\boldsymbol{\beta}}\}$.

There are two ways to provide coefficient estimates to the GE model. First, they can be estimated using the `gme` `EstimationModel` that is defined as part of the model preparation. If the `EstimationModel` has been successfully estimated via the `estimate` method, then the `gegravity` model will use those coefficient estimates stored in the `EstimationModel` and no additional inputs are required.

It is also possible to supply alternative coefficient values, such as those derived from a different dataset or taken from the literature. To do so, they can be supplied at the point the `gegravity` model is first defined using the `parameter_values` argument. The external parameter values must be supplied in a particular format. The most reliable format to use is to use the gegravity `CostCoeffs` class, which can be defined using a `DataFrame` that contains columns of trade cost proxy names and coefficient estimates, respectively.[13] Alternatively, the model can accept a `gme` `SlimResults` or `statsmodels` `GLMResultsWrapper` object, which match the structure of estimates encapsulated in an estimated `EstimationModel`.

4.1.3 Elasticity of substitution

The model requires an elasticity of substitution, which determines the extent to which trade responds to changes in trade costs. Neither the `gegravity` nor `gme` packages estimate an elasticity value in general so it is expected that users supply this parameter from an external source. There are many different values to select from throughout the literature, derived using different methodologies and for a variety of sectors and product aggregations. The elasticity is supplied via the `sigma` argument of the `OneSectorGE` class when first defining it.

4.2 Counterfactual experiment

Counterfactual experiments in the `gegravity` model are based on changes to trade costs. These changes are introduced by supplying a modifying version of model's underlying trade cost proxies (Z_{ij}). The most reliable way to construct the counterfactual data is to begin

[12] Although not used by the GE model, a properly specified gravity equation should also include exporter (μ_{it}) and importer (ν_{jt}) fixed effects to control for the multilateral resistances in the econometric model.

[13] For more information, see the technical documentation at https://peter-herman.github.io/gegravity/.

by simply copying the GE model's baseline data from its data attribute: `OneSectorGE.baseline_data.copy()`. As mentioned before, it is important to include `.copy()` when creating a copy of the data to insure that the new version of the data is indeed a copy and not merely a reference to the base data. If the copied data is only a reference, changes made to the copied data will also be applied to the baseline data. The copied data can then be modified to represent the desired counterfactual set of cost proxies (Z_{ij}^{*}). For example, this may entail changing the values of a trade agreement variable for some country pairs or by raising or lowering the values of other measures of trade frictions. Once the counterfactual data is prepared, the experiment can be defined by supplying the counterfactual data to the model using the `OneSectorGE.define_experiment()` method.

4.3 Helping the model solve

The GE gravity model is a complex model that must solve large systems of equations on three different occasions. Because of this complexity, it is difficult to insure that all possible baselines and counterfactual experiments will solve. There are two tools built into the model to help troubleshoot and rectify cases in which the model fails to solve.

4.3.1 Rescaling the OMR terms

One common reason for solver failures is that the IMR and OMR terms, which are simultaneously solved for in each non-linear solver routine, can differ in magnitude substantially. When this is the case, the solver may face numeric challenges as a small numeric adjustment in terms with different magnitudes can have widely differing impacts on the function values, making a solution difficult to find. The differences between the magnitudes of the MR terms are unique to each model specification and depend on the underlying data. They can be especially prominent in models examining disaggregated industries, for example.

A solution to this MR magnitude issue is to rescale the OMR terms within the non-linear solver routines, which has no effect on the ultimate model outcomes but can fix the numeric issues. This can be done via the argument `omr_rescale` in the `OnesectorGE.build_baseline()` method. The value supplied should be of the form 10^x for positive or negative integer values of x (i.e., $\{..., 0.01, 0.1, 1, 10, ...\}$). The default value is 1.

To aid in finding an effective OMR rescale factor, the package includes the `OneSectorGE.check_omr_rescale()` method. This method, which can be used after the model is defined, attempts to solve the model using a range of potential rescale factors. By default, the model tries rescale factors ranging from 10^{-10} to 10^{10}, but this can be altered via the `omr_rescale_range` argument. The method returns a `DataFrame` reporting whether the model solved successfully for each of the tested rescale factors. It also provides several other types of diagnostic information such as the maximum function values from the solved system of equations (should be close to zero), and a sample of the produced OMR terms. These terms in particular are helpful for examining which rescale factors tend to produce stable OMRs. For example, many different rescale factors may numerically solve the model but only a subset produce consistent solution MRs. In these cases, the rescale factors that produce consistent OMRs are preferable.

To illustrate, table 4 presents a sample of the information produced by the `check_omr_rescale` method. These outcomes were produced by running the following code on the example in section 2.2 before `build_baseline` is performed.

```
rescale_eval = ge_model.check_omr_rescale(omr_rescale_range=3)
```

From the method's output, we see that a rescale factor of 0.001 fails to solve the model. Factors from 0.01 to 1 solve the model but produce unstable solution values. Factors 10 through 1000 both solve the model and produce consistent solutions, making them good candidates for the rescale factor.

Table 4: Example output from `OneSectorGE.check_omr_rescale()`

omr_rescale	solved	max_func_value	reference_importer_omr
0.001	False	0.088	2.340
0.01	True	3.683e-11	2.918
0.1	True	2.610e-09	2.921
1	True	7.410e-10	2.968
10	True	9.854e-10	2.918
100	True	3.629e-10	2.918
1000	True	3.3922e-09	2.918

This table presents example output from the routine checking OMR rescale factors. The column "omr_rescale" reports the rescale factor tested, "soved" indicates if the baseline model was solved with that rescale factor, "max_func_value" returns the largest function values and is indicative of how good the solution is at solving the system (smaller is better), and "reference_importer_omr" reports the reference importers OMR term in the solution.

It should also be noted that an OMR rescale value that solves the baseline model may not work in solving the full GE model, which must solve for both the MR terms and prices. In these cases, it may be possible to get the full model to solve by picking an alternative rescale factor.

4.3.2 Checking inputs

The model may also fail to solve due to incomplete or otherwise problematic data inputs. For example, missing values or columns that are of the wrong type can cause the model to fail to parameterize correctly and/or solve. Users should be careful that the data inputs are complete (e.g. data is square and all trade, output, expenditure, and trade cost columns are complete).

To aid in diagnosing data issues, the `OneSectorGE` model includes a method that tests the model inputs: `OneSectorGE.test_baseline_mr_function()`. This method tests to see if the model's system of MR equations can be computed from the supplied inputs (but does not find the system's solution). This can help narrow down whether a model failing to solve is because of problematic inputs or due to other numerical issues in the non-linear solver. If the inputs are the problem, then the `test_baseline_mr_function` will encounter issues and fail to compute any MR values from the system. The method returns a dictionary containing the various parameter values derived from trade costs, output, and expenditure values as well as the system values generated by those parameters and initial values for the endogenous MR terms. Ultimately, this information can be used to determine if needed parameter values are failing to be created, and therefore preventing the system from being solvable.

4.3.3 Other solver options

In addition to the above techniques, there are several other settings that can be adjusted to help the model solve. For each of the three non-linear solver routines, users can adjust the iteration limit, tolerance level, and solver method. For the baseline and conditional equilibrium solvers, these parameters are set by the following `build_baseline` arguments `mr_method`, `mr_max_iter`, and `mr_tolerance`. For the full GE solver, they are set by `simulate` arguments `ge_method`, `ge_tolerance`, and `ge_max_iter`. For more details on these settings, see the documentation for the `scipy` package and `root` function, which is used for the solvers (Virtanen et al., 2020).[14] The default values for both solvers are the 'hybr' method, 1400 iterations, and a tolerance of $1e-8$ for method, max iterations, and tolerance, respectively.

[14] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.root.html

4.4 Model results

The `OneSectorGE` model produces a wide variety of different results, including estimated baseline and counterfactual values for many of the variables in equations (1)–(5) as well as numerous economic indicators derived from those variables. It also reports percentage changes in these values between the baseline and counterfactual experiment, calculated as $100 \times (x^* - x)/x$ for each variable or index x. After the model has been fully solved and a counterfactual experiment simulated, these results can be found in several of the `OneSectorGE` class' attributes. The most prominent results attributes are discussed here and a full listing of all stored results can be found in the technical documentation at https://peter-herman.github.io/gegravity/.

It should be noted that in many cases, the trade outcomes distinguish between "modeled" trade and "observed" trade. "Modeled" trade represents the model constructed trade flows based on the parameterized gravity model. That is, they are the trade flows implied by the estimated MR terms and trade costs. The modeled flows may not perfectly replicate the actual baseline trade flows supplied as inputs to the model due to the fact that the model can not perfectly capture 100 percent of the variation in the data. By comparison, "observed" trade results are based on the actual observed values. In the baseline cases, they simply reflect the supplied baseline trade flows. In the experiment case, they reflect the observed baseline flows multiplied by the model-estimated percentage changes.

The main results attributes of the `OneSectorGE` class are the following.

- `OneSectorGE.country_results`: These are the main country level results for each country in the sample. They include consumer and factory gate prices, IMRs and OMRs, real GDP, welfare statistics, terms of trade, total foreign imports and exports, and intranational trade. All results are reported in percent changes from the baseline.
- `OneSectorGE.bilateral_trade_results`: These are the bilateral trade results produced by the model. They consist of the modeled baseline trade values, experiment trade values, and the percentage change.
- `OneSectorGE.aggregate_trade_results`: This attribute contains aggregate information about trade at the country level, both in modeled levels and percentage changes. Specifically, it reports total foreign exports, foreign imports, intranational trade, shipments (foreign exports + intranational trade), and consumption (foreign imports + intranational trade) for each country in the sample.
- `OneSectorGE.country_mr_terms`: These are the baseline, conditional, and experiment IMR and OMR terms for each country.
- `OneSectorGE.bilateral_costs`: These are the model constructed baseline and experiment trade costs (τ_{ij} and τ^*_{ij}).
- `OneSectorGE.outputs_expenditures`: These are the baseline and experiment output and expenditure values for each country.

Most results can also be exported to comma separated text files (csv) using the `OneSectorGE.export_results()` method. The method writes three files in the user-supplied directory containing all (i) country level results, (ii) bilateral results, and (iii) solver diagnostic information, respectively. Between the three files, almost all results from the model are stored.

4.5 Post-estimation analysis

In addition to the main results produced by the model, `OneSectorGE` has several tools that conduct different types of post-estimation analysis. These types of analysis include the projection of estimated changes in trade onto the observed baseline trade levels in order to

create counterfactual values of trade, calculations of changes in trade compositions between user-specified groups of countries, and calculations of the magnitude of the counterfactual shock at the bilateral or country level. Together, these tools help users better understand the underlying effects and consequences of the counterfactual experiment at a more granular level and uncover the reasons for the model's predictions.

4.5.1 Calculating counterfactual levels

As discussed in section 4.4, the `gegravity` package primarily reports certain values as modeled by the gravity model rather than as observed in the input data. However, users that are interested in how the estimated counterfactual effects would translate to observed values, there is the method `OneSectorGE.calculate_levels()`. The method computes the estimated change in observed values by taking the model estimated percentage changes and reapplying them to the observed baseline data.[15] The method can produce either country level or bilateral results by supplying either `"country"` or `"bilateral"` to the method's `how` argument. In either case, the method returns a `DataFrame` of calculated values. Counterfactual levels can also be calculated when exporting results by supplying the argument `include_levels=True` to the `export_results` method.

4.5.2 Composition of trade

Users may be interested in studying the changes in a country or countries' composition of trade. The method `OneSectorGE.trade_share()` can help easily provide this information. For example, building on the example from section 2.2, the command `ge_model.trade_share(exporters =["USA","MEX"],importers = ["CAN"])` will return figures for the share of Canada's imports coming from its NAFTA/USMCA partners in the baseline and the counterfactual scenario as well as the share of the U.S. and Mexico's exports going to Canada.

4.5.3 Weighted cost shocks

One challenge of GE modeling is that it can be difficult to identify where the major stimuli are in the model. A model with many countries and/or many changes in trade costs can have many compounding and offsetting forces, making it difficult to trace the underlying influences behind counterfactual results. The method `OneSectorGE.trade_weighted_shock()` can help shed some light on where the major effects of a counterfactual experiment are taking place.

In general, the magnitude of the estimated effects stemming from a change in trade costs depends on both the size of the change in costs and the size of the affected trade flows. The method computes an atheoretical weighted shock by first multiplying the value of bilateral trade between each country pair by the absolute difference between their baseline and experiment trade costs. Second, these weighted shocks are normalized by the largest shock so that the produced measures range from 0 (no change) to 1 (the largest weighted change).

The method can produce weighted shocks at either the bilateral or country level. If the method is called using the argument `how = "bilateral"`, it returns a `DataFrame` containing the weighted shocks for each bilateral pair. In general, the largest shocks are those with the greatest influence over the model's counterfactual estimates. If the method is called using the argument `how = "country"`, a `DataFrame` containing summary statistics based on these measures is produced. By default, these results produce measures of each country's mean,

[15] The model works a bit like a circle in this regard. Observed values are used to parameterize the model. The model then produces modeled baseline values based on the parameters. The counterfactual experiment simulates new experiment values, which are also modeled values. The modeled baseline and experiment values are used to calculate percent changes. Finally, the percent changes are taken back to the original observed data to create "observed counterfactual" values.

media, and max shocks are produced, both on the importer and exporter side. However, users can supply alternative methods of summarizing the measures using the `aggregations` argument.

To illustrate the information that can be gained from this tool, refer back to the example presented in section 2.2. An examination of the the bilateral weighted cost shocks would indicate that the largest weighted cost shock was on Canadian imports from Japan. The only other non-zero shock was on Canadian exports to Japan, which experienced the same policy change but the trade flow was smaller in value. This information helps us understand that the most influential factor underlying the counterfactual results is likely the impact on Canadian imports from Japan. Although somewhat trivial in this case, this type of breakdown can be very informative in more complex experiments where the policy change differs across parties or affects a greater number of countries.

5 Monte Carlo analysis

The second major tool in the `gegravity` package is the `MonteCarloGE` class, which is a tool for conducting Monte Carlo analyses with the GE gravity model. The purpose of this tool is to produce GE estimates that reflect the statistical uncertainty in the underlying econometric gravity estimates. In particular, the Monte Carlo model produces GE estimates with standard errors that reflect the standard errors of its econometric inputs. This section details the `MonteCarloGE` class and provides an example of its use.

5.1 The MonteCarloGE model

The `MonteCarloGE` class is an extension of the main `OneSectorGE` class that conducts a series of `OneSectorGE` trial simulations using a randomly drawn set of trade cost parameters. This process allows users to produce baseline and counterfactual estimates with standard errors that reflect the statistical uncertainty in the true underlying model parameters.

The `MonteCarloGE` model draws random vectors of cost parameters ($\{\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, ...\boldsymbol{\beta}^n\}$) for a user-specified n-many trials. These vectors of cost parameters are drawn from the multivariate normal distribution $\mathcal{N}(\hat{\boldsymbol{\beta}}, \mathcal{V})$ where $\hat{\boldsymbol{\beta}}$ is the vector of cost coefficient estimates from the underlying econometric gravity model and $\mathcal{V}$ denotes their corresponding variance-covariance matrix (Dobson, 2002, Sec. 5.4). A separate `OneSectorGE` model is solved (baseline and counterfactual experiment) for each trial.

The Monte Carlo model reports the results of the trials either through summary statistics derived from all trials or for each trial individually. By default, the Model will produce mean values for each type of model result produced by the `OneSectorGE` as well as their standard errors and standard deviations. The model also permits the inclusion of other types of summarizing functions such as median values or different percentiles.[16] The model can also return the complete set of results for every trial if requested.

Use of the `MonteCarloGE` is very similar to that of the `OneSectorGE` model described in section 4. The inputs, outputs, and underlying model structure are mostly the same. The main difference from a user's perspective is that the `MonteCarloGE` model conducts all three of the `OneSectorGE` simulation steps (`build_baseline`, `define_experiment`, and `simulate`) as part of the single `MonteCarloGE.run_trials` method. Because these steps are all consolidated with less ability to diagnose potential issues at each stage, it is best to first test a model specification using the `OneSectorGE` model in order to verify that it solves cleanly before attempting it as a `MonteCarloGE` model. The most notable change in input requirements is that the `MonteCarloGE` model requires that the trade cost parameters ($\hat{\boldsymbol{\beta}}$) be supplied via a `gme EstimationModel` that was estimated with the `full_results` argument equal to `True`. This insures that the `EstimationModel` contains a variance-covariance

[16] The model should be able to compute any function that can be supplied to the `pandas DataFrame.agg()` method.

model for use in the Monte Carlo parameter distribution. Because of these additional data requirements, `MonteCarloGE` does not have an option for supplying alternative cost coefficient values.

It should be noted that the Monte Carlo model may be more prone to solver issues than the `OneSectorGE` model. `MonteCarloGE` attempts to solve many similar but not identical models using the same inputs—including the `omr_rescale` factor—and it is possible that the solvers will not converge for all possible parameter draws. In many cases, alternative choices for the OMR rescale factor can rectify this issue but it is not guaranteed. If a particular trial fails to solve, the process will continue and the eventual results will simply reflect the trials that did solve.

5.2 Monte Carlo example

The following example demonstrates the use of the `MonteCarloGE` model. It builds on the example presented in section 2.2. A copy of the code for this this example can be found at https://gist.github.com/peter-herman/a2ebf3997bfd6e9cb3268298d49b64b5.

Beginning with the `gme EstimationData` object from the earlier example, we can define and estimate a slightly modified estimation model. Note the `full_results` argument in this example, which provides additional required information for the Monte Carlo model.

```
est_model = gme.EstimationModel(estimation_data=est_data,
                                lhs_var="trade",
                                rhs_var=cost_variables,
                                fixed_effects=[["importer"], ["exporter"]],
                                omit_fixed_effect=[["importer"]],
                                retain_modified_data=True,
                                full_results=True)
est_model.estimate()
```

With the econometric model estimated, we can create a `MonteCarloGE` model. In this example, we will have the model conduct 10 trials. In practice, users may want to consider using many more trials (law of large numbers) but should be aware that expanding the number of trials requires more memory and takes more time. We will also set the seed so that the results are reproducible. Other than those two new arguments, the other arguments should be familiar from the `OneSectorGE` model.

```
monte_model = MonteCarloGE(est_model,
                           year="2006",
                           trials=10,
                           reference_importer="DEU",
                           sigma=5,
                           expend_var_name="E",
                           output_var_name="Y",
                           cost_variables=cost_variables,
                           results_key="all",
                           seed=0)
```

When the model is defined, it draws the vectors of cost parameter values to use in each trial. We can examine those draws via the following two model attributes. The `coeff_sample` attribute returns the values for each individual trial while the `sample_stats` attribute contains summary statistics of that sample. As an illustration, the returned summary statistics are presented in table 5

```
full_sample = monte_model.coeff_sample
print(monte_model.sample_stats)
```

Table 5: Summary statistics for the Monte Carlo GE parameter draws

	pta	contiguity	common_language	lndist	international
beta_estimate	0.471	0.892	0.033	-0.390	-3.413
stderr_estimate	0.108	0.133	0.084	0.073	0.215
sample_count	10	10	10	10	10
sample_mean	0.438	0.938	0.061	-0.385	-3.397
sample_std	0.067	0.153	0.059	0.069	0.200
sample_min	0.349	0.591	-0.031	-0.538	-3.627
sample_25%	0.382	0.873	0.019	-0.418	-3.533
sample_50%	0.450	0.968	0.080	-0.370	-3.443
sample_75%	0.484	1.058	0.105	-0.331	-3.316
sample_max	0.542	1.089	0.131	-0.309	-2.996

This table depicts summary statistics for the model's ten random samples of cost coefficients. The first two rows, beta_estimate and stderr_estimate, depict the parameter estimates from the econometric model. The remaining rows summarize the ten Monte Carlo draws for each cost variable.

Next, we create the data to use for the counterfactual experiment, which follows the process in the earlier example. Again, we will use a Canada-Japan PTA as the experiment.

```
exp_data = monte_model.baseline_data.copy()
exp_data.loc[(exp_data["importer"] == "CAN") &
             (exp_data["exporter"] == "JPN"), "pta"] = 1
exp_data.loc[(exp_data["importer"] == "JPN") &
             (exp_data["exporter"] == "CAN"), "pta"] = 1
```

With the counterfactual data created, we can run the simulations. For the `MonteCarloGE` model, this is all done via a single method `run_trials`. The `run_trails` model features a couple of additional arguments that are not present in the `OneSectorGE` model. These new arguments determine what type of information and how much of it is returned as results. The `results_stats` argument specifies how to summarize the results across all the trials. In this example, we will produce mean values and standard errors (`sem`). The `all_results` argument determines whether the model retains the individual results for every trial after completion. By setting it equal to `False`, the model computes the summary statistics and disposes of the full results so as to avoid holding many potentially large data tables in memory.

```
monte_model.run_trials(experiment_data=exp_data,
                       omr_rescale=100,
                       result_stats = ["mean", "sem"],
                       all_results = False)
```

While running, the model should print the status and success or failure of each trial. Upon completion, we can examine the results. Begin by checking to see if any trials failed to solve. Next, we can access the many different type of results that the model produces. The results attributes are the same as those produced by the `OneSectorGE` model and discussed in section 4.4.

```
print(monte_model.num_failed_trials)
country_results = monte_model.country_results
bilat_results = monte_model.bilateral_trade_results
```

Table 6 presents the model results from this example. The results should look similar to those produced by the `OneSectorGE` model but now contain additional rows corresponding to each of the statistics requested. In this example, the rows reflect the mean estimated

percentage changes and their standard errors across all ten trials for each country and category of results. As expected, the mean estimates are similar to those produced by the `OneSectorGE` model, presented in table 3.

Had the model been run with `full_results = True`, those results could be accessed by adding the prefix "all_" to the standard results attributes (e.g. `monte_model.all_country_results`).

6 Conclusion

The `gegravity` package offers a convenient, freely-available, easy to use set of tools for conducting GE structural gravity modeling in Python. The model is based on well established, popular versions of the workhorse model in international trade. These tools should help researchers, policy analysts, and other interested parties rapidly conduct counterfactual analyses of a wide range of domestic and international policy changes.

Table 6: `MonteCarloGE` modeled effects of a hypothetical Canada-Japan preferential trade agreement.

	Factory gate price	IMR	OMR	Real GDP	Foreign exports	Foreign imports
AUS	0.003	0.010	-0.003	-0.007	-0.081	-0.031
	(0.000)	(0.001)	(0.000)	(0.000)	(0.005)	(0.002)
AUT	-0.003	0.002	0.003	-0.005	-0.032	-0.024
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.001)
BEL	-0.002	0.000	0.002	-0.002	-0.035	-0.028
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
BRA	-0.005	0.000	0.005	-0.005	-0.074	-0.061
	(0.000)	(0.000)	(0.000)	(0.000)	(0.005)	(0.004)
CAN	-0.123	-0.503	0.123	0.382	1.408	1.348
	(0.011)	(0.029)	(0.011)	(0.020)	(0.101)	(0.096)
CHE	-0.002	0.001	0.002	-0.003	-0.032	-0.026
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
CHN	0.008	0.011	-0.008	-0.002	-0.032	-0.060
	(0.001)	(0.001)	(0.001)	(0.000)	(0.002)	(0.004)
DEU	-0.002	0.000	0.002	-0.002	-0.033	-0.032
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
DNK	-0.004	0.003	0.004	-0.006	-0.037	-0.027
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
ESP	-0.002	0.002	0.002	-0.004	-0.049	-0.031
	(0.000)	(0.000)	(0.000)	(0.000)	(0.003)	(0.002)
FIN	-0.003	0.004	0.003	-0.007	-0.041	-0.032
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
FRA	-0.002	0.001	0.002	-0.003	-0.037	-0.028
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
GBR	-0.002	0.001	0.002	-0.003	-0.055	-0.033
	(0.000)	(0.000)	(0.000)	(0.000)	(0.003)	(0.002)
HKG	0.018	0.019	-0.018	-0.001	-0.064	0.020
	(0.002)	(0.002)	(0.002)	(0.000)	(0.004)	(0.002)
IDN	0.004	0.011	-0.004	-0.007	-0.037	-0.024
	(0.001)	(0.001)	(0.001)	(0.000)	(0.002)	(0.002)
IND	0.004	0.009	-0.004	-0.005	-0.045	-0.029
	(0.001)	(0.001)	(0.001)	(0.000)	(0.003)	(0.002)
IRL	-0.005	0.001	0.005	-0.006	-0.037	-0.038
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
ITA	-0.002	0.001	0.002	-0.003	-0.041	-0.034
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
JPN	0.226	0.177	-0.226	0.049	1.142	2.155
	(0.014)	(0.012)	(0.014)	(0.003)	(0.066)	(0.133)
KOR	0.013	0.017	-0.013	-0.004	-0.041	-0.048
	(0.001)	(0.002)	(0.001)	(0.000)	(0.003)	(0.003)
MEX	-0.019	-0.007	0.019	-0.011	-0.095	-0.072
	(0.001)	(0.001)	(0.001)	(0.001)	(0.006)	(0.005)
MYS	0.009	0.017	-0.009	-0.007	-0.025	-0.017
	(0.001)	(0.001)	(0.001)	(0.000)	(0.002)	(0.001)
NLD	-0.002	0.001	0.002	-0.003	-0.035	-0.028
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
POL	-0.002	0.003	0.002	-0.005	-0.039	-0.027
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
SGP	0.009	0.012	-0.009	-0.002	-0.035	-0.033
	(0.001)	(0.001)	(0.001)	(0.000)	(0.002)	(0.002)
SWE	-0.003	0.003	0.003	-0.006	-0.042	-0.035
	(0.000)	(0.000)	(0.000)	(0.000)	(0.002)	(0.002)
THA	0.005	0.011	-0.005	-0.006	-0.034	-0.026
	(0.001)	(0.001)	(0.001)	(0.000)	(0.002)	(0.002)
TUR	-0.000	0.005	0.000	-0.006	-0.049	-0.030
	(0.000)	(0.000)	(0.000)	(0.000)	(0.003)	(0.002)
USA	-0.038	-0.031	0.038	-0.007	-0.360	-0.176
	(0.002)	(0.002)	(0.002)	(0.000)	(0.022)	(0.010)
ZAF	-0.001	0.005	0.001	-0.006	-0.057	-0.037
	(0.000)	(0.000)	(0.000)	(0.000)	(0.003)	(0.002)

This table depicts results from the example `MonteCarloGE` model in section 5.2. For each country and indicator, the mean estimated percentage change in the counterfactual experiment across all the trials is presented with it's corresponding standard error in parentheses.

References

Anderson, J. E., M. Larch, and Y. V. Yotov (2018). GEPPML: General equilibrium analysis with ppml. *The World Economy 41*(10), 2750–2782.

Anderson, J. E. and E. van Wincoop (2003). Gravity with gravitas: A solution to the border problem. *American Economic Review 93*, 170–192.

Anderson, J. E. and Y. V. Yotov (2016, March). Terms of trade and global efficiency effects of free trade agreements, 1990–2002. *Journal of International Economics 99*, 279–298.

Arkolakis, C., A. Costinot, and A. Rodríguez-Clare (2012). New trade models, same old gains? *American Economic Review 102*(1), 94–130.

Baier, S. L., Y. V. Yotov, and T. Zylkin (2019). On the widely differing effects of free trade agreements: Lessons from twenty years of trade integration. *Journal of International Economics 116*, 206–226.

Borchert, I., M. Larch, S. Shikher, and Y. V. Yotov (2020). The international trade and production database for estimation (ITPD-E). *International Economics, forthcoming*.

Brakman, S., H. Garretsen, and T. Kohl (2018). Consequences of Brexit and options for a 'global Britain'. *Papers in Regional Science 97*(1), 55–72.

Correia, S., P. Guimarães, and T. Zylkin (2019). ppmlhdfe: Fast Poisson Estimation with High-Dimensional Fixed Effects.

de Sousa, J., T. Mayer, and S. Zignago (2012). Market access in global and regional trade. *Regional Science and Urban Economics 42*(6), 1037—52.

Dobson, A. J. (2002). *An Introduction to Generalized Linear Models* (2nd ed.). Chapman Hall/CRC.

Eaton, J. and S. Kortum (2002). Technology, geography, and trade. *Econometrica 70*(5), 1741–1779.

Gurevich, T. and P. R. Herman (2018, February). The dynamic gravity dataset: 1948-2016. Office of Economics Working Paper 2018-02-A, U.S. International Trade Commission.

Gurevich, T., P. R. Herman, F. Toubal, and Y. V. Yotov (2021). One nation, one language? domestic language diversity, trade and welfare. CESifo working paper no. 8860.

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant (2020). Array programming with NumPy. *Nature 585*, 357–362.

Head, K., T. Mayer, and J. Ries (2010). The erosion of colonial trade linkages after independence. *Journal of International Economics 81*(1), 1–14.

Herman, P. R., S. Ahmad, S. Shikher, T. Gurevich, G. Kenneally, S. Schreiber, C. Payan, and R. Ubee (2018). *gme: Gravity Modeling Environment*. Python package version 1.2.

Hinz, J., A. Hudletz, and J. Wanner (2019). Separating the wheat from the chaff: Fast estimation of GLMs with high-dimensional fixed effects.

Kohl, T. (2019). The Belt and Road Initiative's effect on supply-chain trade: Evidence from structural gravity equations. *Cambridge Journal of Regions, Economy and Society 12*(1), 77–104.

Larch, M., J. Wanner, Y. V. Yotov, and T. Zylkin (2019). Currency unions and trade: A ppml re-assessment with high-dimensional fixed effects. *Oxford Bulletin of Economics and Statistics 83*(3).

McKinney, W. (2010). Data structures for statistical computing in python. In S. van der Walt and J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference*, pp. 56–61.

Santos Silva, J. M. and S. Tenreyro (2015). `PPML`: Stata module to perform poisson pseudo-maximum likelihood estimation. *Statistical Software Components S458102*.

U.S. International Trade Commission (2021). *Global Economic Impact of Missing and Low Pesticide Maximum Residue Levels, Vol. 2*. Publication Number: 5160. Washington, DC: USITC.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods 17*, 261–272.

Woelwer, A.-L., J. P. Burgard, J. Kunst, M. Vargas, R. Francois, L. Henry, and H. Doytchinova (2020). *`gravity`: Estimation Methods for Gravity Models*. R package version 0.9.9.

Yotov, Y. V., R. Piermartini, J.-A. Monteiro, and M. Larch (2016). *An Advanced Guide to Trade Policy Analysis: The Structural Gravity Model (Online Revised Version)*. World Trade Organization and the United Nations Conference on Trade and Development.

Zongo, W. J. and B. Larue (2019). A counterfactual experiment about the eradication of cattle diseases on beef trade. *Canadian Journal of Agricultural Economics/Revue Canadienne d'agroéconomie 67*(4), 379–396.

Zylkin, T. (2019). GE_GRAVITY: Stata module to solve a simple general equilibrium one sector Armington-CES trade model. *Statistical Software Components* (S458678). Boston College Department of Economics.