The {teems} R package user manual

Matthew Cantele

Table of contents

1. Introduction

Reproducibility within computable general equilibrium (CGE) and partial equilibrium modelling analyses has long been elusive, in part due to proprietary data and software. Here we address one aspect of this equation with the introduction of the TEEMS (Trade and Environment Equilibrium Modelling System) open-source software suite.

At its core, TEEMS comprises an R package `teems` used to compose model runs and teems-solver, a C-based solver capable of solving a large system of nonlinear equations by leveraging a range of mathematical and scientific computation libraries. Auxiliary repositories to facilitate model runs include teems-models, containing internally available and immediately compatible models, and teems-mappings which contains ready-to-use mappings from coarse to fine resolutions for all core sets in most GTAP-based models.

With a base within the general purpose R scripting language, users able to seamlessly adjust model components including set aggregations, parameter weighting, closure and shock specifications as well as a large number of options pertaining to the solution and model outputs. Model scripts can then be shared with colleagues, reviewers, and policy-makers for vetting, modification, and reproduction.

1.1. Motivation

Reproducibility has been elusive also due to the shear complexity of compiling a single model run. A "simple" model run for example *broadly* involves:

1. Data and set selection
2. Set mappings
3. Model selection
4. Variable omission
5. Closure selection including any swaps
6. Shock designation
7. Set-specific basedata loading, modification, and aggregation
8. Set- and weight-specific parameter loading, modification, and aggregation
9. Selection of matrix solution methods
10. Number of step and subinterval designations
11. Constrained optimization solution
12. Composition of all outputs into structured data

Due to the complexity involved, in the course of building the TEEMS framework we found that in the absence of a reproducible architecture, there is a very high possibility of persistent logic errors propagating through to the output without any obvious indications that this is taking place.

Models which incorporate time steps and additional auxiliary data (e.g., land-use data) or are coupled with other models such as is the case with most integrated assessment models (IAMs) are even more complex and thus more susceptible to a wide range of errors.

Ultimately we hope that this software mitigates the steep learning curve requisite to running equilibrium models and allows scientists from various disciplines to advance model representation of their fields. The TLDR – running a CGE model *does not* have to be so hard!

1.2. Errors and model compatibility

This software suite has been under development for many years. Despite our best efforts, it is possible that you will encounter issues – particularly given the wide range of Tablo model files that exist in the field. If your model file does not work, please get in touch and we will seek to expand compatibility.

1.3. About this manual

This manual is a step-by-step user guide to `teems`.

Part I.

Getting started

2. Dependencies

This chapter outlines what is needed to run your first CGE model. Ideally all components would be open-source and maybe one day they will be, but for the moment there are still some minor hurdles to overcome. Before beginning to write model scripts in R with `teems`, users will need to gain access to a GTAP database and build the teems-solver. GEMPack users can skip the solver step although they will be missing out on a considerable degree of functionality both within the solution process itself as well as regarding post-model data availability.

For those skeptical of using R or under the impression that R is only for statistical analysis, we would invite you to investigate speed tests between the R data.table package (widely employed within `teems`) and comparable packages in Python and Julia. For all but the largest models, the time needed to prepare input files is trivial. The solver itself is written in C and uses highly performant Fortran libraries.

2.1. Data

Currently the only data for teems-compatible CGE model runs is available from the Global Trade Analysis Project (GTAP). Although the most recent databases are proprietary, GTAP has consistently open-accessed databases two versions out (GTAP 10 database as of the current GTAP 12 release). In order to access the freely available database, users will need to register with GTAP and download the "FlexAgg" format. For GTAP 10 this is accessible under the "GDyn Data Base" entry in the GTAP 10 "Satellite Data and Utilities" section here. For GTAP members that have purchased a database, the current `teems` version is capable of handling the "FlexAgg" format for GTAP Databases 9, 10, and 11. GTAP Data Base 12 will be added shortly.

2.2. R

R (>= 4.3.0) is required. Download and install the latest release for your platform from CRAN.

RStudio is the recommended IDE for working with `teems` scripts. It provides a script editor, integrated console, and a file browser that makes it straightforward to open, edit, and run the example scripts generated by `ems_example()`.

2.3. teems-R package

The R package is currently in beta (v0.0.4) and open for testing. It requires R >= 4.3.0 and is installed via remotes:

```
remotes::install_github("teemsphere/teems-R@v0.0.4")
```

2.4. teems-solver

`teems` calls Docker to solve the constrained optimization problem using low-level C and Fortran routines. The solver build is available here: teems-solver. The latest solver build will always be compatible with the latest R package version.

2.4.1. Prerequisites

Two prerequisites beyond Docker are required before building:

- **Docker** — install from docker.com. Linux users must also configure Docker to run without `sudo`.
- **HSL libraries** — four sparse linear algebra libraries (MA48, MA51, HSL_MC66, HSL_MP48) must be obtained directly from HSL prior to building. These are available at no cost for academic use.

2.4.2. Build options

Two Docker build approaches are available. The **expedited build** (~5 min) is recommended for most users and uses a pre-built base image containing all open-source dependencies. The **full build** (~40 min) compiles everything from source.

3. Quickstart

This chapter gets you from a fresh install to a solved CGE model in as few steps as possible, using the freely available GTAP 10 database and the classic GTAP model (version 6.2).

Before continuing, make sure you have completed all prerequisites in the Dependencies chapter:

- R (>= 4.3.0) installed
- `teems` R package installed
- GTAP 10 FlexAgg HAR files downloaded
- `teems-solver` Docker image built

3.1. Generate example scripts

`ems_example()` writes a set of ready-to-run R scripts to disk, with your data paths already injected at the top of each file. Point it at your HAR files and a project directory:

```
library(teems)

paths <- ems_example(
  model     = "GTAPv6",
  write_dir = "~/my_project",
  type      = "scripts",
  dat_input = "~/dat/GTAP/v10/flexAgg/gsddat.har",
  par_input = "~/dat/GTAP/v10/flexAgg/gsdpar.har",
  set_input = "~/dat/GTAP/v10/flexAgg/gsdset.har"
)
```

`paths` is a character vector of the written script file paths. Each script is self-contained and immediately runnable — no further editing is required to get a first run off the ground.

3.2. Step 1: verify the solver with a null run

Run `null.R` first. A null run applies no shocks, so every model variable should return zero. If it completes without error and all values are zero, the solver, data pipeline, and model file are all working correctly. Note that the examples below use `source()` to run a specific script but for interactive runs opening the scripts generated by `ems_example()` within RStudio is recommended.

```
source(paths[basename(paths) == "null.R"])

# null.R performs these checks automatically:
check        # TRUE  - all variable values are 0
var_check    # TRUE  - output contains the expected number of variables
coeff_check  # TRUE  - output contains the expected number of coefficients
```

If `check` is `TRUE`, the setup is confirmed and you are ready to apply shocks.

3.3. Step 2: run a uniform shock

`full_uniform.R` applies a 1% population shock (`pop`) uniformly across all regions. It is the simplest non-trivial run and a good sanity check before customizing anything.

```
source(paths[basename(paths) == "full_uniform.R"])

# full_uniform.R checks that the shock came through:
check  # TRUE - all pop values equal 1 (percentage change)
```

`outputs` is a data frame where each row corresponds to a solved variable or coefficient. The `dat` column holds the result data for that row.

```
# Inspect results for one variable
outputs$dat$pop
```

See the Compose chapter for a full guide to working with `outputs`.

3.4. Next steps

The other scripts in `paths` cover the full range of shock and swap configurations available for GTAPv6. Recommended progression:

Script	Try it when…
`part_uniform.R`	You want to shock a single region or sector
`part_uniform_part_swap.R`	You need to change the closure for part of the model
`custom_full.R`	You have heterogeneous shock values from external data
`full_uniform.R` with a different variable	You want to explore other exogenous variables

For a deeper look at each script pattern and how to customize them, see the Example files chapter. For data aggregation options (region and sector mappings), see Loading and specifying sets.

Part II.

Data and model inputs

4. Loading and specifying sets

4.1. Overview

Set mappings aggregate the fine-grained regions, sectors, and endowments in the GTAP database down to the resolution required for a given model run. In `ems_data()`, these are passed as named arguments via `...`, where each argument name corresponds to a set name in the model Tablo file.

Any set provided with a mapping is treated as a **parent mapping** and aggregated accordingly. Sets not explicitly mapped are handled automatically: if a set's elements are a subset of a mapped set's elements, the parent mapping is inherited; if a set's elements span multiple mapped sets, the relevant portion of each parent mapping is applied. If a set has no element-level relationship to any mapped set it remains unaggregated.

For the standard GTAP models the minimum required mappings are:

Format	Minimum mappings
GTAPv6.2	`REG`, `PROD_COMM`, `ENDW_COMM`
GTAPv7.0	`REG`, `ACTS`, `ENDW`

All other read-in sets (e.g. `MARG_COMM` in GTAPv6.2, `MARG` in GTAPv7.0) are derived automatically from the parent mapping. Additional mappings can always be supplied explicitly to override the inferred result.

4.2. Internal mappings

Internal mappings ship with `teems` and are specified by a string name without a `.csv` extension. All internal mappings, organized by database version, data format, and set name, are available at teems-mappings.

4.2.1. Region (`REG`)

Region mappings apply to the `REG` set in both GTAPv6.2 and GTAPv7.0 formats. The `AR5`, `WB7`, and `WB23` mappings are derived from the countrycode R package (ar5, region, and region23 respectively).

Name	Regions	Description
big3	3	China (chn), United States (usa), rest of world (row)
AR5	5	IPCC Fifth Assessment Report regions: asia, eit, lam, maf, oecd
WB7	7	World Bank 7-region classification
WB23	19	World Bank 23-region classification
R32	33	IIASA-based SSP 32-region mapping
medium	65	Custom medium-resolution aggregation
large	104	Custom large-resolution aggregation
huge	153	Near country-level with minor groupings
full	160	Full (unaggregated)

4.2.2. Sector (PROD_COMM, ACTS)

Sector mappings apply to PROD_COMM in GTAPv6.2 and ACTS in GTAPv7.0.

Name	Sectors	Description
macro_sector	5	Broadest aggregation: crops, food, livestock, mnfcs, svces
food	21	Food-focused: disaggregated food and agriculture; aggregated manufacturing and services
energy	23	Energy-focused: disaggregated energy sectors
manufacturing	31	Manufacturing-focused: disaggregated manufacturing sectors
services	23	Services-focused: disaggregated services sectors
medium	40	Custom medium-resolution aggregation
full	65	Full (unaggregated)

Margin and commodity subsets (MARG_COMM in GTAPv6.2, MARG in GTAPv7.0) are derived automatically from the parent sector mapping. If margin elements are aggregated into a broader sector by the chosen mapping, the margin set reflects that aggregation accordingly.

4.2.3. Endowment (ENDW_COMM, ENDW)

Endowment mappings apply to ENDW_COMM in GTAPv6.2 and ENDW in GTAPv7.0.

Name	Factors	Description
labor_agg	4	Aggregated labor: capital, labor, land, natlres

Name	Factors	Description
`labor_diff`	5	Differentiated labor: capital, land, natlres, sklab, unsklab
`full`	8	Full (unaggregated)

4.3. External mappings

Custom mappings can be provided as a file path to a two-column CSV or a data frame (e.g., `data.table` or `tibble`), where the first column contains origin elements and the second column contains destination (aggregated) elements. See teems-mappings for example files. Note that all set elements are transformed to lowercase within the solver outputs.

```
v7_data <- ems_data(dat_input = "path/to/gsdfdat.har",
                    par_input = "path/to/gsdfpar.har",
                    set_input = "path/to/gsdfset.har",
                    REG = "path/to/custom_REG_mapping.csv",  # external mapping
                    ACTS = "macro_sector",
                    ENDW = "labor_agg"
)
```

4.4. Set naming convention

Set names in ***any loaded data*** must be provided as the model-specific concatenation of the standard set name plus the variable-specific index. For example:

Set	Index	Column name
REG	r	`REGr`
REG	s	`REGs`
COMM	c	`COMMc`
ACTS	a	`ACTSa`
ENDW	e	`ENDWe`
ALLTIME	t	`ALLTIMEt`

This naming convention disambiguates data entries for variables that have multiple instances of the same set. Set position (i.e., column order) is inconsequential for any inputs.

5. Loading data

5.1. Overview

The `ems_data()` function loads and prepares GTAP database files for use in CGE model runs. It handles the three core GTAP data files (dat, par, and set) and can optionally convert between GTAP v6.2 and v7.0 formats. This function is also used to input any time steps (for temporally dynamic models), load set mappings for sets that are read into the model (i.e., not constructed from set operations), and incorporate auxiliary data via `ems_aux()`.

```
ems_data(dat_input, par_input, set_input, time_steps = NULL,
         aux_input = NULL, target_format = NULL, ...)
```

Argument	Type	Description
`dat_input`	Character	Path to a HAR file containing basedata coefficient data
`par_input`	Character	Path to a HAR file containing parameter coefficient data
`set_input`	Character	Path to a HAR file containing set elements and attributes
`time_steps`	Integer vector or `NULL`	Time steps for intertemporal models (see Time steps)
`aux_input`	List or `NULL`	Auxiliary data created with `ems_aux()` (see Auxiliary inputs)
`target_format`	Character or `NULL`	Target data format conversion: `"GTAPv6"` or `"GTAPv7"`
`...`	Named arguments	Set mappings for read-in sets (see Loading and specifying sets)

All three input files (`dat_input`, `par_input`, `set_input`) and model-specific set mappings are required. `time_steps` is required for intertemporal models.

5.2. Input files

Compatible input files are currently limited to those produced by the Global Trade Analysis Project (GTAP). Although the most recent database releases are proprietary, GTAP has consistently open-accessed databases two versions out (GTAP 10 database as of the current GTAP 12 release).

In order to access the freely available database, users will need to register with GTAP and download the "FlexAgg" format. For GTAP 10 this is accessible under the "GDyn Data Base" entry in the GTAP 10 "Satellite Data and Utilities" section here. For paying GTAP members, the current `teems` version is capable of handling the "FlexAgg" format for GTAP Databases 10 and 11.

The following command will load data consistent with the standard GTAP model, using broad aggregations for regions, commodities, and endowments. Users will need to specify the location of data (`"path/to/flexAgg11c17/gsdfdat.har"` placeholder).

```
v7_data <- ems_data(dat_input = "path/to/flexAgg11c17/gsdfdat.har", # basedata coefficients
                    par_input = "path/to/flexAgg11c17/gsdfpar.har", # parameter coefficients
                    set_input = "path/to/flexAgg11c17/gsdfset.har", # set elements
                    REG = "big3",
                    ACTS = "macro_sector",
                    ENDW = "labor_diff"
)
```

Note that the actual names of HAR data files vary according the GTAP release.

Input Type	GTAP v9	GTAP v10	GTAP v11
dat_input	gddat.har	gsddat.har	gsdfdat.har
par_input	gdpar.har	gsdpar.har	gsdfpar.har
set_input	gdset.har	gsdset.har	gsdfset.har

5.3. Set mappings

Set mappings aggregate the fine-grained regions, sectors, and endowments in the GTAP database down to the resolution required for a given model run. The set names used in `...` depend on the data format — see the Loading and specifying sets chapter for the full list of internal mappings and instructions for supplying custom CSV mappings.

5.4. Auxiliary inputs

With more complex models it is often necessary to load auxiliary data or directly modify existing database headers. The `ems_aux()` function prepares auxiliary inputs for injection via the `aux_input` argument. See the dedicated Auxiliary data chapter for full documentation, including input formats, `type = "set"` for new set creation, replacement vs. novel header behaviour, and error/warning reference.

5.5. Time steps

For temporally dynamic models (e.g., GTAP-INT, GTAP-RE), time steps must be provided representing t0 plus actual year steps from t0. Time steps can be inputted in either actual year increments

or represented as chronological years. Note that if chronological years are used, t0 must correspond with the reference year of the database being used.

Explicit time steps (equivalent to `c(2014, 2015, 2016, 2017, 2018, 2020, 2022, 2024, 2026, 2028, 2030)`) due to reference year associated with the loaded data inputs.

```
data <- ems_data(dat_input = "path/to/flexagg10AY14/gsddat.har",
                 par_input = "path/to/flexagg10AY14/gsdpar.har",
                 set_input = "path/to/flexagg10AY14/gsdset.har",
                 REG = "WB23",
                 PROD_COMM = "services",
                 ENDW_COMM = "labor_agg",
                 time_steps = c(0, 1, 2, 3, 4, 6, 8, 10, 12, 14, 16)
)
```

Chronological time steps (note reference year of input data)

```
data <- ems_data(dat_input = "path/to/flexAgg11c17/gsdfdat.har",
                 par_input = "path/to/flexAgg11c17/gsdfpar.har",
                 set_input = "path/to/flexAgg11c17/gsdfset.har",
                 REG = "R32",
                 ACTS = "medium",
                 ENDW = "labor_diff",
                 time_steps = c(2017, 2018, 2020, 2022, 2024, 2026, 2028, 2030)
)
```

5.6. Converting formats

GTAP databases are available in the classic v6.2 and standard v7.0 formats, corresponding to the classic and standard GTAP models. If you wish to use a classic-based model with newer GTAP databases or vice-versa, `target_format` will convert the underlying database. Note that set mappings are to be specified according to the model to be used, *not* the original format of inputted data.

GTAP 11 database converted to v6.2 data format:

```
v6_data <- ems_data(dat_input = "path/to/flexAgg11c17/gsdfdat.har",
                    par_input = "path/to/flexAgg11c17/gsdfpar.har",
                    set_input = "path/to/flexAgg11c17/gsdfset.har",
                    REG = "big3",
                    PROD_COMM = "macro_sector",
                    ENDW_COMM = "labor_agg",
                    target_format = "GTAPv6"
)
```

GTAP 10 v6.2 database converted to v7.0 data format:

```
v7_data <- ems_data(dat_input = "path/to/flexagg10AY14/gsddat.har",
                    par_input = "path/to/flexagg10AY14/gsdpar.har",
                    set_input = "path/to/flexagg10AY14/gsdset.har",
                    REG = "AR5",
                    ACTS = "macro_sector",
                    ENDW = "labor_agg",
                    target_format = "GTAPv7"
)
```

5.7. Data aggregation

Data is aggregated according to type, with non-parameter coefficients summed by target set mappings and weighted averages calculated for the parameters below. A simple mean is applied to parameters not listed below. If custom parameter values are desired, these can be loaded in their final format into the `ems_model()` function, and no aggregation or modification will take place. In the below notation, hat (ˆ) indicates the new parameter value and asterisk indicates the destination set mapping.

5.7.1. GTAP v6.2 format parameter weight methodology

Headers, descriptions, and index ranges for parameters and associated data within the GTAP v6.2 model. GTAP-INT weights are identical to GTAP v6.2 with the addition of an invariant time set.

	Header	Description	Set Index
Param.			
σ_{d}	ESBD	Armington CES for dom./imp. allocation	$i \in$ TRAD_COMM
σ_{m}	ESBM	Armington CES for regional allocation of imports	$i \in$ TRAD_COMM
σ_{va}	ESBV	CES between primary factors in production	$j \in$ PROD_COMM
γ	INCP	CDE expansion parameter	$i \in$ TRAD_COMM, $r \in$ REG
β	SUBP	CDE substitution parameter	$i \in$ TRAD_COMM, $r \in$ REG
Data			
	EVFA	Endowments – Firms purchases at agents' prices	$i \in$ ENDW_COMM, $j \in$ PROD_COMM, $r \in$ REG
	VDFA	Inter. – Firms' dom. purchases at agents' prices	$i \in$ TRAD_COMM, $j \in$ PROD_COMM, $r \in$ REG
	VDGA	Inter. – Government dom. purchases at agents'	$i \in$ TRAD_COMM, $r \in$ REG
	VDPA	Inter. – Household dom. purchases at agents'	$i \in$ TRAD_COMM, $r \in$ REG
	VIFA	Inter. – Firms' imp. at agents' prices	$i \in$ TRAD_COMM, $j \in$ PROD_COMM, $r \in$ REG

Header	Description	Set Index
VIGA	Inter. – Government imp. at agents' prices	$i \in$ TRAD_COMM, $r \in$ REG
VIPA	Inter. – Household imp. at agents' prices	$i \in$ TRAD_COMM, $r \in$ REG

5.7.1.1. Aggregation methods

$$\hat{\sigma}_{\mathrm{d}_i} = \frac{\sum_i^{i^*} \left(\sigma_{\mathrm{d}_i} \sum_r (vdpa_{i,r} + vdga_{i,r} + vdfa_{i,r} + vipa_{i,r} + viga_{i,r} + vifa_{i,r})\right)}{\sum_{i,r}^{i^*} (vdpa_{i,r} + vdga_{i,r} + vdfa_{i,r} + vipa_{i,r} + viga_{i,r} + vifa_{i,r})} \tag{5.1}$$

$$\hat{\sigma}_{\mathrm{m}_i} = \frac{\sum_i^{i^*} \left(\sigma_{\mathrm{m}_i} \sum_r (vipa_{i,r} + viga_{i,r} + vifa_{i,r})\right)}{\sum_{i,r}^{i^*} (vipa_{i,r} + viga_{i,r} + vifa_{i,r})} \tag{5.2}$$

$$\hat{\sigma}_{\mathrm{va}_j} = \begin{cases} \dfrac{\sum_j^{j^*} \left(\sigma_{\mathrm{va}_j} \sum_r evfa_{j,r}\right)}{\sum_{j,r}^{j^*} evfa_{j,r}} & \text{if } j \neq cgds \\ \sigma_{\mathrm{va}_j} & \text{if } j = cgds \end{cases} \tag{5.3}$$

$$\hat{\gamma}_{i,r} = \frac{\sum_{i,r}^{i^*r^*} \gamma_{i,r} \sum_{i,r} (vdpa_{i,r} + vipa_{i,r})}{\sum_{i,r}^{i^*r^*} (vdpa_{i,r} + vipa_{i,r})} \tag{5.4}$$

$$\hat{\beta}_{i,r} = \frac{\sum_{i,r}^{i^*r^*} \left(\beta_{i,r} \sum_{i,r} (vdpa_{i,r} + vipa_{i,r})\right)}{\sum_{i,r}^{i^*r^*} (vdpa_{i,r} + vipa_{i,r})} \tag{5.5}$$

5.7.2. GTAP v7.0 format parameter weight methodology

Headers, descriptions, and index ranges for parameters and associated data within the GTAP v7 model. GTAP-RE weights are identical to GTAP v7.0 with the addition of an invariant time set.

	Header	Description	Set Index
Param.			
σ_{d}	ESBD	Armington CES for domestic/imported allocation	$c \in$ COMM
σ_{m}	ESBM	Armington CES for regional allocation of imports	$c \in$ COMM
σ_{va}	ESBV	CES between primary factors in production	$a \in$ ACTS, $r \in$ REG
γ	INCP	CDE expansion parameter	$c \in$ COMM, $r \in$ REG
β	SUBP	CDE substitution parameter	$c \in$ COMM, $r \in$ REG
Data			

Header	Description	Set Index
EVFP	Primary factor purchases at purchasers' prices	$e \in$ ENDW, $a \in$ ACTS, $r \in$ REG
VDFP	Domestic purchases by firms at purchasers' prices	$c \in$ COMM, $a \in$ ACTS, $r \in$ REG
VDGP	Domestic purchases by government at purchasers' prices	$c \in$ COMM, $r \in$ REG
VDPP	Domestic purchases by households at purchasers' prices	$c \in$ COMM, $r \in$ REG
VMFP	Import purchases by firms at purchasers' prices	$c \in$ COMM, $a \in$ ACTS, $r \in$ REG
VMGP	Import purchases by government at purchasers' prices	$c \in$ COMM, $r \in$ REG
VMPP	Import purchases by households at purchasers' prices	$c \in$ COMM, $r \in$ REG

5.7.2.1. Aggregation methods

$$\hat{\sigma}_{\mathrm{d}_c} = \frac{\sum_c^{c^*} \left(\sigma_{\mathrm{d}_c} \sum_r (vdpp_{c,r} + vmpp_{c,r} + vdgp_{c,r} + vmgp_{c,r} + vdfp_{c,r} + vmfp_{c,r})\right)}{\sum_{c,r}^{c^*} (vdpp_{c,r} + vmpp_{c,r} + vdgp_{c,r} + vmgp_{c,r} + vdfp_{c,r} + vmfp_{c,r})} \tag{5.6}$$

$$\hat{\sigma}_{\mathrm{m}_c} = \frac{\sum_c^{c^*} \left(\sigma_{\mathrm{m}_c} \sum_r (vmpp_{c,r} + vmgp_{c,r} + vmfp_{c,r})\right)}{\sum_{c,r}^{c^*} (vmpp_{c,r} + vmgp_{c,r} + vmfp_{c,r})} \tag{5.7}$$

$$\hat{\sigma}_{\mathrm{va}_a} = \frac{\sum_a^{a^*} \left(\sigma_{\mathrm{va}_a} \sum_r evfp_{a,r}\right)}{\sum_{a,r}^{a^*} evfp_{a,r}} \tag{5.8}$$

$$\hat{\gamma}_{c,r} = \frac{\sum_{c,r}^{c^*r^*} \gamma_{c,r} \sum_{c,r} (vdpp_{c,r} + vmpp_{c,r})}{\sum_{c,r}^{c^*r^*} (vdpp_{c,r} + vmpp_{c,r})} \tag{5.9}$$

$$\hat{\beta}_{c,r} = \frac{\sum_{c,r}^{c^*r^*} \left(\beta_{c,r} \sum_{c,r} (vdpp_{c,r} + vmpp_{c,r})\right)}{\sum_{c,r}^{c^*r^*} (vdpp_{c,r} + vmpp_{c,r})} \tag{5.10}$$

6. Auxiliary data

6.1. Overview

`ems_aux()` prepares a single auxiliary input for injection into the `ems_data()` pipeline via the `aux_input` argument. Auxiliary data can replace an existing header in the loaded database or introduce a novel header not present in the original files. All auxiliary data is loaded at full (unaggregated) resolution; `ems_data()` subsequently aggregates it according to the active set mappings and the `type` argument.

```
ems_aux(input, type, header = NULL)
```

Argument	Type	Description
`input`	Data frame, character vector, or file path	The auxiliary data to load. Accepts a data frame or data frame extension (e.g., tibble, data.table) for `"dat"` and `"par"` types; a character vector for `"set"`; or a path to a CSV or GTAP HAR file
`type`	Character	Aggregation type: `"dat"` (sum aggregation), `"par"` (mean or weighted mean aggregation), or `"set"` (final set elements, no aggregation)
`header`	Character or `NULL`	Header name for the input. Required unless `input` is a GTAP HAR file, which already contains header metadata

6.2. Input formats

`ems_aux()` accepts three input formats determined by `input`.

6.2.1. Data frame

A data frame, tibble, or data.table containing a `Value` column and one or more set columns identifying the dimensions of the coefficient. The `header` argument is required.

```
# Load at full resolution, modify, then prepare as auxiliary input
.data_full <- ems_data(
  dat_input = "path/to/gsdfdat.har",
  par_input = "path/to/gsdfpar.har",
  set_input = "path/to/gsdfset.har",
  REG = "full", ACTS = "full", ENDW = "labor_agg"
)

pop <- .data_full$POP
pop$Value <- pop$Value * 1.25

aux_pop <- ems_aux(input = pop, type = "dat", header = "POP")
```

6.2.2. CSV file

A path to a CSV file formatted identically to the data frame format above (set columns + `Value` column). The `header` argument is required.

```
write.csv(pop, "path/to/pop.csv", row.names = FALSE)
aux_pop <- ems_aux(input = "path/to/pop.csv", type = "dat", header = "POP")
```

6.2.3. HAR file

A path to a GTAP header array (`.har`) file. All headers within the file are loaded as a named list. The `header` argument is not required because the metadata is already contained in the file.

```
aux_gdp <- ems_aux(input = "path/to/gdpextra.har", type = "par")
```

6.2.4. Character vector (`type = "set"`)

When `type = "set"`, `input` can take the form a character vector of set elements. Set inputs define a new named set in the loaded data and are not aggregated; the elements provided become the final set. This is useful when a model references a set that is not derived from the GTAP database (e.g., a tax instrument classification).

```
tax_types <- c("prodtax", "pfacttax", "inctax", "inputtax",
               "contax", "invtax", "govtax", "xtax", "mtax")

aux_tax <- ems_aux(input = tax_types, type = "set", header = "ALLTAX")
```

The resulting object is passed to `aux_input` in `ems_data()` and will be available in the loaded data under the name `ALLTAX`.

6.3. Using auxiliary data in `ems_data()`

A single `ems_aux()` output may be passed directly to `aux_input`, or multiple outputs combined in a list:

```
# Single auxiliary input
.data <- ems_data(
  dat_input = "path/to/gsdfdat.har",
  par_input = "path/to/gsdfpar.har",
  set_input = "path/to/gsdfset.har",
  aux_input = aux_pop,
  REG = "big3", ACTS = "macro_sector", ENDW = "labor_agg"
)

# Multiple auxiliary inputs
.data <- ems_data(
  dat_input = "path/to/gsdfdat.har",
  par_input = "path/to/gsdfpar.har",
  set_input = "path/to/gsdfset.har",
  aux_input = list(aux_pop, aux_tax, aux_gdp),
  REG = "big3", ACTS = "macro_sector", ENDW = "labor_agg"
)
```

When `ems_data()` processes auxiliary input, it merges each item into the loaded database before aggregation. The merge behavior differs depending on whether the header already exists in the database.

7. Loading models

7.1. Overview

The `ems_model()` function loads the model and its closure, conducts pre-deployment checks, determines temporal dynamics, and allows for the loading of aggregated coefficient data. The output of this function is a tibble with attributes containing the parsed model input and is a required input to the `"model"` argument of the `ems_deploy()` function.

```
ems_model(model_file, closure_file, var_omit = NULL, ...)
```

Argument	Type	Description
`model_file`	Character	File name in working directory or path to a `.tab` file
`closure_file`	Character	File name in working directory or path to a `.cls` closure file
`var_omit`	Character vector or `NULL`	Variables to substitute with `0` and remove from closure
`...`	Named arguments	Coefficient value injections (numeric, data frame, or CSV path)

Both `model_file` and `closure_file` are required arguments.

8. The model file

Model input in `teems` is currently limited to Tablo (`.tab`) files. Vetted models and their closures are available at teems-models.

Model	Files	Description
GTAPv6	`GTAPv6.tab`, `GTAPv6.cls`	GTAP version 6.2 format
GTAPv7	`GTAPv7.tab`, `GTAPv7.cls`	GTAP version 7.0 format
GTAP_INT	`GTAP_INT.tab`, `GTAP_INT.cls`	GTAP intertemporal
GTAP_RE	`GTAP_RE.tab`, `GTAP_RE.cls`	GTAP rational expectations

Users may load their own models by providing a path to a Tablo file and closure:

```
model <- ems_model(
  model_file = "path/to/custom_model.tab",
  closure_file = "path/to/custom_model.cls"
)
```

We recommend that users seeking to load custom models first review the formatting of the vetted models at teems-models in order to make any necessary modifications for use within the `teems` software ecosystem. In some cases it may be easiest to use existing compatible models as a starting point for modifications. Although the teems-solver does not support all current GEMPack declarations, there is usually a workaround. Running a `diff` on the original GTAP models and those in teems-models is a good starting point for understanding potential incompatibilities.

8.1. Closure selection

Standard model-specific closures are available within the model repository: teems-models. The general format consists of:

1. exogenous
2. A `\n` delimited list of exogenous variables
3. ;
4. rest exogenous
5. ;

8.2. Incompatible features

The teems-solver was originally designed as a GEMPack emulator and we have no intention of attempting to reproduce the very thorough GEMPack documentation on the Tablo scripting language and underlying solution software. *Most* Tablo model files (`.tab`) that work with GEMPack will also work with the teems-solver with some minor modifications. There are however a range of newer GEMPack features that are not compatible and are not likely to be incorporated. Fortunately there are workarounds for the vast majority as described below.

Considerable efforts have been made to correctly parse Tablo files but it is likely that custom Tablo files will need to be modified. If you find an incompatibility not listed here and feel that it should be addressed, please feel free to contact us with your model file. Some of the more frequent incompatibilities and workarounds are listed here.

8.2.1. Supported Tablo statements

The parser recognizes the following Tablo statements: `File`, `Coefficient`, `Read`, `Update`, `Set`, `Subset`, `Formula`, `Assertion`, `Variable`, `Equation`, `Write`, and `Zerodivide`. Statements not in this list are either unsupported (e.g., `Omit`, raising an error) or ignored (`Postsim`, raising a warning).

8.2.2. Set and subset declarations

Multiple + or - operators within a single declaration are not supported. Instead of

```
Set A123 = A1 + A2 + A3;
```

Use

```
Set A12 = A1 + A1;
Set A123 = A12 + A3;
```

Identical set assignment without a `Read` statement is not supported. Instead of

```
Set A # example set A # maximum size 5 read elements from file GTAPSETS header "H2";
Set B # example set # = Set A;
```

Use

```
Set A # example set A # maximum size 5 read elements from file GTAPSETS header "H2";
Set B # example set B # maximum size 5 read elements from file GTAPSETS header "H2";
```

Binary set switches using a colon within a Set definition is not supported. The SLUG coefficient is not used to identify sluggish endowments. Declare these sets explicitly or via set operations.

Instead of

```
Set ENDWM # mobile endowments # = (all,e,ENDW:ENDOWFLAG(e,"mobile") ne 0);
```

Declare sets explicitly within the Tablo file:

```
Set ENDWM # mobile endowment # (capital, unsklab, sklab);
```

Note that sets in the full form (all elements) and subsetted form (model-specific aggregations) can be loaded using `...` with `ems_model()` and `type = "set"` with `ems_aux()`.

Capital goods The `CGDS` sector in v6.2 format models is renamed to `zcgds` by default due to R `data.table` C-locale sorting and preference for all set elements to be lowercase.

8.2.3. Formulas and Equations

No IF statements in Formula or Equation RHS (use sets). Instead of

```
Formula (all,c,COMM)(all,r,REG)
    VCB(c,r) = VDB(c,r) + sum{d,REG, VXSB(c,r,d)} + IF[c in MARG, VST(c,r)];
```

Use

```
Formula (all,c,MARG)(all,r,REG)
    VCB(c,r) = VDB(c,r) + sum{d,REG, VXSB(c,r,d)} + VST(c,r);
Formula (all,c,NMRG)(all,r,REG)
    VCB(c,r) = VDB(c,r) + sum{d,REG, VXSB(c,r,d)};
```

8.3. Variable omissions

Variables specified with `var_omit` will be removed from the model closure and replaced with `0` within the model file. If a variable designated for omission is not found in the model, an error is raised.

Standard v7.0 data format variable omissions: `c("atall", "avaall", "tfe", "tfm", "tgd", "tgm", "tid", "tim")` Standard v6.2 data format variable omissions: `c("atall", "tfd", "avaall", "tf", "tfm", "tgd", "tgm", "tpd", "tpm")`

8.4. Coefficient value injection

The `...` argument in `ems_model()` allows users to assign aggregation-specific values to model coefficients. This is used to inject modified values into existing coefficients as well as provide data to new coefficients (aggregated). All data inputted here must be model-aggregation specific and will not be subject to aggregation. For loading disaggregated data, see `ems_aux()` and `ems_data()`. The left-hand side (name) must correspond to a coefficient (not header) declared within the model input. Three input types are supported, with model "Read" and "Formula" declarations modified accordingly:

8.4.1. Uniform numeric value

A single numeric value can be assigned to a coefficient. All set combinations for that coefficient will receive this uniform value.

```
model <- ems_model(
  model_file = ems_example("GTAP_RE.tab"),
  closure_file = ems_example("GTAP_RE.cls"),
  KAPPA = 0.54321
)
```

Providing a numeric vector of length greater than 1 will raise an error. Instead here use a data frame with the appropriate sets to assign multiple heterogeneous values to a coefficient.

8.4.2. Data frame

A data frame (or data frame extension such as tibble or data.table) can be used to assign heterogeneous values across set combinations. The data frame must contain columns for each set index using the model-specific naming convention (set name + variable index, e.g., `REGr`, `COMMc`, `ALLTIMEt`) and a `Value` column.

```
# Heterogeneous values for SUBPAR coefficient (Read-type)
COMMc <- c("crops", "food", "livestock", "mnfcs", "svces")
REGr <- c("usa", "chn", "row")
ALLTIMEt <- seq(0, 10)

SUBPAR <- expand.grid(
  COMMc = COMMc,
  REGr = REGr,
  ALLTIMEt = ALLTIMEt
)

SUBPAR$Value <- runif(nrow(SUBPAR))

model <- ems_model(
  model_file = ems_example("GTAP_RE.tab"),
  closure_file = ems_example("GTAP_RE.cls"),
  SUBPAR = SUBPAR
)
```

This works for both `Read`-type and `Formula`-type coefficients:

```
# Heterogeneous values for CPHI coefficient (Formula-type)
REGr <- c("usa", "chn", "row")
ALLTIMEt <- seq(0, 10)
```

```
CPHI <- expand.grid(
  REGr = REGr,
  ALLTIMEt = ALLTIMEt
)
CPHI$Value <- runif(nrow(CPHI))

model <- ems_model(
  model_file = ems_example("GTAP_RE.tab"),
  closure_file = ems_example("GTAP_RE.cls"),
  CPHI = CPHI
)
```

8.4.3. CSV file

A path to a CSV file can be used as an alternative to a data frame. The CSV must follow the same structure: set index columns and a `Value` column.

```
# Write coefficient data to CSV
write.csv(SUBPAR, "SUBPAR.csv", row.names = FALSE)

model <- ems_model(
  model_file = ems_example("GTAP_RE.tab"),
  closure_file = ems_example("GTAP_RE.cls"),
  SUBPAR = "SUBPAR.csv"
)
```

8.4.4. Combining injections

Multiple coefficient injections can be combined in a single call along with variable omissions:

```
model <- ems_model(
  model_file = ems_example("GTAP_RE.tab"),
  closure_file = ems_example("GTAP_RE.cls"),
  var_omit = c("atall", "avaall", "tfe", "tfm", "tgd", "tgm", "tid", "tim"),
  KAPPA = 0.03,
  SUBPAR = SUBPAR,
  CPHI = CPHI
)
```

If the coefficient name is not found in the model, an error is raised indicating that the aggregated data coefficient is not declared within the provided model input.

8.4.5. Set naming convention

Set names follow a model-specific concatenation of the standard set name and the variable-specific index (e.g., `REGr`, `COMMc`, `ACTSa`). See the Loading and specifying sets chapter for the full convention table.

8.5. Intertemporal models

For intertemporal models, `ems_model()` determines temporal dynamics by detecting the timestep header in the Tablo file. By default, the expected header in the GTAP-RE model is `"YEAR"`. If a model uses a different header for timestep data, set it via `ems_option_set()`:

```
ems_option_set(timestep_header = "AYRS")
```

If the expected timestep header is not found in the model, an error will be raised with a suggestion to use `ems_option_set()` to configure a custom timestep header. The same concept applies to the number of timesteps:

```
ems_option_set(n_timestep_header = "NINTERVAL")
```

Default values for these settings (and others) may be obtained using:

```
ems_option_get()
```

8.6. Future expansions

- Write bidirectional Tablo ↔ LaTeXconversion script to allow for LaTeXmodel files
- `auto-omit` switch to automatically drop as many variables as possible based on closure and shock specifications
- Model file checks are still rudimentary and a more thorough suite of checks will gradually be put in place

Part III.

Swaps and shocks

9. Closure swaps

9.1. Overview

Simple full variable swaps can be directly inputted as a string into the `swap_in` and `swap_out` arguments of `ems_deploy()`. More complex swaps must be handled with the dedicated `ems_swap()` function.

9.2. Simple (full variable) swaps

Here simple swaps refers swaps in which a variable and all its components are being swapped. This is the most common type of swap and is directly handled within `ems_deploy()`.

A single full variable swap

```
cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  swap_in = "qfd",
  swap_out = "tfd"
)
```

Multiple full variable swaps

```
cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  swap_in = c("qfd", "yp"),
  swap_out = c("tfd", "dppriv")
)
```

Note that swaps (and validity checks) are handled in the order they are inputted and swaps "in" are handled before swaps "out".

9.3. Complex (partial variable) swaps

9.3.1. ems_swap()

`ems_swap()` prepares a partial variable swap for use in the `swap_in` or `swap_out` arguments of `ems_deploy()`. Set arguments in `...` use the model-specific set-index naming convention (set name concatenated with the variable's index letter, e.g. `REGr`, `ACTSa`) to identify which tuples to include in the swap.

```
ems_swap(var, ...)
```

Argument	Type	Description
`var`	Character	Variable name as it appears in the model file
`...`	Named arguments	Set-index pairs identifying the tuples to include in the swap

Partial variable swaps entail swapping parts of a variable (i.e., subsets of the constituitive sets) and must be processed through the `ems_swap()` function. In the above example the full "qfd" and "yp" variables were made exogenous while the full "tfd" and "dppriv" variable become endogenous. Within the GTAPv7.0 model, "qfd" encompasses the "COMM", "ACTS", and "REG" sets:

```
Variable (orig_level=VDFB)(all,c,COMM)(all,a,ACTS)(all,r,REG)
    qfd(c,a,r) # demand for domestic commodity c by activity a in region r #;
```

What if we wish to shock only certain parts of this variable (e.g., a region or commodity) and allow the remainder of the variable to endogenously adjust according to model equations? In this case we would swap on the parts of the variable that we will provide values to and leave the remainder. If, for example, we have a shock value for "asia" within "REG" and "crops" within "ACTS", the following `ems_swap()` call will prepare this swap for input into the `swap_in` argument of `ems_deploy()`:

```
qfd_in <- ems_swap(
  var = "qfd",
  REGr = "asia",
  ACTSa = "crops"
)
```

Note that sets specified in this manner are identified by the model-specific concatenation of the standard set name plus the variable-specific index. This is necessary because many variables contain multiple instances of the same set and none of the `teems` functions relies upon input entry order or order within the model to make distinctions. In this case the "qfd" set subindices are "c", "a", and "r", yielding "COMMc", "ACTSa", and "REGr". In order to retain a valid closure we also use the same approach to the outgoing variable "tfd":

```
tfd_out <- ems_swap(
  var = "tfd",
  REGr = "asia",
  ACTSa = "crops"
)
```

While the subindices for both variables are the same within GTAPv7.0, note that these may vary in some models such as GTAPv6.2 where the REG subindex is "s" for "qfd" and "r" for "tfd". Both swaps may now be loaded within `ems_deploy()`:

```
cmf_path <- ems_deploy(
  data = data,
  model = model,
  swap_in = qfd_in,
  swap_out = tfd_out
)
```

Or if additional full variable swaps are also desired, a list is used to load multiple swaps for each direction

```
cmf_path <- ems_deploy(
  data = data,
  model = model,
  swap_in = list(qfd_in "yp"),
  swap_out = list(tfd_out, "dppriv")
)
```

Finally, selection of multiple elements for each set will expand the swap to encompass all associated tuples. For example:

```
qfd_in <- ems_swap(
  var = "qfd",
  REGr = c("asia", "lam"),
  ACTSa = "crops"
)
```

is equivalent to loading

```
qfd_in_1 <- ems_swap(
  var = "qfd",
  REGr = "asia",
  ACTSa = "crops"
)
```

and

```
qfd_in_2 <- ems_swap(
  var = "qfd",
  REGr = "lam",
  ACTSa = "crops"
)
```

9.4. Future expansions

- List of vetted swaps by model

10. Uniform shocks

10.1. Overview

`ems_uniform_shock()` applies a homogeneous percentage change over an entire variable or a subset of its elements. The set arguments accepted in `...` depend on the variable specified and the set mappings loaded in `ems_data()`.

```
ems_uniform_shock(var, value, ...)
```

Argument	Type	Description
`var`	Character	Variable name as it appears in the model file
`value`	Numeric	Percentage change applied to all targeted elements
`...`	Named arguments	Set-index pairs for partial variable shocks (see Partial variable uniform shocks)

10.2. Full variable uniform shocks

The most simple uniform shock is applied to a full variable, meaning that all combinations of sets associated with that variable are allocated the same exogenous value. In the example below all regions (as determined by selected region mappings) will by increased by 1%.

```
pop_shk <- ems_uniform_shock(var = "pop",
                             value = 1
)
```

Here is an example of the usage above. Numeraire shocks (in static models) can be carried out using a uniform shock:

```
pfact_shk <- ems_uniform_shock(var = "pfactwld",
                               value = 1
)
```

10.3. Partial variable uniform shocks

As with the syntax regarding swaps (link), the scope of a uniform shock may be determined by specifying elements within the sets associated with the shocked variable. For example, if we have a region "chn" and wish to allocate a shock specifically to this region:

```
partial <- ems_uniform_shock(var = "aoall",
                             REGr = "chn",
                             value = -1
)
```

Here is an example of a partial variable swap followed by a partial variable uniform shock. If a full swap is followed by a partial variable shock (example), the unshocked elements will remain at 0.

In the case of intertemporal models, the shock year may be specified using a chronological or timestep format (see "Time step summary" output from `ems_data()`).

```
partial <- ems_uniform_shock(
  var = "aoall",
  REGr = "chn",
  PROD_COMMj = "crops",
  Year = 2017,
  value = -1
)
```

Here is a full example using chronological formats and a uniform shock.

Note that sets specified in this manner are identified by the model-specific concatenation of the standard set name plus the variable-specific index. Also note that exogenous components of a variable not allocated a shock (all other regions except "chn") will not vary.

10.4. Multiple shocks and mixed swaps

Multiple uniform shocks of different scope can be combined in a list and passed to `ems_deploy()`. Swaps can also be mixed, combining partial `ems_swap()` objects with full variable strings:

```
partial <- ems_uniform_shock(
  var = "qfd",
  REGs = "usa",
  PROD_COMMj = "crops",
  value = -1
)

full <- ems_uniform_shock(
  var = "yp",
  value = 0.1
```

```
)

qfd <- ems_swap(
  var = "qfd",
  REGs = "usa",
  PROD_COMMj = "crops"
)

tfd <- ems_swap(
  var = "tfd",
  REGr = "usa",
  PROD_COMMj = "crops"
)

cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  shock = list(partial, full),
  swap_in = list(qfd, "yp"),
  swap_out = list(tfd, "dppriv")
)
```

11. Custom shocks

11.1. Overview

`ems_custom_shock()` allows for heterogeneous shocks specified on a tuple-by-tuple basis, supplied as a data frame or CSV file through `input`. Values are denominated in percentage change. Only the tuples present in `input` are shocked.

```
ems_custom_shock(var, input)
```

Argument	Type	Description
`var`	Character	Variable name as it appears in the model file
`input`	Data frame or Character	Data frame or path to a CSV file containing set columns and a `Value` column (percentage changes)

11.2. Full variable custom shocks

Custom shocks can be carried out over the full variable, with every tuple receiving a specified shock value. Here is a demonstration of 4 custom shocks carried out on all variable tuples for a range of variable dimensions with the GTAP-INT model.

Elements for data frame construction

```
time_steps <- c(0, 1, 2, 3)
REG <- c("chn", "usa", "row")
ENDW_COMM <- c("labor", "capital", "natlres", "land")
TRAD_COMM <- c("svces", "food", "crops", "mnfcs", "livestock")
PROD_COMM <- c("svces", "food", "crops", "mnfcs", "livestock", "zcgds")
MARG_COMM <- "svces"
ALLTIME <- seq(0, length(time_steps) - 1)
```

2D data frame and shock

```
pop <- expand.grid(
  REGr = REG,
  ALLTIMEt = ALLTIME,
  stringsAsFactors = FALSE
)

pop <- pop[do.call(order, pop), ]
pop$Value <- runif(nrow(pop))

pop_shk <- ems_custom_shock(
  var = "pop",
  input = pop
)
```

3D data frame and shock

```
aoall <- expand.grid(
  PROD_COMMj = PROD_COMM,
  REGr = REG,
  ALLTIMEt = ALLTIME,
  stringsAsFactors = FALSE
)

aoall <- aoall[do.call(order, aoall), ]
aoall$Value <- runif(nrow(aoall))

aoall_shk <- ems_custom_shock(
  var = "aoall",
  input = aoall
)
```

4D data frame and shock

```
afeall <- expand.grid(
  ENDW_COMMi = ENDW_COMM,
  PROD_COMMj = PROD_COMM,
  REGr = REG,
  ALLTIMEt = ALLTIME,
  stringsAsFactors = FALSE
)

afeall <- afeall[do.call(order, afeall), ]
afeall$Value <- runif(nrow(afeall))

afeall_shk <- ems_custom_shock(
  var = "afeall",
```

```
  input = afeall
)
```

5D data frame and shock

```
atall <- expand.grid(
  MARG_COMMm = MARG_COMM,
  TRAD_COMMi = TRAD_COMM,
  REGr = REG,
  REGs = REG,
  ALLTIMEt = ALLTIME,
  stringsAsFactors = FALSE
)

atall <- atall[do.call(order, atall), ]
atall$Value <- runif(nrow(atall))

atall_shk <- ems_custom_shock(
  var = "atall",
  input = atall
)
```

Multiple custom shocks are combined in a list for `ems_deploy()`:

```
cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  shock = list(pop_shk, aoall_shk, afeall_shk, atall_shk)
)
```

Custom shocks can also be loaded from CSV files by passing a file path to `"input"` instead of a data frame.

11.3. Partial variable custom shocks

Custom shocks are only carried out on tuples included within the input data frame or CSV. Here we carry out a custom shock (heterogenous values) across part of the `aoall` variable. The non-shocked values will remain at `0` due to their exogenous status.

```
aoall_full <- expand.grid(
  PROD_COMMj = PROD_COMM,
  REGr = REG,
  ALLTIMEt = ALLTIME,
  stringsAsFactors = FALSE
)
```

```
aoall_full$Value <- 0
aoall_full <- aoall_full[do.call(order, aoall_full), ]

aoall <- aoall_full[aoall_full$PROD_COMMj == "crops" & aoall_full$REGr == "chn", ]
aoall$Value <- runif(nrow(aoall))

aoall_shk <- ems_custom_shock(var = "aoall", input = aoall)
```

12. Scenario shocks

12.1. Overview

`ems_scenario_shock()` is an intertemporal shock type that specifies heterogeneous shocks at the tuple level as absolute values. Inputs must cover all pre-aggregation tuples and span the full extent of the variable — no partial variable scenario shocks are permitted. Values are aggregated according to the set mappings in `ems_data()` and then converted to percentage changes internally, making scenario shocks portable across different model aggregations.

```
ems_scenario_shock(var, input)
```

Argument	Type	Description
`var`	Character	Variable name as it appears in the model file
`input`	Data frame or Character	Data frame or path to a CSV file containing all pre-aggregation tuples, a `Year` column (chronological years), and a `Value` column (absolute values)

12.2. Using scenario shocks

Scenario shocks differ from custom shocks in a number of respects:

1. Only available for intertemporal models
2. Inputs must contain all *preaggregation* tuples associated with a variable
3. Inputs must span all elements (no partial variable scenario shock)
4. Inputs are denominated in actual values (e.g., million USD)
5. The input dataframe or CSV must contain one column "Year", corresponding to the chronological year for a shock in a specific tuple

There are several applications where it is advantageous to use scenario shock rather than a custom shock. The primary advantage is that scenario shocks allow for seamless changes to model aggregations since the shocks themselves are subject to mappings and aggregation. If we have trajectories available for all tuples within a given variable, a scenario shock will adjust to set mapping inputs in `ems_data()`. The actual values provided do not necessarily need to correspond to realworld values. We could for example use the integer 1 as a base and vary our components according to this base value in the year corresponding to t0.

12.3. Population trajectory example

Here is an example that takes base year population data from the GTAP database (typically not read into the model) and constructs hypothetical pathways for every region in the model. These pathways are valid for any regional aggregation chosen.

Grab population data with `REG` set to "full" to keep regions disaggregated

```
pop <- ems_data(
  dat_input = "~/dat/GTAP/v11c/flexAgg11c17/gsdfdat.har",
  par_input = "~/dat/GTAP/v11c/flexAgg11c17/gsdfpar.har",
  set_input = "~/dat/GTAP/v11c/flexAgg11c17/gsdfset.har",
  REG = "full",
  ACTS = "macro_sector",
  ENDW = "labor_agg"
)$POP
```

Construct a data frame with random growth rates through 2033 using the 2017 base year data

```
pop$Year <- 2017
regions <- unique(pop$REG)
pop_traj <- expand.grid(
  REG = regions,
  Value = 0,
  Year = c(2018, 2019, 2020, 2021, 2023, 2025, 2027, 2029, 2031, 2033),
  stringsAsFactors = FALSE
)
pop <- rbind(pop, pop_traj)

growth_rates <- data.frame(
  REG = regions,
  growth_rate = runif(length(regions), min = -0.01, max = 0.05)
)

pop <- merge(pop, growth_rates, by = "REG")
base_values <- pop[pop$Year == 2017, c("REG", "Value")]
names(base_values)[2] <- "base_value"
pop <- merge(pop, base_values, by = "REG")

pop$Value[pop$Year > 2017] <-
  pop$base_value[pop$Year > 2017] *
    (1 + pop$growth_rate[pop$Year > 2017])^(pop$Year[pop$Year > 2017] - 2017)
pop$growth_rate <- NULL
pop$base_value <- NULL
pop <- pop[order(pop$REG, pop$Year), ]
pop <- pop[, c("REG", "Year", "Value")]
colnames(pop)[1] <- "REGr"
```

And load as a scenario shock

```
pop_trajectory <- ems_scenario_shock(
  var = "pop",
  input = pop
)
```

Note that the object “pop” may also be saved as a .csv and loaded directly from storage.

Part IV.

Deploying and solving the model

13. Deploying the model

13.1. Overview

`ems_deploy()` consolidates all user inputs and carries out all operations necessary to run a CGE model. It accumulates and validates data and model inputs, processes shocks and closure swaps, and writes all required input files to disk. The output file path serves as a required input to `ems_solve()`.

13.2. Arguments

Argument	Default	Description
`.data`		A list of data.tables generated by `ems_data()`
`model`		A tibble generated by `ems_model()`
`shock`	`NULL`	A shock object or list of shock objects produced by `ems_uniform_shock()`, `ems_custom_shock()`, or `ems_scenario_shock()`. When `NULL`, a null shock is carried out which effectively returns base data
`swap_in`	`NULL`	Character vector, `ems_swap()` object, or a list of any combination. Variables to be made exogenous
`swap_out`	`NULL`	Character vector, `ems_swap()` object, or a list of any combination. Variables to be made endogenous
`write_dir`	`tools::R_user_dir("teems", "cache")`	Base directory where the "teems" subdirectory will be created for input files and model solution
`shock_file`	`NULL`	Path to a CSV representing a final shock file. No checks or modifications will be conducted on this file

13.3. Basic usage

The simplest deployment requires only data and model inputs. With no shock supplied, a null shock is carried out:

```
cmf_path <- ems_deploy(
  .data = .data,
  model = model
)
```

13.4. Deploying with shocks and swaps

A single shock can be passed directly:

```
shock <- ems_uniform_shock(var = "pop", value = 1)

cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  shock = shock
)
```

Multiple shocks are combined in a list:

```
pop_shk <- ems_uniform_shock(var = "pop", value = 1)
aoall_shk <- ems_custom_shock(var = "aoall", input = aoall_df)

cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  shock = list(pop_shk, aoall_shk)
)
```

13.5. Closure swaps

Full variable swaps can be passed as strings. Partial swaps use `ems_swap()` objects. These can be mixed in a list:

```
# Full variable swap (strings)
cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  shock = shock,
  swap_in = "qfd",
  swap_out = "tfd"
)

# Mixed partial and full swaps
qfd <- ems_swap(var = "qfd", REGs = "usa", PROD_COMMj = "crops")
```

```
tfd <- ems_swap(var = "tfd", REGr = "usa", PROD_COMMj = "crops")

cmf_path <- ems_deploy(
  .data = .data,
  model = model,
  shock = list(partial, full),
  swap_in = list(qfd, "yp"),
  swap_out = list(tfd, "dppriv")
)
```

13.6. User generated inputs

`ems_deploy()` allows for user-generated shock files via the `shock_file` argument. In the case of shock files, no checks or operations will be conducted (i.e., the shock file as loaded is considered the final input). User-supplied closure files will however be processed if swaps are supplied.

13.7. Persistent files

Good practice in R dictates that a user's machine state is not modified. In order to comply with this guidance, `teems` will only write to temporary folders associated with the `teems` package and these files will be removed upon closure of the session or loading of a subsequent model. If persistent data is desired, users must provide a path to a local directory via `write_dir`:

```
cmf_path <- ems_deploy(
  write_dir = "~/my_project/output",
  .data = .data,
  model = model,
  shock = shock
)
```

14. Solving the model

14.1. Overview

`ems_solve()` solves the constrained optimization problem according to a range of runtime configuration options. In order to solve, a teems Docker image must be prebuilt. Singularity, accuracy, and error checks are carried out following a successful run. By default, solver outputs are automatically parsed into structured R objects via `ems_compose()`.

14.2. Arguments

Argument	Default	Description
`cmf_path`	`NULL`	Path to the CMF file generated by `ems_deploy()`
`n_tasks`	`1L`	Number of tasks to run in parallel. Must be `1L` if `matrix_method` is `"LU"`
`n_subintervals`	`1L`	Number of subintervals for the applied shock. More subintervals may alleviate accuracy issues stemming from large shock magnitudes
`matrix_method`	`"LU"`	Matrix solution method (see below)
`solution_method`	`"Johansen"`	Solution method (see below)
`steps`	`c(2L, 4L, 8L)`	Vector of steps for the modified midpoint method. Must be all odd or all even, length 3
`laA`	`300L`	Memory parameter (%) for `"LU"` and `"SBBD"` methods
`laD`	`200L`	Memory parameter (%) for `"DBBD"` and `"NDBBD"` methods
`laDi`	`500L`	Memory parameter (%) for `"NDBBD"` method
`terminal_run`	`FALSE`	When `TRUE`, exits without running the solver, allowing the user to close R and run from the terminal
`suppress_outputs`	`FALSE`	When `TRUE`, solver outputs are not automatically parsed with `ems_compose()`

14.3. Basic usage

```
outputs <- ems_solve(
  cmf_path = cmf_path,
  matrix_method = "LU",
  solution_method = "Johansen"
)
```

14.4. Matrix methods

Method	Description
`"LU"`	Standard LU decomposition. The most robust and potentially slowest for large models. `n_tasks` must be `1L`. Works with both static and dynamic models
`"DBBD"`	Doubly bordered block diagonal. Parallel solution method for static models. Potentially faster than `"LU"` although less robust
`"SBBD"`	Singly bordered block diagonal. Parallel solution method for intertemporal models. Potentially faster than `"LU"` although less robust
`"NDBBD"`	Nested doubly bordered block diagonal. Parallel solution method for large intertemporal models with many timesteps

14.5. Solution methods

Method	Description
`"Johansen"`	Fast one-step approximation. Should only be used as a rough approximation due to handling of nonlinear equations. See COPS manual
`"mod_midpoint"`	Modified midpoint method that performs multiple passes for improved accuracy. The `steps` parameter controls the number of passes. See COPS manual

14.6. Memory parameters

If the solver returns "Error return from MA48B/BD because LA is ...", increase the relevant memory parameter gradually:

- `laA` applies to `"LU"` and `"SBBD"` methods

- `laD` applies to `"DBBD"` and `"NDBBD"` methods
- `laDi` applies to the `"NDBBD"` method only

```
outputs <- ems_solve(
  cmf_path = cmf_path,
  matrix_method = "LU",
  solution_method = "Johansen",
  laA = 500L
)
```

14.7. Parallel solving

For models that support parallel matrix methods, increase `n_tasks`:

```
outputs <- ems_solve(
  cmf_path = cmf_path,
  n_tasks = 4L,
  matrix_method = "SBBD",
  solution_method = "Johansen"
)
```

14.8. Multi-step solution

For greater accuracy, use the modified midpoint method with subintervals:

```
outputs <- ems_solve(
  cmf_path = cmf_path,
  n_subintervals = 3L,
  matrix_method = "LU",
  solution_method = "mod_midpoint",
  steps = c(2L, 4L, 8L)
)
```

14.9. Terminal mode

For long-running models, `terminal_run` allows the user to close any R IDE prior to running from the terminal:

```
ems_solve(
  cmf_path = cmf_path,
  matrix_method = "LU",
  solution_method = "Johansen",
  terminal_run = TRUE
)
```

15. Solving in-situ

15.1. Overview

`solve_in_situ()` is a wrapper for the teems-solver which conducts a minimal number of checks prior to attempting to solve the constrained optimization problem. In contrast to `ems_solve()`, all model input files must be provided by the user in their final form. No checks or modifications are conducted on input files used in this manner.

This function is intended for users who wish to use the teems solver with their own pre-prepared input files, bypassing the `ems_data()` / `ems_model()` / `ems_deploy()` pipeline, or in combination with it by reusing the files it writes to disk.

15.2. Arguments

Argument	Default	Description
`...`		Named arguments corresponding to input files necessary for an in-situ model run. Names must correspond to "File" statements within the model Tablo file. Values correspond to file paths where these files are found
`model_file`		Path to the model Tablo (.tab) file
`closure_file`		Path to the closure (.cls) file
`shock_file`		Path to the shock file
`write_dir`	`tools::R_user_dir("teems", "cache")`	Base directory for output files
`writeout`	`FALSE`	Whether to parse the Tablo file and append "Write" declarations for all model output coefficients and sets. If `TRUE`, a successful writeout may require modifications to the provided `model_file`. If `FALSE`, outputs will consist of model variables only
`n_tasks`	`1L`	Number of tasks to run in parallel. Must be `1L` if `matrix_method` is `"LU"`
`n_subintervals`	`1L`	Number of subintervals for the applied shock

Argument	Default	Description
`matrix_method`	`"LU"`	Matrix solution method: `"LU"`, `"DBBD"`, `"SBBD"`, or `"NDBBD"`
`solution_method`	`"Johansen"`	Solution method: `"Johansen"` or `"mod_midpoint"`
`steps`	`c(2L, 4L, 8L)`	Vector of steps for the modified midpoint method
`laA`	`300L`	Memory parameter (%) for `"LU"` and `"SBBD"` methods
`laD`	`200L`	Memory parameter (%) for `"DBBD"` and `"NDBBD"` methods
`laDi`	`500L`	Memory parameter (%) for `"NDBBD"` method
`terminal_run`	`FALSE`	When `TRUE`, exits without running the solver
`suppress_outputs`	`FALSE`	When `TRUE`, returns the CMF file path instead of parsed outputs

15.3. Input files

All model-declared input files as well as `model_file`, `closure_file`, and `shock_file` are required. Input files are passed as named arguments via `...`, where each name must correspond to a "File" statement in the model Tablo file. For GTAP-RE, the Tablo file declares:

```
File GTAPDATA # base data file;
File GTAPINT  # intertemporal data file;
File GTAPPARM # parameter file;
File GTAPSETS # set file;
```

These names are then used as the named arguments to `solve_in_situ()`.

15.4. Using with `ems_deploy()` output

A common use case is to run the standard pipeline to generate validated input files, and then re-solve with different solver options using `solve_in_situ()`. The files written by `ems_deploy()` can be located in the write directory:

```r
library(teems)

# Standard pipeline to generate input files
data <- ems_data(
  dat_input = "~/dat/GTAP/v11c/flexAgg11c17/gsdfdat.har",
  par_input = "~/dat/GTAP/v11c/flexAgg11c17/gsdfpar.har",
```

```
  set_input = "~/dat/GTAP/v11c/flexAgg11c17/gsdfset.har",
  REG = "big3",
  ACTS = "macro_sector",
  ENDW = "labor_agg",
  time_steps = c(0, 1, 2, 3, 4, 6, 8, 10, 12, 14, 16)
)

model <- ems_model(
  model_file = "GTAP-REv1.tab",
  closure_file = "GTAP-REv1.cls",
  var_omit = c("atall", "avaall", "tfe", "tfd", "tfm", "tgd", "tgm", "tid", "tim")
)

shock <- ems_scenario_shock(var = "pop", input = pop)

write_dir <- tempdir()
cmf_path <- ems_deploy(
  write_dir = write_dir,
  .data = data,
  model = model,
  shock = shock
)

# Re-solve in-situ using the files written by ems_deploy()
solve_in_situ(
  GTAPDATA = file.path(write_dir, "teems", "GTAPDATA.txt"),
  GTAPINT  = file.path(write_dir, "teems", "GTAPINT.txt"),
  GTAPPARM = file.path(write_dir, "teems", "GTAPPARM.txt"),
  GTAPSETS = file.path(write_dir, "teems", "GTAPSETS.txt"),
  model_file = "GTAP-REv1.tab",
  closure_file = "GTAP-REv1.cls",
  shock_file   = list.files(file.path(write_dir, "teems"),
                            pattern = "shf", full.names = TRUE),
  write_dir = "~/in_situ_run/",
  n_tasks = 1,
  n_subintervals = 1,
  matrix_method = "SBBD",
  solution_method = "mod_midpoint",
  writeout = TRUE
)
```

15.5. Output modes

By default, `solve_in_situ()` returns a list of data.tables containing model variables. The `writeout` parameter controls whether coefficients and sets are also included:

- `writeout = FALSE` (default): returns a list of data.tables with model variables only
- `writeout = TRUE`: returns a tibble containing model output variables, coefficients, and sets. Note that a successful writeout may require modifications to the Tablo file provided

If `suppress_outputs` is `TRUE`, the function returns the CMF file path instead of parsed outputs. This path can be passed to `ems_compose()` for manual parsing.

15.6. Persistent output

As with `ems_deploy()`, output files are written to a temporary directory by default and removed on session close. Specify `write_dir` for persistent output:

```
outputs <- solve_in_situ(
  GTAPDATA = "path/to/GTAPDATA.txt",
  GTAPINT  = "path/to/GTAPINT.txt",
  GTAPPARM = "path/to/GTAPPARM.txt",
  GTAPSETS = "path/to/GTAPSETS.txt",
  model_file  = "path/to/GTAP-REv1.tab",
  closure_file = "path/to/GTAP-REv1.cls",
  shock_file   = "path/to/shocks.shf",
  write_dir = "~/my_project/output",
  matrix_method = "LU",
  solution_method = "Johansen"
)
```

16. Composing results

16.1. Overview

`ems_compose()` retrieves and processes results from a solved model run. Results are parsed according to the specified type (variables, coefficients, or all). Data validation and consistency checks are performed during the parsing process.

By default, `ems_solve()` calls `ems_compose()` automatically, so this function is primarily used when `suppress_outputs = TRUE` was passed to `ems_solve()` or `solve_in_situ()`, or when re-parsing results from a previous run.

16.2. Arguments

Argument	Default	Description
`cmf_path`		Path to the CMF file generated by `ems_deploy()`
`type`	`"all"`	Type of data to parse: `"all"`, `"variable"`, or `"coefficient"`
`name`	`NULL`	Character vector to filter results by name, returning a subset of the selected type
`minimal`	`FALSE`	Whether to run a minimal number of checks and modifications to output data

16.3. Basic usage

After a standard model run with suppressed outputs:

```
ems_solve(
  cmf_path = cmf_path,
  matrix_method = "LU",
  solution_method = "Johansen",
  suppress_outputs = TRUE
)
```

```
# Parse all results
outputs <- ems_compose(cmf_path = cmf_path)
```

16.4. Output types

The `type` argument controls what is parsed from the solver output:

```
# All model variables and coefficients
all_outputs <- ems_compose(cmf_path = cmf_path, type = "all")

# Percentage change values for model variables only
variables <- ems_compose(cmf_path = cmf_path, type = "variable")

# Values for model coefficients only
coefficients <- ems_compose(cmf_path = cmf_path, type = "coefficient")
```

16.5. Filtering by name

The `name` argument filters results to a subset of the selected type:

```
# Retrieve a single variable
qfd <- ems_compose(cmf_path = cmf_path, type = "variable", name = "qfd")

# Retrieve multiple variables
trade_vars <- ems_compose(cmf_path = cmf_path, type = "variable",
                          name = c("qfd", "qfm", "qgd"))
```

16.6. Minimal mode

When `minimal = TRUE`, a reduced number of checks and modifications are applied to the output data:

1. Chronological data is not loaded and the model file is not parsed
2. No post-model set checks are conducted between model sets as written out from the model input and solver set binaries
3. Time steps remain in their model format (integer indices rather than chronological years)

This option must be set to `TRUE` when using output from `solve_in_situ()` with `writeout = FALSE`, since there is no set writeout to validate against:

```
outputs <- solve_in_situ(
  ...,
  model_file  = "model.tab",
  closure_file = "model.cls",
  shock_file   = "shocks.shf",
  suppress_outputs = TRUE
)

variables <- ems_compose(cmf_path = outputs, type = "variable", minimal = TRUE)
```

Part V.

Configuration

17. Package options

17.1. Overview

teems provides a set of advanced options that control package behavior across function calls. Options are managed through three functions:

- `ems_option_get()` retrieves current option values
- `ems_option_set()` modifies option values
- `ems_option_reset()` restores all options to their defaults

Options persist for the duration of the R session and are reset on package reload.

17.2. Retrieving options

View all current options:

```
ems_option_get()
```

Retrieve a specific option by name:

```
ems_option_get("verbose")
ems_option_get("ndigits")
```

17.3. Setting options

One or more options can be set in a single call:

```
ems_option_set(verbose = FALSE)
ems_option_set(verbose = FALSE, ndigits = 8)
```

17.4. Resetting options

Reset all options to their default values:

```
ems_option_reset()
```

17.5. Available options

Option	Default	Description
`verbose`	`TRUE`	If `FALSE`, function-specific diagnostics are silenced
`ndigits`	`6`	Number of digits to the right of the decimal point written to file for numeric type double. Passed to `format()` nsmall and `round()` digits
`docker_tag`	`"latest"`	Docker tag specifying which teems Docker image to use
`accuracy_threshold`	`0.8`	Threshold (converted to a percentage) against which 4-digit precision is compared. A warning is generated if the threshold is not met
`write_sub_dir`	`"teems"`	Name of the subdirectory within the base `write_dir` set by `ems_deploy()`
`margin_sectors`	`c("atp", "otp", "wtp")`	Default margin sectors
`check_shock_status`	`TRUE`	If `FALSE`, no check on shock element endogenous/exogenous status is conducted
`expand_ETRE`	`TRUE`	If `TRUE`, missing tuples in the ETRE (ETRAE) data header are added with value -1e-05 and the sluggish endowment set is replaced with the general endowment set. The ETRE header is not consistent across databases regarding inclusion of non-sluggish endowments
`full_exclude`	`c("DREL", "DVER", "XXCR", "XXCD", "XXCP", "SLUG", "EFLG")`	Headers to fully exclude from all aspects of the model run. Failure to designate these headers properly will result in errors because some headers cannot be aggregated or mapped. Modify with caution
`timestep_header`	`"YEAR"`	Coefficient containing a numeric vector of timestep intervals. For novel intertemporal models - modify with caution

`n_timestep_header`	`"NTSP"`	Coefficient containing a numeric vector length one with sum of timestep intervals. For novel intertemporal models - modify with caution

Part VI.

Examples

18. Example files

18.1. Overview

`ems_example()` writes bundled example model and script files to disk. It is the recommended starting point for new users, providing ready-to-run material for each of the four vetted model variants without requiring any additional setup beyond GTAP data access.

Two output modes are available, controlled by the `type` argument:

- `"model_files"` (default): writes the model (`.tab`) and closure (`.cls`) files for the chosen model.
- `"scripts"`: writes a complete set of example R scripts covering the full range of shock, swap, and solver configurations supported by that model. Each script is a self-contained, runnable workflow from data loading through to result parsing.

18.2. Arguments

Argument	Default	Description
`model`		Model name. One of `"GTAPv6"`, `"GTAPv7"`, `"GTAP-INT"`, `"GTAP-RE"`
`write_dir`	`tools::R_user_dir("teems", "cache")`	Directory where files are written
`type`	`"model_files"`	Output type: `"model_files"` or `"scripts"`
`dat_input`	`NULL`	Path to the basedata HAR file. Required when `type = "scripts"`
`par_input`	`NULL`	Path to the parameters HAR file. Required when `type = "scripts"`
`set_input`	`NULL`	Path to the sets HAR file. Required when `type = "scripts"`
`target_format`	`NULL`	Target GTAP format for data conversion (e.g. `"GTAPv7"`). Required when `type = "scripts"` and a format conversion is needed

18.3. Writing model files

The simplest use of `ems_example()` extracts a model's Tablo and closure files so they can be inspected or used directly with `ems_model()` or `solve_in_situ()`.

```
library(teems)

# Write GTAPv7 model files to the default cache directory
paths <- ems_example("GTAPv7")

# Or to a specific directory
paths <- ems_example("GTAP-RE", write_dir = "~/my_project/model")
```

The returned list contains two named paths:

```
paths$model_file   # path to the .tab file
paths$closure_file # path to the .cls file
```

These can be passed directly to `ems_model()`:

```
model <- ems_model(
  model_file   = paths$model_file,
  closure_file = paths$closure_file
)
```

18.4. Writing example scripts

When `type = "scripts"`, `ems_example()` writes a directory of R scripts covering the full range of configurations available for the chosen model — uniform shocks, custom shocks, partial shocks, closure swaps, scenario shocks (for intertemporal models), and null runs. Each script is injected with the supplied data paths and is immediately runnable.

```
paths <- ems_example(
  model      = "GTAPv7",
  write_dir  = "~/my_project",
  type       = "scripts",
  dat_input  = "~/dat/GTAP/v10/flexAgg/gsddat.har",
  par_input  = "~/dat/GTAP/v10/flexAgg/gsdpar.har",
  set_input  = "~/dat/GTAP/v10/flexAgg/gsdset.har"
)
```

If the HAR files are in a format that requires conversion (e.g. GTAP v9 files being used with a v7.0 model), supply `target_format`:

```
paths <- ems_example(
  model         = "GTAPv7",
  write_dir     = "~/my_project",
  type          = "scripts",
  dat_input     = "~/dat/GTAP/v9/flexAgg/gddat.har",
  par_input     = "~/dat/GTAP/v9/flexAgg/gdpar.har",
  set_input     = "~/dat/GTAP/v9/flexAgg/gdset.har",
  target_format = "GTAPv7"
)
```

18.5. Using the generated scripts

`ems_example()` returns a character vector of paths — one per written script. Each script is ready to run; the data paths supplied to `ems_example()` are injected at the top of the file as plain assignments.

18.5.1. Script naming convention

Script file names encode the shock and swap configuration:

File name	What it demonstrates
`null.R`	Zero-shock run — useful for verifying the model solves before applying shocks
`numeraire.R`	Uniform 1% population shock — minimal working example
`full_uniform.R`	Uniform shock across all elements of a variable
`part_uniform.R`	Uniform shock on a subset of elements
`part_uniform_full_swap.R`	Partial shock plus a full-variable closure swap
`part_uniform_part_swap.R`	Partial shock plus a partial closure swap
`part_uniform_part_swap_mixed.R`	Partial shock plus mixed (full + partial) swaps
`part_uniform_year.R`	Partial shock filtered to a single time step (intertemporal models)
`custom_full.R`	Heterogeneous shock values supplied as a data frame
`custom_full_csv.R`	Heterogeneous shock values read from a CSV file
`custom_full_year.R`	Heterogeneous shock with a time dimension
`scenario.R`	Scenario shock from absolute values (intertemporal models only)

18.5.2. Structure of a generated script

Each script begins with the injected assignments followed by the template body. Opening `part_uniform.R` in an editor shows:

```
dat_input = "~/dat/GTAP/v10/flexAgg/gsddat.har"
par_input = "~/dat/GTAP/v10/flexAgg/gsdpar.har"
set_input = "~/dat/GTAP/v10/flexAgg/gsdset.har"
target_format = NULL
write_dir = "~/my_project"
model_file = "~/my_project/GTAP-INT.tab"
closure_file = "~/my_project/GTAP-INT.cls"

.data <- ems_data(
  dat_input = dat_input,
  par_input = par_input,
  set_input = set_input,
  REG = "big3",
  PROD_COMM = "macro_sector",
  ENDW_COMM = "labor_agg",
  time_steps = c(0, 1, 2),
  target_format = target_format
)

model <- ems_model(
  model_input = model_input,
  closure_file = closure_file
)

partial <- ems_uniform_shock(
  var = "aoall",
  REGr = "chn",
  PROD_COMMj = "crops",
  value = -1
)

cmf_path <- ems_deploy(
  write_dir = write_dir,
  .data = .data,
  model = model,
  shock = partial
)

outputs <- ems_solve(
  cmf_path = cmf_path,
  matrix_method = "LU",
  solution_method = "Johansen"
)
```

18.6. Output

`ems_example()` returns a character vector of file paths to the written files. For `type = "model_files"` this is the `.tab` and `.cls` paths. For `type = "scripts"` this is one path per written script.

By default, files are written to `tools::R_user_dir("teems", "cache")` and do not persist across sessions. Specify `write_dir` to control the output location as well as generate persistent files.