

The {teems} R package user manual

Matthew Cantele

Table of contents

1. Introduction	1
1.1. Motivation	1
1.2. Errors and model compatibility	2
1.3. About this manual	2
 I. Getting started	 3
2. Dependencies	4
2.1. Data	4
2.2. R	4
2.3. teems-R package	5
2.4. teems-solver	5
3. Quickstart	6
3.1. Generate example scripts	6
3.2. Step 1: verify the solver with a null run	6
3.3. Step 2: run a uniform shock	7
3.4. Next steps	7
4. Example files	8
4.1. Overview	8
4.2. Arguments	8
4.3. Writing model files	8
4.4. Writing example scripts	9
4.5. Using the generated scripts	10
4.5.1. Script naming convention	10
4.5.2. Structure of a generated script	10
4.6. Output	11
 II. Data and model inputs	 12
5. Loading and specifying sets	13
5.1. Overview	13
5.2. Internal mappings	13
5.2.1. Region (REG)	13
5.2.2. Sector (PROD_COMM, ACTS)	14
5.2.3. Endowment (ENDW_COMM, ENDW)	14
5.3. External mappings	14
5.4. Set naming convention	15

6. Loading data	16
6.1. Overview	16
6.2. Input files	16
6.3. Set mappings	17
6.4. Time steps	17
6.5. Converting formats	18
6.5.1. Loading HAR files as arrays	18
6.5.2. Converting between formats	19
6.6. Data aggregation	19
6.6.1. GTAP v6.2 format parameter weight methodology	20
6.6.2. GTAP v7.0 format parameter weight methodology	21
7. Loading models	23
7.1. Overview	23
7.2. The model file	23
7.3. Closure selection	24
7.4. Incompatible features	24
7.4.1. Supported Tablo statements	25
7.4.2. Set and subset declarations	25
7.4.3. Formulas and Equations	26
7.5. Variable omissions	26
7.6. Coefficient value injection	26
7.6.1. Uniform numeric value	26
7.6.2. Data frame	27
7.6.3. CSV file	28
7.6.4. Combining injections	28
7.6.5. Set naming convention	28
7.7. Intertemporal models	28
III. Swaps and shocks	30
8. Closure swaps	31
8.1. Overview	31
8.2. Simple (full variable) swaps	31
8.3. Complex (partial variable) swaps	32
8.3.1. <code>ems_swap()</code>	32
8.3.2. Subset swaps	34
9. Uniform shocks	35
9.1. Overview	35
9.2. Full variable uniform shocks	35
9.3. Partial variable uniform shocks	35
9.4. Subset shocks	36
9.5. Multiple shocks and mixed swaps	37
10. Custom shocks	38
10.1. Overview	38
10.2. Full variable custom shocks	38

Table of contents

10.3. CSV input	40
10.4. Partial variable custom shocks	41
10.5. Chronological year input	41
11. Scenario shocks	43
11.1. Overview	43
11.2. Using scenario shocks	43
11.3. Population trajectory example	44
IV. Deploying and solving the model	46
12. Deploying the model	47
12.1. Overview	47
12.2. Arguments	47
12.3. Basic usage	47
12.4. Deploying with shocks and swaps	48
12.5. Closure swaps	48
12.6. User-generated inputs	49
13. Solving the model	50
13.1. Overview	50
13.2. Arguments	50
13.3. Basic usage	51
13.4. Solution methods	51
13.5. Matrix methods	51
13.6. Multi-step solution	51
13.7. Memory parameters	52
13.8. Parallel solving	52
13.9. Terminal mode	52
13.10 Solving in-situ	53
13.10.1 Arguments	53
13.10.2 Input files	54
13.10.3 Output modes	55
14. Composing results	56
14.1. Overview	56
14.2. Arguments	56
14.3. Basic usage	56
14.4. Filtering output	57
14.5. Return value	57
V. Configuration	58
15. Package options	59
15.1. Overview	59
15.2. Retrieving options	59
15.3. Setting options	59

Table of contents

15.4. Resetting options	59
15.5. Available options	60

1. Introduction

Reproducibility within computable general equilibrium (CGE) and partial equilibrium modeling 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 Modeling 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 are 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 sheer 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.

1. Introduction

Models which incorporate time steps and additional auxiliary data (e.g., land-use data) or are coupled with other models 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 TL;DR – 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 perhaps 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 and in 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. The `teems-solver` is written in C and uses highly performant Fortran libraries.

2.1. Data

The only data currently `teems`-compatible is available from the Global Trade Analysis Project (GTAP). Although the most recent databases are proprietary, GTAP has consistently made databases available two versions behind the current release (GTAP 10).

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 through 12.

2.2. R

R ($\geq 4.3.0$) 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()`. The 49th Parallel theme with Fira Code font makes for an aesthetically pleasing coding experience.

2.3. teems-R package

The R package is available on CRAN.

```
# install package
install.packages("teems")
```

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.

Two prerequisites:

- **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.

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 all dependencies 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 ($\geq 4.3.0$) 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",
  path = "~/my_project",
  type = "scripts",
  dat_input = "v6_data/gsddat.har",
  par_input = "v6_data/gsdpar.har",
  set_input = "v6_data/gsdset.har"
)
```

`paths` is a character vector of the written script file paths. Each script is self-contained and immediately runnable.

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. 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.

3. Quickstart

```
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
```

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 tibble where each row corresponds to a solved variable or coefficient. The `dat` list column holds the result data for that row in `data.table` format.

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

See the Composing results 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...
<code>part_uniform.R</code>	You want to shock a single region or sector
<code>part_uniform_part_swap.R</code>	You need to change the closure for part of the model
<code>custom_full.R</code>	You have heterogeneous shock values from external data
<code>full_uniform.R</code> 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.

4. Example files

4.1. Overview

`ems_example()` writes bundled example model and script files to disk. It is a good starting point for new users, providing ready-to-run material for each of the four vetted models 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 a 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.

4.2. Arguments

Argument	Default	Description
<code>model</code>		Model name. One of "GTAPv6", "GTAPv7", "GTAP-INT", "GTAP-RE"
<code>path</code>		Directory where files will be written
<code>type</code>	"model_files"	Output type: "model_files" or "scripts"
<code>dat_input</code>	NULL	Path to the basedata HAR file, or the <code>dat</code> list from <code>GTAP_convert()</code> . Required when <code>type = "scripts"</code>
<code>par_input</code>	NULL	Path to the parameters HAR file, or the <code>par</code> list from <code>GTAP_convert()</code> . Required when <code>type = "scripts"</code>
<code>set_input</code>	NULL	Path to the sets HAR file, or the <code>set</code> list from <code>GTAP_convert()</code> . Required when <code>type = "scripts"</code>

4.3. 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()`.

4. Example files

```
library(teems)

# Write GTAPv7 model files to a temporary directory
paths <- ems_example("GTAPv7", tempdir())

# Or to a specific directory
paths <- ems_example("GTAP-RE", "~/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"]]
)
```

4.4. Writing example scripts

When `type = "scripts"`, `ems_example()` writes a directory of R scripts covering a range of configurations available for the chosen model. Each script is injected with the supplied data paths and is immediately runnable.

```
paths <- ems_example(
  model = "GTAPv7",
  path = "~/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"
)
```

The data inputs can also be the in-memory list objects returned by `GTAP_convert()`, which is useful when the raw data needs format conversion before running:

```
# Convert v7.0 data to v6.2 format, then generate scripts for the classic model
converted <- GTAP_convert(
  dat_har = "~/dat/GTAP/v12/gsdldat.har",
  par_har = "~/dat/GTAP/v12/gsdldpar.har",
  set_har = "~/dat/GTAP/v12/gsdldset.har",
  target = "GTAPv6"
```

```
)

paths <- ems_example(
  model = "GTAPv6",
  path = "~/my_project",
  type = "scripts",
  dat_input = converted$dat,
  par_input = converted$par,
  set_input = converted$set
)
```

When list objects are supplied, `ems_example()` writes them to RDS files alongside the scripts and injects `readRDS()` calls at the top of each script in place of plain path assignments.

4.5. 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.

4.5.1. Script naming convention

Script file names encode the shock and swap configuration:

File name	What it demonstrates
<code>null.R</code>	Zero-shock run — useful for verifying the model solves before applying shocks
<code>numeraire.R</code>	Uniform 1% population shock — minimal working example
<code>full_uniform.R</code>	Uniform shock across all elements of a variable
<code>part_uniform.R</code>	Uniform shock on a subset of elements
<code>part_uniform_full_swap.R</code>	Partial shock plus a full-variable closure swap
<code>part_uniform_part_swap.R</code>	Partial shock plus a partial closure swap
<code>part_uniform_part_swap_mixed.R</code>	Partial shock plus mixed (full + partial) swaps
<code>part_uniform_subset_swap.R</code>	Partial shock and swaps with mixed multiple elements and subsets
<code>custom_full.R</code>	Heterogeneous shock values supplied as a data frame
<code>custom_partial.R</code>	Heterogeneous shock values supplied as a data frame on a subset of elements
<code>custom_full_csv.R</code>	Heterogeneous shock values read from a CSV file

4.5.2. 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:

4. Example files

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

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

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

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

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

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

4.6. 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.

Part II.

Data and model inputs

5. Loading and specifying sets

5.1. Overview

Set mappings aggregate the granular regions, sectors, and endowments in the GTAP database to the resolution desired 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.

5.2. 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`.

5.2.1. 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	Description
<code>big3</code>	China (<code>chn</code>), United States (<code>usa</code>), Rest of World (<code>row</code>)

5. Loading and specifying sets

Name	Description
AR5	IPCC Fifth Assessment Report regions: asia, eit, lam, maf, oecd
WB7	World Bank 7-region classification
WB23	World Bank 23-region classification
R32	IIASA-based SSP 32-region mapping
medium	Custom medium-resolution aggregation
large	Custom large-resolution aggregation
huge	Near country-level with minor groupings
full	Full (unaggregated)

5.2.2. Sector (PROD_COMM, ACTS)

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

Name	Description
macro_sector	Broad evenly weighted aggregation: crops, food, livestock, mnfcs, svces
food	Food-focused: disaggregated food and agriculture; aggregated manufacturing and services
energy	Energy-focused: disaggregated energy sectors
manufacturing	Manufacturing-focused: disaggregated manufacturing sectors
services	Services-focused: disaggregated services sectors
medium	Custom medium-resolution aggregation
full	Full (unaggregated)

5.2.3. Endowment (ENDW_COMM, ENDW)

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

Name	Description
labor_agg	Aggregated labor: capital, labor, land, natlres
labor_diff	Differentiated labor: capital, land, natlres, sklab, unsklab
full	Full (unaggregated)

5.3. 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 = "v7_data/gsdmdat.har",
  par_input = "v7_data/gsdmpar.har",
  set_input = "v7_data/gsdmsset.har",
  REG = "path/to/custom_REG_mapping.csv", # external mapping
  ACTS = "macro_sector",
  ENDW = "labor_agg"
)
```

5.4. Set naming convention

Set names in *any loaded data* (e.g., `ems_model()`) 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.

6. Loading data

6.1. 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 is also used to specify any time steps (for temporally dynamic models) and load set mappings for sets that are read into the model (i.e., not constructed from set operations).

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

Argument	Type	Description
<code>dat_input</code>	Character or list	Path to a HAR file containing basedata coefficient data, or a <code>dat</code> list from <code>[GTAP_convert()]</code>
<code>par_input</code>	Character or list	Path to a HAR file containing parameter coefficient data, or a <code>par</code> list from <code>[GTAP_convert()]</code>
<code>set_input</code>	Character or list	Path to a HAR file containing set elements and attributes, or a <code>set</code> list from <code>[GTAP_convert()]</code>
<code>time_steps</code>	Integer vector or NULL	Time steps for intertemporal models (see Time steps)
<code>...</code>	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.

6.2. 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 made databases publicly available two versions behind the current release (GTAP 10 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

6. Loading data

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.

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 (“v7_data/gsdmdat.har” placeholder).

```
v7_data <- ems_data(dat_input = "v7_data/gsdmdat.har", # basedata coefficients
                    par_input = "v7_data/gsdmpar.har", # parameter coefficients
                    set_input = "v7_data/gsdmsset.har", # set elements
                    REG = "big3",
                    ACTS = "macro_sector",
                    ENDW = "labor_diff"
)
```

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

Input Type	GTAP v9	GTAP v10	GTAP v11	GTAP v12
dat_input	gddat.har	gsddat.har	gsdmdat.har	gsdmdat.har
par_input	gdpar.har	gsdpar.har	gsdmpar.har	gsdmpar.har
set_input	gdset.har	gsdset.har	gsdmsset.har	gsdmsset.har

6.3. 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.

6.4. Time steps

For temporally dynamic models (e.g., GTAP-INT, GTAP-RE), time steps must be provided representing t_0 plus actual year steps from t_0 . Time steps can be inputted in either actual year increments or represented as chronological years. Note that if chronological years are used, t_0 must correspond with the reference year of the database being used.

The following uses explicit time steps (steps from t_0), equivalent to chronological years `c(2014, 2015, 2016, 2017, 2018, 2020, 2022, 2024, 2026, 2028, 2030)` given the 2014 reference year of the loaded data.

```
data <- ems_data(dat_input = "v6_data/gsddat.har",
                 par_input = "v6_data/gsdpar.har",
                 set_input = "v6_data/gsdset.har",
                 time_steps = c(0, 1, 2, 3, 4, 6, 8, 10, 12, 14, 16),
                 REG = "WB23",
```

6. Loading data

```
    PROD_COMM = "services",  
    ENDW_COMM = "labor_agg"  
)
```

Chronological time steps (note reference year of input data).

```
data <- ems_data(dat_input = "v7_data/gsdmdat.har",  
               par_input = "v7_data/gsdmpar.har",  
               set_input = "v7_data/gsdmsset.har",  
               time_steps = c(2017, 2018, 2020, 2022, 2024, 2026, 2028, 2030),  
               REG = "R32",  
               ACTS = "medium",  
               ENDW = "labor_diff"  
)
```

6.5. Converting formats

`GTAP_convert()` reads GTAP HAR files into R and optionally converts between the classic v6.2 and standard v7.0 formats. The returned list can be passed directly to `ems_data()` or `ems_example()`. Note that set mappings must be specified according to the model to be used, *not* the original format of the input data.

6.5.1. Loading HAR files as arrays

When `target` is omitted, `GTAP_convert()` simply parses the HAR files into lists of arrays without any format conversion:

```
dat <- GTAP_convert(  
  dat_har = "v7_data/gsdmdat.har",  
  par_har = "v7_data/gsdmpar.har",  
  set_har = "v7_data/gsdmsset.har"  
)  
  
v7_data <- ems_data(dat_input = dat$dat,  
                  par_input = dat$par,  
                  set_input = dat$set,  
                  REG = "big3",  
                  ACTS = "macro_sector",  
                  ENDW = "labor_diff"  
)
```

6.5.2. Converting between formats

To use a classic-based model with a newer v7.0 database, or a standard model with an older v6.2 database, specify `target`:

GTAP 12 v7.0 database converted to v6.2 format for use with a classic GTAP model:

```
converted <- GTAP_convert(
  dat_har = "v7_data/gsdmdat.har",
  par_har = "v7_data/gsdmpar.har",
  set_har = "v7_data/gsdmsset.har",
  target = "GTAPv6"
)

v6_data <- ems_data(dat_input = converted$dat,
  par_input = converted$par,
  set_input = converted$set,
  REG = "big3",
  PROD_COMM = "macro_sector",
  ENDW_COMM = "labor_agg"
)
```

GTAP 10 v6.2 database converted to v7.0 format for use with the standard GTAP model:

```
converted <- GTAP_convert(
  dat_har = "v6_data/gsddat.har",
  par_har = "v6_data/gsdpar.har",
  set_har = "v6_data/gsdset.har",
  target = "GTAPv7"
)

v7_data <- ems_data(dat_input = converted$dat,
  par_input = converted$par,
  set_input = converted$set,
  REG = "AR5",
  ACTS = "macro_sector",
  ENDW = "labor_agg"
)
```

6.6. 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 directly loaded in their final format into the `ems_model()` function, and no aggregation or modification will take place. In the below notation, hat (^) indicates the new parameter value and asterisk indicates the destination set mapping.

6.6.1. 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.

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

6.6.1.1. Aggregation methods

$$\hat{\sigma}_{d_i} = \frac{\sum_i^{i^*} (\sigma_{d_i} \sum_r (vdpa_{i,r} + vdga_{i,r} + vdfa_{i,r} + vipa_{i,r} + viga_{i,r} + vifa_{i,r}))}{\sum_{i,r}^{i^*} (vdpa_{i,r} + vdga_{i,r} + vdfa_{i,r} + vipa_{i,r} + viga_{i,r} + vifa_{i,r})} \quad (6.1)$$

$$\hat{\sigma}_{m_i} = \frac{\sum_i^{i^*} (\sigma_{m_i} \sum_r (vipa_{i,r} + viga_{i,r} + vifa_{i,r}))}{\sum_{i,r}^{i^*} (vipa_{i,r} + viga_{i,r} + vifa_{i,r})} \quad (6.2)$$

$$\hat{\sigma}_{va_j} = \begin{cases} \frac{\sum_j^{j^*} (\sigma_{va_j} \sum_r evfa_{j,r})}{\sum_{j,r}^{j^*} evfa_{j,r}} & \text{if } j \neq cgds \\ \sigma_{va_j} & \text{if } j = cgds \end{cases} \quad (6.3)$$

6. Loading data

$$\hat{\gamma}_{i,r} = \frac{\sum_{i,r}^{i^*r^*} \gamma_{i,r} \sum_{i,r} (vdpa_{i,r} + vipa_{i,r})}{\sum_{i,r}^{i^*r^*} (vdpa_{i,r} + vipa_{i,r})} \quad (6.4)$$

$$\hat{\beta}_{i,r} = \frac{\sum_{i,r}^{i^*r^*} (\beta_{i,r} \sum_{i,r} (vdpa_{i,r} + vipa_{i,r}))}{\sum_{i,r}^{i^*r^*} (vdpa_{i,r} + vipa_{i,r})} \quad (6.5)$$

6.6.2. 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	$c \in \text{COMM}$
σ_m	ESBM	Armington CES for regional allocation of imports	$c \in \text{COMM}$
σ_{va}	ESBV	CES between primary factors in production	$a \in \text{ACTS}, r \in \text{REG}$
γ	INCP	CDE expansion parameter	$c \in \text{COMM}, r \in \text{REG}$
β	SUBP	CDE substitution parameter	$c \in \text{COMM}, r \in \text{REG}$
Data			
	EVFP	Primary factor purchases at purchasers' prices	$e \in \text{ENDW}, a \in \text{ACTS}, r \in \text{REG}$
	VDFP	Domestic purchases by firms at purchasers' prices	$c \in \text{COMM}, a \in \text{ACTS}, r \in \text{REG}$
	VDGP	Domestic purchases by government at purchasers' prices	$c \in \text{COMM}, r \in \text{REG}$
	VDPP	Domestic purchases by households at purchasers' prices	$c \in \text{COMM}, r \in \text{REG}$
	VMFP	Import purchases by firms at purchasers' prices	$c \in \text{COMM}, a \in \text{ACTS}, r \in \text{REG}$
	VMGP	Import purchases by government at purchasers' prices	$c \in \text{COMM}, r \in \text{REG}$
	VMPP	Import purchases by households at purchasers' prices	$c \in \text{COMM}, r \in \text{REG}$

6.6.2.1. Aggregation methods

$$\hat{\sigma}_{d_c} = \frac{\sum_c^{c^*} (\sigma_{d_c} \sum_r (vdpp_{c,r} + vmpp_{c,r} + vdgp_{c,r} + vmgp_{c,r} + vdfp_{c,r} + vmfp_{c,r}))}{\sum_{c,r}^{c^*} (vdpp_{c,r} + vmpp_{c,r} + vdgp_{c,r} + vmgp_{c,r} + vdfp_{c,r} + vmfp_{c,r})} \quad (6.6)$$

6. Loading data

$$\hat{\sigma}_{m_c} = \frac{\sum_c^{c^*} (\sigma_{m_c} \sum_r (vmpp_{c,r} + vmgp_{c,r} + vmfp_{c,r}))}{\sum_{c,r}^{c^*} (vmpp_{c,r} + vmgp_{c,r} + vmfp_{c,r})} \quad (6.7)$$

$$\hat{\sigma}_{va_a} = \frac{\sum_a^{a^*} (\sigma_{va_a} \sum_r evfp_{a,r})}{\sum_{a,r}^{a^*} evfp_{a,r}} \quad (6.8)$$

$$\hat{\gamma}_{c,r} = \frac{\sum_{c,r}^{c^*r^*} \gamma_{c,r} \sum_{c,r} (vdpp_{c,r} + vmpp_{c,r})}{\sum_{c,r}^{c^*r^*} (vdpp_{c,r} + vmpp_{c,r})} \quad (6.9)$$

$$\hat{\beta}_{c,r} = \frac{\sum_{c,r}^{c^*r^*} (\beta_{c,r} \sum_{c,r} (vdpp_{c,r} + vmpp_{c,r}))}{\sum_{c,r}^{c^*r^*} (vdpp_{c,r} + vmpp_{c,r})} \quad (6.10)$$

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

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

7.2. The model file

Model input in `teems` is currently limited to Tablo (`.tab`) files. Vetted models and their closures are available at `teems-models` and can also be accessed via the `ems_example()` function.

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.

7.3. Closure selection

Standard model-specific closures are available within the model repository: `teems-models`. The general format consists of an `exogenous` block listing all (`\n` delimited) exogenous variables, followed by `;`, `rest endogenous`, and `;` to mark the end:

```
exogenous
pop
aoreg("row")
qe(ENDWC,REG,INITIME)
txs(MARG,"row",REG,FWDTIME)
...
;
rest endogenous
;
```

Each entry is either a full variable name (`pop`), an entry with specific set elements (`aoreg("row")`), an entry with subsets (`qe(ENDWC,REG,INITIME)`) or a mixed entry (`txs(MARG,"chn",REG,FWDTIME)`).

7.4. Incompatible features

The `teems-solver` was originally designed to emulate GEMPack, 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 below.

7.4.1. 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).

7.4.2. 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 are 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);
```

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

7.4.3. 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)};
```

7.5. Variable omissions

Variables specified with `var_omit` are set to zero in the model file and removed from the closure. 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")`.

7.6. 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. All data inputted here must be model-aggregation specific and will not be subject to aggregation. The left-hand side (name) must correspond to a coefficient (not header) declared within the model input. Data frame and CSV inputs are validated against the set aggregations from `ems_data()`. Dimensions must match the aggregated resolution, not the original database resolution. Three input types are supported, with model “Read” and “Formula” declarations modified accordingly:

7.6.1. Uniform numeric value

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

```
GTAP_RE <- ems_example("GTAP-RE")
model <- ems_model(
  model_file = GTAP_RE[["model_file"]],
  closure_file = GTAP_RE[["closure_file"]],
  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.

7.6.2. 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)
GTAP_RE <- ems_example("GTAP-RE")
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 = GTAP_RE[["model_file"]],
  closure_file = GTAP_RE[["closure_file"]],
  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 = GTAP_RE[["model_file"]],
  closure_file = GTAP_RE[["closure_file"]],
  CPHI = CPHI
)
```

7.6.3. 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 = GTAP_RE[["model_file"]],
  closure_file = GTAP_RE[["closure_file"]],
  SUBPAR = "SUBPAR.csv"
)
```

7.6.4. Combining injections

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

```
model <- ems_model(
  model_file = GTAP_RE[["model_file"]],
  closure_file = GTAP_RE[["closure_file"]],
  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.

7.6.5. 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.

7.7. Intertemporal models

For intertemporal models, `ems_model()` determines temporal dynamics by detecting intertemporal sets 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")
```

7. Loading models

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()
```

Part III.

Swaps and shocks

8. Closure swaps

8.1. Overview

Swaps allow the user to reconfigure which variables are exogenous (provided by the modeler) and which are endogenous (solved by the model). 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.

8.2. Simple (full variable) swaps

Here, simple swaps refer to 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 = dat,  
  model = model,  
  swap_in = "qfd",  
  swap_out = "tfd"  
)
```

Multiple full variable swaps:

```
cmf_path <- ems_deploy(  
  .data = dat,  
  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”.

8.3. Complex (partial variable) swaps

8.3.1. `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
<code>var</code>	Character	Variable name as it appears in the model file
<code>...</code>	Named arguments	Set-index pairs restricting the swap. Each value is a single element (" <code>element</code> "), multiple elements (<code>c("a", "b")</code>), or a named subset (" <code>SUBSET</code> ")

Partial variable swaps entail swapping parts of a variable (i.e., subsets of the constitutive 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” variables become endogenous. Within the GTAPv7 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”:

8. Closure swaps

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

While the subindices for both variables are the same within GTAPv7, note that these may vary in some models such as GTAPv6 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 = dat,  
  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 = dat,  
  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"
)
```

8.3.2. Subset swaps

A named model subset may be passed in place of individual elements. `teems` expands the subset to all its elements at the aggregation level specified in `ems_data()`. This is useful when a model defines meaningful groupings that map directly to closure logic such as margin commodities or forward time steps in intertemporal models.

In the GTAP-RE model, `MARG` is a subset of `COMM` containing margin commodities, and `FWDTIME` is a subset of `ALLTIME` containing forward time steps. Passing these subset names directly restricts the swap to only those elements:

```
qxs_in <- ems_swap(
  var = "qxs",
  COMMc = "MARG",
  REGs = c("chn", "usa"),
  ALLTIMEt = "FWDTIME"
)

txs_out <- ems_swap(
  var = "txs",
  COMMc = "MARG",
  REGs = c("chn", "usa"),
  ALLTIMEt = "FWDTIME"
)

cmf_path <- ems_deploy(
  .data = dat,
  model = model,
  shock = partial,
  swap_in = qxs_in,
  swap_out = txs_out
)
```

Note that `REGs = c("chn", "usa")` is a multi-element specification, while `COMMc = "MARG"` and `ALLTIMEt = "FWDTIME"` are subset specifications. Both the above forms can be combined freely in a single `ems_swap()` call.

9. Uniform shocks

9.1. 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
<code>var</code>	Character	Variable name as it appears in the model file
<code>value</code>	Numeric	Percentage change applied to all targeted elements
<code>...</code>	Named arguments	Set-index pairs restricting the shock. Each value is a single element (" <code>element</code> "), multiple elements (<code>c("a", "b")</code>), or a named subset (" <code>SUBSET</code> ")

9.2. Full variable uniform shocks

The simplest 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 be increased by 1%.

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

9.3. Partial variable uniform shocks

As with the syntax regarding swaps, 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:

9. Uniform shocks

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

If a full swap is followed by a partial variable shock, the exogenous 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",
  value = -1,
  REGr = "chn",
  ACTSa = "crops",
  Year = 2017
)
```

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.

9.4. Subset shocks

A named model subset may be passed in place of individual elements, identical to the subset form in `ems_swap()`. `teems` expands the subset to all its elements at the aggregation level specified in `ems_data()`.

In the GTAP-RE model, `MARG` is a subset of `COMM` containing margin commodities, and `FWDTIME` is a subset of `ALLTIME` containing forward time steps. Using these subset names directly restricts the shock to only those elements:

```
partial <- ems_uniform_shock(
  var = "qxs",
  value = -1,
  COMMc = "MARG",
  REGs = c("chn", "usa"),
  ALLTIMEt = "FWDTIME"
)
```

As in the swaps example, `REGs = c("chn", "usa")` is a multi-element specification while `COMMc = "MARG"` and `ALLTIMEt = "FWDTIME"` are subset specifications. This shock would then be deployed alongside the corresponding subset swaps:

```
cmf_path <- ems_deploy(
  .data = dat,
  model = model,
  shock = partial,
  swap_in = qxs_in,
  swap_out = txs_out
)
```

9.5. 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",
  value = -1,
  REGs = "usa",
  PROD_COMMj = "crops"
)

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 = dat,
  model = model,
  shock = list(partial, full),
  swap_in = list(qfd, "yp"),
  swap_out = list(tfd, "dppriv")
)
```

10. Custom shocks

10.1. 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
<code>var</code>	Character	Variable name as it appears in the model file
<code>input</code>	Data frame or Character	Data frame or path to a CSV file containing set columns and a <code>Value</code> column (percentage changes)

10.2. 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 = dat,
  model = model,
  shock = list(pop_shk, aoall_shk, afeall_shk, atall_shk)
)

```

10.3. CSV input

A path to a CSV file can be passed to `input` instead of a data frame. The CSV must follow the same structure: one column per set index using the model-specific naming convention and a `Value` column. Column order is inconsequential.

```

pop_csv <- tempfile(fileext = ".csv")
write.csv(pop, pop_csv, row.names = FALSE)

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

```

10.4. 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(
  ACTSa = ACTS,
  REGr = REG,
  ALLTIMEt = ALLTIME,
  stringsAsFactors = FALSE
)
aoall_full$Value <- 0
aoall_full <- aoall_full[do.call(order, aoall_full), ]
aoall <- aoall_full[
  !(aoall_full$ACTSa == "crops" & aoall_full$REGr == "usa") &
  aoall_full$ALLTIMEt != 3,
]
aoall$Value <- runif(nrow(aoall))
aoall_shk <- ems_custom_shock(var = "aoall", input = aoall)
```

10.5. Chronological year input

For intertemporal models, the time dimension can be specified using chronological years instead of integer time steps by using a `Year` column in place of `ALLTIMEt`. The year values must correspond to the reference year and steps defined in `ems_data()`.

```
time_steps <- c(2017, 2018, 2019, 2020)

dat <- ems_data(
  dat_input = dat_input,
  par_input = par_input,
  set_input = set_input,
  time_steps = time_steps,
  REG = "big3",
  ACTS = "macro_sector",
  ENDW = "labor_agg"
)
```

Construct the input data frame with a `Year` column rather than `ALLTIMEt`:

```
REG <- c("chn", "usa", "row")

pop <- expand.grid(
  REGr = REG,
```

```
    Year = time_steps,  
    stringsAsFactors = FALSE  
  )  
  
pop <- pop[do.call(order, pop), ]  
pop$Value <- runif(nrow(pop))  
  
pop_shk <- ems_custom_shock(  
  var = "pop",  
  input = pop  
)
```

With `ems_custom_shock()`, a `Year` column is recognized automatically and mapped to the corresponding integer time step internally.

11. Scenario shocks

11.1. 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 must be in the same units as the underlying database coefficient and 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
<code>var</code>	Character	Variable name as it appears in the model file
<code>input</code>	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)

11.2. 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 (no partial variable scenario shock)
3. Inputs are denominated in absolute values in the units of the underlying database coefficient (e.g., million \$USD, population)
4. 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`.

11.3. 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.

Set up the model run with chronological time steps:

```
year <- 2017
time_steps <- year + c(0, 1, 2, 3)

dat <- ems_data(
  dat_input = "v7_data/gsdmdat.har",
  par_input = "v7_data/gsdmpar.har",
  set_input = "v7_data/gsdmsset.har",
  time_steps = time_steps,
  REG = "big3",
  ACTS = "macro_sector",
  ENDW = "labor_agg"
)

GTAP_RE <- ems_example("GTAP-RE")
model <- ems_model(
  model_file = GTAP_RE[["model_file"]],
  closure_file = GTAP_RE[["closure_file"]]
)
```

Load POP at full regional resolution (unaggregated) as the scenario baseline. The `REG = "full"` call does not require `time_steps` since it is only used to extract the base-year coefficient:

```
pop <- ems_data(
  dat_input = "v7_data/gsdmdat.har",
  par_input = "v7_data/gsdmpar.har",
  set_input = "v7_data/gsdmsset.har",
  REG = "full",
  ACTS = "macro_sector",
  ENDW = "labor_agg"
)$POP
```

Construct trajectories with compound growth through each future time step. `tail(time_steps, -1)` yields the forward time steps, keeping the trajectory in sync with whatever `time_steps` was passed to `ems_data()`:

11. Scenario shocks

```
pop$Year <- year
regions <- unique(pop$REG)

pop_traj <- expand.grid(REG = regions, Year = tail(time_steps, -1),
                      stringsAsFactors = FALSE)
pop_traj$Value <- 0
pop <- rbind(pop, pop_traj)

set.seed(42)
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 == year, c("REG", "Value")]
names(base_values)[2] <- "base_value"
pop <- merge(pop, base_values, by = "REG")

pop$Value[pop$Year > year] <-
  pop$base_value[pop$Year > year] *
  (1 + pop$growth_rate[pop$Year > year])^(pop$Year[pop$Year > year] - year)

pop$growth_rate <- NULL
pop$base_value <- NULL
pop <- pop[order(pop$REG, pop$Year), c("REG", "Year", "Value")]
```

`ems_data()` returns coefficients with plain set names (e.g., REG). Rename to the model's set-index convention before passing to `ems_scenario_shock()`:

```
colnames(pop)[colnames(pop) == "REG"] <- "REGr"
```

Pass `pop` as the scenario shock and deploy:

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

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

`pop` may also be saved as a CSV and loaded directly by passing the file path to `input`.

Part IV.

Deploying and solving the model

12. Deploying the model

12.1. Overview

`ems_deploy()` validates all 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()`.

12.2. Arguments

Argument	Default	Description
<code>.data</code>		A list of data.tables generated by <code>ems_data()</code>
<code>model</code>		A tibble generated by <code>ems_model()</code>
<code>shock</code>	NULL	A shock object or list of shock objects produced by <code>ems_uniform_shock()</code> , <code>ems_custom_shock()</code> , or <code>ems_scenario_shock()</code> . When NULL, no shock is applied and input data is returned without change
<code>swap_in</code>	NULL	A character vector, list generated by <code>ems_swap()</code> , or combination provided as a list. Variables to be made exogenous
<code>swap_out</code>	NULL	A character vector, list generated by <code>ems_swap()</code> , or combination provided as a list. Variables to be made endogenous
<code>shock_file</code>	NULL	Path to a file with <code>.shf</code> extension representing a fully prepared shock file. No checks or modifications are carried out on this file

12.3. Basic usage

The simplest deployment requires only data and model inputs. With no shock supplied, input data is returned without change:

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

12.4. 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 = dat,
  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 = dat,
  model = model,
  shock = list(pop_shk, aoall_shk)
)
```

12.5. Closure swaps

A single full variable swap swaps one variable in and one out using plain strings:

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

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

Multiple full variable swaps are passed as character vectors. Partial swaps use `ems_swap()` objects and can be mixed with strings in a list:

```

# Multiple full variable swaps
cmf_path <- ems_deploy(
  .data = dat,
  model = model,
  shock = shock,
  swap_in = c("qfd", "yp"),
  swap_out = c("tfd", "dppriv")
)

# Mixed partial and full swaps
yp_row <- ems_swap("yp", REGr = "row")
dppriv_row <- ems_swap("dppriv", REGr = "row")

cmf_path <- ems_deploy(
  .data = dat,
  model = model,
  shock = shock,
  swap_in = list(yp_row, "qfd"),
  swap_out = list(dppriv_row, "tfd")
)

```

12.6. 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. Solving the model

13.1. 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()`.

13.2. Arguments

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

Argument	Default	Description
<code>append_args</code>	NULL	Character vector of additional arguments appended to the Docker run command (e.g., <code>c("-smlthreads 2", "-maxthreads 2")</code>)

13.3. Basic usage

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

13.4. Solution methods

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

13.5. Matrix methods

Increase `n_tasks` to take advantage of parallel matrix solution methods.

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

13.6. Multi-step solution

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

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

13.7. 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,
  solution_method = "Johansen",
  matrix_method = "LU",
  laA = 500L
)
```

13.8. Parallel solving

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

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

13.9. 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,
  solution_method = "Johansen",
  matrix_method = "LU",
  terminal_run = TRUE
)

```

13.10. Solving in-situ

`solve_in_situ()` calls the teems-solver directly with user-supplied input files, bypassing the `ems_data()` / `ems_model()` / `ems_deploy()` pipeline. The model file will be parsed but all input files must be provided in their final form. No checks or modifications are applied to input files.

This function is intended for users who wish to use the teems solver with pre-prepared input files, or in combination with the standard pipeline by reusing the files written to disk by `ems_deploy()`. If all input files already reside in `model_dir`, no copying is performed; otherwise files are copied into `model_dir` before solving.

13.10.1. 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
<code>model_dir</code>		Base directory where input files will be copied if not already present, and model outputs will be written
<code>model_file</code>		Path to the model Tablo (.tab) file
<code>closure_file</code>		Path to the closure (.cls) file
<code>shock_file</code>		Path to the shock (.shf) file
<code>solution_method</code>	"Johansen"	Solution method: "Johansen" or "mod_midpoint"
<code>matrix_method</code>	"LU"	Matrix solution method: "LU", "DBBD", "SBBBD", or "NDBBD"
<code>n_subintervals</code>	1L	Number of subintervals for the applied shock
<code>steps</code>	c(2L, 4L, 8L)	Vector of steps for the modified midpoint method
<code>n_tasks</code>	1L	Number of tasks to run in parallel. Must be 1L if <code>matrix_method</code> is "LU"
<code>laA</code>	300L	Memory parameter (%) for "LU" and "SBBBD" methods

Argument	Default	Description
laD	200L	Memory parameter (%) for "DBBD" and "NDBBD" methods
laDi	500L	Memory parameter (%) for "NDBBD" method
suppress_outputs	FALSE	When TRUE, returns the CMF file path instead of parsed outputs
terminal_run	FALSE	When TRUE, exits without running the solver
append_args	NULL	Character vector of additional arguments appended to the Docker run command (e.g., <code>c("-smllthreads 2", "-maxthreads 2")</code>)

13.10.2. 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 GTAPPARAM # parameter file;
File GTAPSETS # set file;
```

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

```
solve_in_situ(
  GTAPDATA = file.path(write_dir, "teems", "GTAPDATA.txt"),
  GTAPINT  = file.path(write_dir, "teems", "GTAPINT.txt"),
  GTAPPARAM = file.path(write_dir, "teems", "GTAPPARAM.txt"),
  GTAPSETS = file.path(write_dir, "teems", "GTAPSETS.txt"),
  model_dir = "~/in_situ_run/",
  model_file = "GTAP-RE.tab",
  closure_file = "GTAP-RE.cls",
  shock_file = "a_shock_file.shf",
  solution_method = "mod_midpoint",
  matrix_method = "SBBD",
  n_subintervals = 1L,
  n_tasks = 1L
)
```

13.10.3. Output modes

By default, `solve_in_situ()` returns a tibble containing model output variables and coefficients. If `suppress_outputs = TRUE`, the function returns the CMF file path instead, which can be passed to `ems_compose()` for manual parsing.

14. Composing results

14.1. Overview

`ems_compose()` retrieves and processes results from a solved model run. Results are parsed and filtered according to the `which` argument. 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.

14.2. Arguments

Argument	Default	Description
<code>cmf_path</code>		Path to the CMF file generated by <code>ems_deploy()</code>
<code>which</code>	"all"	"all" returns all model variables and all coefficients (if coefficient output files are present). Any other value is treated as a character vector of variable and/or coefficient names to retrieve

14.3. Basic usage

After a standard model run with suppressed outputs:

```
ems_solve(  
  cmf_path = cmf_path,  
  solution_method = "Johansen",  
  matrix_method = "LU",  
  suppress_outputs = TRUE  
)  
  
# Parse all results  
outputs <- ems_compose(cmf_path = cmf_path)
```

14.4. Filtering output

Pass a character vector of names to `which` to retrieve a subset of results. Variables and coefficients can be mixed freely in a single call:

```
# All model variables and coefficients (default)
all_outputs <- ems_compose(cmf_path = cmf_path)

# Single variable by name
qfd <- ems_compose(cmf_path = cmf_path, which = "qfd")

# Single coefficient by name
SAVE <- ems_compose(cmf_path = cmf_path, which = "SAVE")

# Multiple names with mixed variables and coefficients
results <- ems_compose(cmf_path = cmf_path, which = c("qfd", "qfm", "SAVE"))
```

Note that some coefficient names differ from their associated HAR headers. For example, the coefficient exposed as "VTMFSD" corresponds to the VTWR header. Use the name as it appears in the model file, not the header.

14.5. Return value

`ems_compose()` returns a tibble with one row per result and four columns:

Column	Description
name	Variable or coefficient name as it appears in the model file
label	Description string from the model file
type	"variable" or "coefficient"
dat	List-column of data.tables, one per row, containing the result values

Part V.

Configuration

15. Package options

15.1. Overview

teems provides a set of advanced options that control package behavior across function calls. For example, teems is rather chatty and users may wish to silence all innocuous messages with `ems_option_set(verbose = FALSE)`. 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.

15.2. Retrieving options

View all current options:

```
ems_option_get()
```

Retrieve a specific option by name. An error is raised if the name is not a valid option:

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

15.3. 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)
```

15.4. Resetting options

Reset all options to their default values:

```
ems_option_reset()
```

15.5. Available options

Option	Default	Description
<code>verbose</code>	<code>TRUE</code>	If <code>FALSE</code> , function-specific diagnostics are silenced
<code>tempdir</code>	<code>tempdir()</code>	Directory used for temporary file storage during a model run
<code>ndigits</code>	6	Exact number of digits to the right of the decimal point written to file for numeric type double. Passed to <code>format()</code> <code>nsmall</code> and <code>round()</code> <code>digits</code>
<code>accuracy_threshold</code>	0.8	Threshold (converted to a percentage) against which 4-digit precision is compared. A warning is generated if the threshold is not met
<code>check_shock_status</code>	<code>TRUE</code>	If <code>FALSE</code> , no check on shock element endogenous/exogenous status is conducted
<code>timestep_header</code>	"YEAR"	Coefficient containing a numeric vector of timestep intervals. For novel intertemporal models — modify with caution
<code>n_timestep_header</code>	"NTSP"	Coefficient containing a numeric vector length one with sum of timestep intervals. For novel intertemporal models — modify with caution
<code>full_exclude</code>	<code>c("DREL", "DVER", "XXCR", "XXCD", "XXCP", "SLUG", "EFLG")</code>	Headers to fully exclude from all aspects of the model run. Failure to designate these headers properly will result in errors. Modify with caution
<code>docker_tag</code>	"latest"	Docker tag specifying which Docker image to use