

Package: GEModelR (via r-universe)

October 27, 2026

Type Package

Title TABLO Models in R with Legacy and Sparse Solvers

Version 0.1.0

Depends R (>= 3.5.0)

Author David Zenz [aut, cre, cph], Maros Ivanic [aut] (Author of the reviewed public-domain/CC0 upstream tabloToR baseline)

Maintainer David Zenz <zenz@wiiw.ac.at>

URL <https://github.com/DavidZenz/GEModelR>

Config/Documentation/Status pending

BugReports <https://github.com/DavidZenz/GEModelR/issues>

Description Interpret GEMPACK-style TABLO models in R. The package keeps the original in-memory solver and adds an opt-in integer-indexed sparse execution engine for low-memory simulations.

License GPL (>= 3)

Encoding UTF-8

Imports Matrix (>= 1.6-5), Rcpp, SparseM, methods

LinkingTo Rcpp

Suggests knitr, pkgdown, rmarkdown, testthat (>= 3.2.0), withr

Config/testthat/edition 3

VignetteBuilder knitr

Config/Attribution-People David Zenz; Maros Ivanic

Config/Attribution-Record docs/provenance/ATTRIBUTION.md

Config/Attribution-Evidence R/GEModel.R::GEModel\$loadTablo;
R/GEModel.R::GEModel\$solveModel;
R/processTablo.R::processTablo;
R/sparseCompiler.R::sparse_compile_spec;
R/sparseSolver.R::sparse_solve_model;
src/sparse-schur.cpp::GEModelR_schur_accumulate_global

RoxygenNote 7.3.3
Repository <https://davidzenz.r-universe.dev>
Date/Publication 2026-10-27 13:02:55 UTC
RemoteUrl <https://github.com/DavidZenz/GEModelR>
RemoteRef main
RemoteSha 9b257ccf9baf9a79b5b3748d2ee60bcb10e84508

Contents

GEModelR-package	2
GEModel	5
Index	6

GEModelR-package	<i>GEModelR package</i>
------------------	-------------------------

Description

GEModelR interprets GEMPACK-style TABLO models in R. The supported public facade is the GEModel reference-class generator. It preserves the established load, configure, solve, and inspect workflow while offering a legacy in-memory engine and an opt-in sparse engine.

Supported workflow

Create an instance with GEModel\$new(), load a TABLO file with loadTablo(), and load its input data with loadData(). The engine selected by loadData() initializes the model runtime; solveModel() must use the same engine. To change engines, load the data again. Configure model closure and shocks with setClosure() and setShocks(), then solve and inspect the instance’s solution, variableValues, compact output, and lastDiagnostics fields.

```
model <- GEModel$new()
model$loadTablo("model.tab")
model$loadData(input_data, engine = "sparse")
model$setShocks(shocks)
model$solveModel(engine = "sparse", backend = "Matrix")
```

Supported methods

- The supported reference-class methods are:
- loadTablo(tabloPath) Parse and compile a TABLO model.
 - loadData(inputData, engine = c("legacy", "sparse")) Load input data and initialize the selected runtime.
 - setShocks(shocks) Set explicit model shocks.

`setClosure(exogenous_variables)` Set the exogenous-variable closure and invalidate dependent solve state.

`setMemoryBudget(bytes)` Set this instance's sparse memory budget in bytes.

`estimateMemory(engine = c("legacy", "sparse"), postsim = TRUE)` Estimate memory requirements for the selected runtime.

`solveModel(iter = 3, steps = c(1, 3), engine = c("legacy", "sparse"), postsim = TRUE, diagnostics = FALSE, ...)` Solve with an explicit engine and, for the sparse engine, an independent requested backend.

`retryPostsim(diagnostics = FALSE)` Retry postsimulation from accepted numerical state without solving again.

`saveState(file)` **and** `loadState(file)` Save or restore a validated version-1 logical model state.

Use these methods for supported model changes. Compiler, source-data, index, and cache fields are implementation state and are not supported mutation inputs.

Runtime engines and sparse backends

The engine argument chooses the model runtime: `legacy` uses the established in-memory solver and `sparse` uses sparse execution. The sparse backend argument independently chooses a solver implementation and defaults to `"Matrix"`. The recognized backend identifiers and tiers are:

`Matrix` Supported general sparse backend and default.

`StructuredSchur` **and** `StructuredSchurFGMRES` Supported structured R backends.

`SparseM` **and** `StructuredSchurFGMRESCpp` Compatibility-only backend requests.

`SuiteSparse` Recognized but unavailable in supported installed builds pending portable support. Explicitly select `backend="Matrix"` for a supported sparse solve. An explicit `SuiteSparse` request fails capability preflight before matrix emission or runtime compilation.

Optional backends can fail capability preflight when their requirements are unavailable. A requested backend is not silently replaced by another backend. The legacy engine is a separate runtime choice, not a sparse backend.

Output modes

`solveModel(output = "full")` materializes the compatibility output structure. Compact output and variable/dimension selectors are supported only by the sparse engine; the legacy engine accepts full output and no selectors. `output = "compact"` returns a compact projection; use `variables` and `dimensions` to select the requested values and labels. The defaults are `output = c("full", "compact")`, `variables = NULL`, and `dimensions = NULL`.

Solve diagnostics and conditions

Every solve attempt stores a version-1 envelope in `model$lastDiagnostics`. Its stable fields are:

`schema_version` Envelope schema version, currently 1.

`engine, requested_backend, implementation` Requested runtime/backend and the implementation that handled the request.

`status, condition_class` Stable outcome status and primary condition class, when one applies.

`accepted_numerical_state, retryable_postsims` Whether a numerical state was accepted and whether postsimulation can be retried without another solve.

`failure_phase, failure_reason` Failure location and reason for failed attempts.

`cleanup_status` Solve-scoped cleanup outcome.

Stable statuses are `running`, `succeeded`, `validation_failed`, `capability_failed`, `numerical_failed`, `postsim_failed`, and `committed_state_failed`. Each attempt replaces the previous envelope before validation or dispatch and stores its final success or ordinary error outcome before returning or rethrowing, so an error does not leave a stale success record. A postsimulation failure can report that its accepted numerical state is retained and retryable through `retryPostsims()`.

Stable primary condition classes are `GEModelR_validation_error`, `GEModelR_capability_error`, `GEModelR_numerical_error`, `GEModelR_postsim_error`, `GEModelR_retryable_postsim_error`, and `GEModelR_committed_state_error`. The retryable postsimulation class also inherits from `GEModelR_postsim_error`. Set `diagnostics = TRUE` in `solveModel()` or `retryPostsims()` to add residual history, timings, allocation and capability evidence, detailed cleanup, and memory estimates.

Supported high-level options

These options are read by the solver or logical-state serializer. They do not change numerical defaults when unset.

`GEModelR.sparse.lu_order` Integer from 0 through 3; default 3. Controls sparse LU ordering where the selected sparse solve path uses LU factorization.

`GEModelR.sparse.suite_sparse_ordering` Reserved SuiteSparse ordering; SuiteSparse is unavailable in supported installed builds, so this option does not enable that backend; default "amd". Accepted values are "cholmod", "amd", "metis", "best", "natural", and "none". "given" is rejected.

`GEModelR.sparse.structured_residual_tolerance` Sparse candidate true-residual tolerance; default $2e-7$.

`GEModelR.sparse.elimination_pivot_tolerance` Pivot tolerance relative to the largest original coefficient in each structured elimination block; default $1e-12$. Zero pivots are always rejected.

`GEModelR.sparse.schur_region_batch_size` Structured Schur region batch size; default 8.

`GEModelR.sparse.schur_panel_size` Structured Schur panel size; default 64.

`GEModelR.sparse.schur_restart` Structured Schur iterative solver restart size; default 80.

`GEModelR.sparse.schur_max_iterations` Structured Schur iterative solver iteration limit; default 500.

`GEModelR.sparse.schur_tolerance` Structured Schur iterative solver tolerance; default $2e-7$.

`GEModelR.sparse.schur_cpp_threads` Requested thread count for the explicit C++ backend; default 1.

`GEModelR.serialization.max_bytes` Maximum logical-state payload size in bytes; default 268435456.

`GEModelR.serialization.max_elements` Maximum logical-state payload element count; default 50000000.

The existing `memory_budget` argument to `solveModel()` overrides the per-instance budget for a solve, while `setMemoryBudget(bytes)` sets that instance's budget. No global solver-memory option is defined. Transaction/fault controls and structural or numerical caches are private implementation details.

Author(s)

Maintainer: David Zenz <zenz@wiiw.ac.at> [copyright holder]

Authors:

- Maros Ivanic (Author of the reviewed public-domain/CC0 upstream tabloToR baseline)

See Also

Useful links:

- <https://github.com/DavidZenz/GEModelR>
- Report bugs at <https://github.com/DavidZenz/GEModelR/issues>

GEModel

GEModel reference-class facade

Description

Create a GEModel reference-class generator. Instances provide the supported interface for loading a TABLO model and its data, configuring closure and shocks, solving with the legacy or sparse engine, and inspecting the resulting model state and solve diagnostics. Use the model methods for supported state changes rather than modifying compiler, index, or cache fields directly. SuiteSparse is a recognized backend identifier but is unavailable in supported installed builds pending portable support. Explicitly select `backend="Matrix"` for a supported sparse solve. Explicit SuiteSparse requests fail capability preflight before matrix emission or runtime compilation.

See Also

[GEModelR-package](#)

Index

* **package**

GEModelR-package, [2](#)

GEModel, [5](#)

GEModel-class (GEModel), [5](#)

GEModelR (GEModelR-package), [2](#)

GEModelR-package, [2](#)