

# Package ‘factorana’

April 22, 2026

**Type** Package

**Title** Factor Model Estimation with Latent Variables

**Version** 1.2.0

**Description** A flexible framework for estimating factor models with multiple latent variables. Supports linear, probit, ordered probit, and multinomial logit model components. Features include multi-stage estimation, automatic parameter initialization, analytical gradients and Hessians, and parallel estimation. Methods are described in Heckman, Humphries, and Veramendi (2016) <[doi:10.1016/j.jeconom.2015.12.001](https://doi.org/10.1016/j.jeconom.2015.12.001)>, Heckman, Humphries, and Veramendi (2018) <[doi:10.1086/698760](https://doi.org/10.1086/698760)>, and Humphries, Joensen, and Veramendi (2024) <[doi:10.1257/pandp.20241026](https://doi.org/10.1257/pandp.20241026)>.

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Imports** Rcpp (>= 1.0.0), MASS, nnet, parallel, stats, utils

**LinkingTo** Rcpp, RcppEigen

**Suggests** doParallel, foreach, jsonlite, knitr, nloptr, rmarkdown, testthat (>= 3.0.0), trustOptim

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**NeedsCompilation** yes

**Author** Greg Veramendi [aut, cre],  
Jess Xiong [aut]

**Maintainer** Greg Veramendi <[greg.veramendi@gmail.com](mailto:greg.veramendi@gmail.com)>

**Repository** CRAN

**Date/Publication** 2026-04-22 13:40:08 UTC

## Contents

build\_dynamic\_previous\_stage . . . . . 2

cleanup_parallel_workers . . . . .	3
components_table . . . . .	4
components_to_latex . . . . .	5
define_dynamic_measurement . . . . .	6
define_estimation_control . . . . .	8
define_factor_model . . . . .	9
define_model_component . . . . .	10
define_model_system . . . . .	12
disable_adaptive_quadrature_cpp . . . . .	14
estimate_and_write . . . . .	14
estimate_factorscores_rcpp . . . . .	15
estimate_model . . . . .	17
estimate_model_rcpp . . . . .	18
evaluate_factorscore_likelihood_cpp . . . . .	20
evaluate_likelihood_cpp . . . . .	21
evaluate_loglik_only_cpp . . . . .	22
extract_free_params_cpp . . . . .	22
fix_coefficient . . . . .	23
fix_type_intercepts . . . . .	24
gauss_hermite_quadrature . . . . .	25
get_parameter_info_cpp . . . . .	26
initialize_factor_model_cpp . . . . .	26
initialize_parameters . . . . .	27
print.components_table . . . . .	28
print.factorana_factorscores . . . . .	28
print.factorana_result . . . . .	29
print.factorana_table . . . . .	29
print.model_component . . . . .	30
print.summary.factorana_result . . . . .	30
results_table . . . . .	31
results_to_latex . . . . .	32
set_adaptive_quadrature_cpp . . . . .	33
set_observation_weights_cpp . . . . .	34
summary.factorana_result . . . . .	35
write_model_config_csv . . . . .	35

**Index****36**


---

 build\_dynamic\_previous\_stage

*Build a Stage 2 previous\_stage object from a dynamic measurement fit*


---

**Description**

Constructs a dummy previous\_stage result that plugs the anchor-period measurement intercepts into every factor slot, pairs them with the (tied) shared loadings and residual sigmas / thresholds, and is ready to pass as previous\_stage to a Stage 2 SE\_linear or SE\_quadratic model via [define\\_model\\_system](#).

**Usage**

```
build_dynamic_previous_stage(dyn, stage1_result, data, anchor_period = 1L)
```

**Arguments**

dyn	A <code>dynamic_measurement</code> object from <a href="#">define_dynamic_measurement</a> .
stage1_result	The result object from <code>estimate_model_rcpp(dyn\$model_system, ...)</code> .
data	The same data frame passed to <a href="#">define_dynamic_measurement</a> (needed to rebuild the dummy model system's components — components don't retain the data they were defined on).
anchor_period	Integer index into <code>dyn\$period_prefixes</code> giving the period whose measurement intercepts should be carried into Stage 2. The recommended choice is 1: under $E[f_1] = 0$ this period's intercepts identify the true DGP intercepts. Default: 1.

**Value**

A list suitable as `previous_stage` in [define\\_model\\_system](#): has fields `model_system`, `estimates`, `std_errors`, `convergence`, `loglik`. Every per-component measurement parameter is the corresponding Stage 1 estimate, with the intercepts overwritten to the anchor-period values for all periods.

**See Also**

[define\\_dynamic\\_measurement](#).

---

cleanup\_parallel\_workers

*Clean up orphaned parallel worker processes*

---

**Description**

When using parallel estimation with Ctrl-C interrupts, worker processes may continue running after the main process exits. This function finds and terminates any orphaned R worker processes started by the parallel package.

**Usage**

```
cleanup_parallel_workers(signal = "TERM", verbose = TRUE, list_only = FALSE)
```

**Arguments**

signal	Signal to send (default "TERM" for graceful shutdown, use "KILL" for force)
verbose	Whether to print messages (default TRUE)
list_only	If TRUE, only list potential workers without killing them (default FALSE)

**Value**

Invisible NULL (or vector of PIDs if `list_only = TRUE`)

**Examples**

```
# Safe to run: list potential orphaned parallel workers without killing them.
cleanup_parallel_workers(list_only = TRUE, verbose = FALSE)

# After interrupting a parallel job with Ctrl-C, terminate orphaned workers:
cleanup_parallel_workers()

# Force kill if graceful shutdown doesn't work:
cleanup_parallel_workers(signal = "KILL")
```

---

components_table	<i>Create a components table for a single model</i>
------------------	---

---

**Description**

Creates a table with model components as columns and parameter types as rows. This is similar to how SEM/CFA software displays results.

**Usage**

```
components_table(result, digits = 3, show_se = TRUE, stars = TRUE)
```

**Arguments**

result	A factorana_result object from estimate_model_rcpp()
digits	Number of decimal places (default 3)
show_se	Show standard errors in parentheses (default TRUE)
stars	Add significance stars (default TRUE)

**Value**

A data frame with the formatted table

**Examples**

```
# Estimate a small model and display the components table
set.seed(1); n <- 200
f <- rnorm(n)
dat <- data.frame(intercept = 1,
                  y1 = 1.0 * f + rnorm(n, 0, 0.5),
                  y2 = 0.8 * f + rnorm(n, 0, 0.5))
fm <- define_factor_model(n_factors = 1)
```

```

mc1 <- define_model_component("m1", dat, "y1", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = 1)
mc2 <- define_model_component("m2", dat, "y2", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = NA_real_)
ms <- define_model_system(components = list(mc1, mc2), factor = fm)
ctrl <- define_estimation_control(n_quad_points = 8, num_cores = 1)
fit <- estimate_model_rcpp(ms, dat, control = ctrl,
  optimizer = "nlminb", parallel = FALSE, verbose = FALSE)

components_table(fit)

```

---

components\_to\_latex      *Export components table to LaTeX*

---

### Description

Creates a LaTeX table from a `factorana_result` with components as columns.

### Usage

```

components_to_latex(
  result,
  digits = 3,
  file = NULL,
  caption = NULL,
  label = NULL,
  stars = TRUE,
  note = NULL,
  booktabs = TRUE
)

```

### Arguments

<code>result</code>	A <code>factorana_result</code> object from <code>estimate_model_rcpp()</code>
<code>digits</code>	Number of decimal places (default 3)
<code>file</code>	Optional file path to write the LaTeX table
<code>caption</code>	Table caption (optional)
<code>label</code>	Table label for cross-referencing (optional)
<code>stars</code>	Add significance stars (default TRUE)
<code>note</code>	Footnote text (optional, default includes significance codes)
<code>booktabs</code>	Use <code>booktabs</code> package formatting (default TRUE)

### Value

A character string containing the LaTeX table code

---

```
define_dynamic_measurement
```

*Define a dynamic measurement system for longitudinal factor models*

---

## Description

Builds a Stage 1 measurement model for a single latent construct observed at two or more time points. Measurement invariance is imposed on loadings and residual sigmas (tied across periods via equality\_constraints), while measurement intercepts are left period-specific. This is the recommended Stage 1 setup for an SE\_linear or SE\_quadratic structural model in the second stage.

## Usage

```
define_dynamic_measurement(
  data,
  items,
  period_prefixes,
  model_type = "linear",
  n_categories = NULL,
  covariates = "intercept",
  evaluation_indicator = NULL
)
```

## Arguments

data	Wide-format data frame. Must contain a column named <code>paste0(prefix, item)</code> for every combination of <code>period_prefixes</code> and <code>items</code> , plus any columns named in <code>covariates</code> and <code>evaluation_indicator</code> .
items	Character vector of item names (e.g., <code>c("m1", "m2", "m3")</code> ).
period_prefixes	Character vector of column prefixes, one per period. Column names are assembled as <code>paste0(prefix, item)</code> . E.g., <code>c("Y_t1_", "Y_t2_")</code> yields data columns <code>"Y_t1_m1"</code> , <code>"Y_t2_m1"</code> , etc. Length of <code>period_prefixes</code> determines the number of latent factors in Stage 1 (one per period).
model_type	One of <code>"linear"</code> , <code>"oprobit"</code> , <code>"probit"</code> , <code>"logit"</code> . The same type is used for every item and every period.
n_categories	Required for <code>model_type = "oprobit"</code> ; the number of ordered categories (shared across items and periods).
covariates	Character vector of covariate column names; default <code>"intercept"</code> . Same covariates for every component.
evaluation_indicator	Name of a column with 0/1 values indicating which observations contribute to each component's likelihood; NULL to use all rows.

## Details

Why period-specific intercepts? Under the usual factor-model identification convention  $E[f_k] = 0$  for every factor  $k$ , pooling measurement intercepts across periods biases them by  $\lambda_m \cdot E[f_2]/2$  when the period-mean drifts between waves — an artefact that propagates into an under-estimate of `se_intercept` in Stage 2. Leaving the intercepts period-specific and carrying the wave-1 intercepts into Stage 2 (via `build_dynamic_previous_stage`) sidesteps the bias.

## Value

An object of class "dynamic\_measurement": a list with

- `model_system`: a `model_system` object ready to pass to `estimate_model_rcpp`.
- `factor_model`: the underlying `factor_model`.
- `items`, `period_prefixes`, `model_type`, `n_categories`, `covariates`, `evaluation_indicator`: the inputs, kept for use by `build_dynamic_previous_stage`.

## See Also

`build_dynamic_previous_stage` constructs the Stage 2 `previous_stage` object from the Stage 1 estimation result.

## Examples

```
# Simulate a simple dynamic single-factor model
set.seed(1); n <- 500
f1 <- rnorm(n); eps <- rnorm(n, 0, sqrt(0.5))
f2 <- 0.4 + 0.6 * f1 + eps
dat <- data.frame(
  intercept = 1, eval = 1L,
  Y_t1_m1 = 1.5 + f1 + rnorm(n, 0, 0.7),
  Y_t1_m2 = 1.0 + 0.9 * f1 + rnorm(n, 0, 0.75),
  Y_t1_m3 = 0.8 + 1.1 * f1 + rnorm(n, 0, 0.65),
  Y_t2_m1 = 1.5 + f2 + rnorm(n, 0, 0.7),
  Y_t2_m2 = 1.0 + 0.9 * f2 + rnorm(n, 0, 0.75),
  Y_t2_m3 = 0.8 + 1.1 * f2 + rnorm(n, 0, 0.65)
)
dyn <- define_dynamic_measurement(
  data = dat, items = c("m1", "m2", "m3"),
  period_prefixes = c("Y_t1_", "Y_t2_"),
  model_type = "linear", evaluation_indicator = "eval"
)
ctrl <- define_estimation_control(n_quad_points = 8, num_cores = 1)
s1 <- estimate_model_rcpp(dyn$model_system, dat, control = ctrl,
  optimizer = "nlminb", parallel = FALSE,
  verbose = FALSE)
prev <- build_dynamic_previous_stage(dyn, s1, dat) # Stage 2 input
```

---

```
define_estimation_control
    Define estimation control settings
```

---

### Description

Define estimation control settings

### Usage

```
define_estimation_control(
  n_quad_points = 16,
  num_cores = 1,
  cluster_type = "auto",
  adaptive_integration = FALSE,
  adapt_int_thresh = 0.5
)
```

### Arguments

n_quad_points	Integer. Number of Gauss-Hermite quadrature points for numerical integration (default = 16)
num_cores	Integer. Number of processes to use for parallel estimation (default = 1)
cluster_type	Character. Type of parallel cluster to use: "auto" (default), "FORK", or "PSOCK". "auto" uses FORK on Unix (faster, shared memory) and PSOCK on Windows. "FORK" forces fork-based parallelism (Unix only, faster due to shared memory). "PSOCK" forces socket-based parallelism (works on all platforms, more overhead).
adaptive_integration	Logical. Whether to use adaptive integration in second-stage estimation (default = FALSE). When TRUE, the number of quadrature points per observation is determined based on the standard error of factor scores from a previous estimation stage.
adapt_int_thresh	Numeric. Threshold for adaptive integration (default = 0.5). Smaller values use more integration points. The formula is: $n\_quad\_obs = 1 + 2 * \text{floor}(factor\_se / factor\_var / adapt\_int\_thresh)$ . When factor_se is small relative to factor variance, fewer quadrature points are used. Legacy code default is 0.5.

### Details

Adaptive integration is useful in two-stage estimation where factor scores have been estimated in a first stage. The key insight is that observations with well-identified factor scores (small SE) need fewer integration points, while observations with poorly-identified factors (large SE) need full quadrature.

To use adaptive integration:



1. Estimate Stage 1 model to get factor scores via `estimate_factor_scores_rcpp()`
2. Initialize FactorModel for Stage 2 via `initialize_factor_model_cpp()`
3. Call `set_adaptive_quadrature_cpp()` with factor scores, SEs, and variances
4. Run estimation - the adaptive settings are applied automatically

The diagnostic output from `set_adaptive_quadrature_cpp(verbose=TRUE)` shows:

- Distribution of integration points across observations
- Average integration points vs standard (shows computational savings)
- Computational reduction percentage

### Value

An object of class `estimation_control` containing control settings

---

`define_factor_model`     *Define latent factor model structure*

---

### Description

Creates an object of class "factor\_model" that specifies the structure of the unobserved latent factors. This includes the number of factors and mixture components. Loading constraints are specified at the component level via `define_model_component()`. Numerical integration settings (quadrature points) are specified in `define_estimation_control()`.

### Usage

```
define_factor_model(
  n_factors,
  n_types = 1,
  factor_structure = "independent",
  n_mixtures = 1,
  factor_covariates = NULL,
  se_covariates = NULL
)
```

### Arguments

<code>n_factors</code>	Integer. Number of latent factors ( $\geq 0$ ). Use 0 for models without latent factors.
<code>n_types</code>	Integer. Number of types ( $\geq 1$ )
<code>factor_structure</code>	Character. Structure of factor dependencies. Options: <ul style="list-style-type: none"> <li>• "independent" (default): Factors are independent</li> <li>• "correlation": Correlated factors via Cholesky decomposition (2 factors only)</li> </ul>

- "SE\_linear": Structural equation  $f_k = \alpha + \alpha_1 f_1 + \dots + \epsilon$
  - "SE\_quadratic": Adds quadratic terms:  $f_k = \alpha + \alpha_1 f_1 + \alpha_2 f_1^2 + \dots + \epsilon$
- n\_mixtures** Integer. Number of discrete mixtures (default = 1, allowed: 1-3)
- factor\_covariates** Character vector. Names of covariates that shift factor means. When specified, the factor distribution becomes  $f_i \sim N((X_i - \text{mean}(X)) * \gamma, \Sigma)$  where  $X_i$  is the covariate vector for observation  $i$  and  $\gamma$  is a matrix of coefficients (one column per factor). This allows factor means to vary by observed characteristics. The covariates must be present in the data passed to `estimate_model_rcpp()`. **Note:** All covariates are automatically demeaned internally to ensure the overall factor mean is zero (required for identification). Including an intercept/constant has no effect since it becomes zero after demeaning. For SE\_linear/SE\_quadratic structures, covariates only affect input factor means (not the outcome factor which is determined by the structural equation).
- se\_covariates** Character vector. Names of covariates that directly affect the outcome factor in SE\_linear/SE\_quadratic models. When specified, the structural equation becomes  $f_k = \text{intercept} + \sum(\alpha_j * f_j) + \sum(\beta_m * X_m) + \epsilon$ . Only valid for factor\_structure = "SE\_linear" or "SE\_quadratic". All covariates are automatically demeaned internally for identification.

**Value**

An object of class "factor\_model"

**Examples**

```
# Single factor model
fm <- define_factor_model(n_factors = 1)

# Two-factor structural equation model
fm_se <- define_factor_model(n_factors = 2, factor_structure = "SE_linear")
```

---

define\_model\_component

*Define a model component*

---

**Description**

Define a model component

**Usage**

```
define_model_component(
  name,
  data,
  outcome,
```

```

    factor,
    evaluation_indicator = NULL,
    covariates,
    model_type = c("linear", "logit", "probit", "oprobit"),
    intercept = TRUE,
    num_choices = 2,
    nrank = NULL,
    exclude_chosen = TRUE,
    rankshare_var = NULL,
    loading_normalization = NULL,
    factor_spec = c("linear", "quadratic", "interactions", "full"),
    use_types = FALSE,
    skip_collinearity_check = FALSE
)

```

### Arguments

name	name of model component #####Long name and s name (like in the C++)
data	data.frame for validation
outcome	Character. Name of the outcome variable.
factor	model. Object of Class factor_model.
evaluation_indicator	Character (optional). Variable used for evaluation subsample.
covariates	List of Character vectors. Names of covariates.
model_type	Character. Type of model (e.g., "linear", "logit", "probit").
intercept	Logical. Whether to include an intercept (default = TRUE).
num_choices	Integer. Number of choices (for multinomial models).
nrank	Integer (optional). Rank for exploded multinomial logit.
exclude_chosen	Logical. For exploded logit, whether to exclude already-chosen alternatives from later ranks (default TRUE). Set to FALSE for exploded nested logit where the same nest can be chosen multiple times.
rankshare_var	Character (optional). Column name (or prefix) for rank-share correction variables. These provide rank-and-choice-specific adjustments to the linear predictor. Data layout: (num_choices-1) * nrank columns, accessed as rankshare_var + (num_choices-1)*irank + icat for irank=0..nrank-1, icat=0..num_choices-2.
loading_normalization	Numeric vector of length n_factors (optional). Component-specific loading constraints. Overrides factor model normalization. <ul style="list-style-type: none"> <li>• NA → loading is free (estimated).</li> <li>• numeric value → loading is fixed at that value (e.g. 1 for identification). If NULL, uses the factor model's default normalization.</li> </ul>
factor_spec	Character. Specification for factor terms in linear predictor. <ul style="list-style-type: none"> <li>• "linear" (default): Only linear factor terms (<math>\lambda * f</math>)</li> <li>• "quadratic": Linear + quadratic terms (<math>\lambda * f + \lambda_{quad} * f^2</math>)</li> </ul>

- "interactions": Linear + interaction terms ( $\lambda * f + \lambda_{inter} * f_j * f_k$ )
  - "full": Linear + quadratic + interaction terms Note: Interaction terms require `n_factors >= 2`.
- `use_types` Logical. Whether this component uses type-specific intercepts when `n_types > 1` in the factor model. Default FALSE. When TRUE and `n_types > 1`, the component will have  $(n\_types - 1)$  type-specific intercept parameters that shift the linear predictor for each non-reference type. This allows types to affect outcome models while keeping measurement models type-invariant.
- `skip_collinearity_check` Logical. If TRUE, skip the multicollinearity check on the design matrix. Useful when many coefficients will be fixed via `fix_coefficient()` after component creation, which resolves the collinearity. Default FALSE.

### Value

An object of class "model\_component". A list representing the model component

### Examples

```
dat <- data.frame(y = rnorm(50), intercept = 1)
fm <- define_factor_model(n_factors = 1)
mc <- define_model_component("Y", dat, "y", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = 1)
```

---

`define_model_system`    *Define a model system*

---

### Description

Define a model system

### Usage

```
define_model_system(
  components,
  factor,
  previous_stage = NULL,
  weights = NULL,
  equality_constraints = NULL,
  free_params = NULL
)
```

**Arguments**

components	A named list of model_component objects
factor	A factor_model object
previous_stage	Optional result from a previous estimation stage. If provided, the previous stage components and parameters will be fixed and prepended to the new components. This enables sequential/multi-stage estimation where early stages are held fixed while later stages are optimized.
weights	Optional. Name of a variable in the data containing observation weights. When specified, each observation's contribution to the log-likelihood is multiplied by its weight. Useful for survey weights, importance sampling, or giving different observations different influence on the estimation. Weights should be positive. The variable is extracted from the data passed to estimate_model_rcpp().
equality_constraints	Optional. A list of character vectors, where each vector specifies parameter names that should be constrained to be equal during estimation. The first parameter in each group is the "primary" (freely estimated), and all other parameters in the group are set equal to the primary. Example: <code>list(c("Y1_loading_1", "Y2_loading_2"), c("Y1_sigma", "Y2_sigma"))</code> This is useful for measurement invariance constraints in longitudinal models.
free_params	Optional. A character vector of parameter names from previous_stage that should remain FREE (not fixed) in the current stage. This allows selectively freeing specific parameters while keeping others fixed. Commonly used to free factor variances while keeping measurement loadings/thresholds fixed. Example: <code>c("factor_var_1", "factor_var_2")</code> to free factor variances. Only used when previous_stage is provided.

**Value**

An object of class "model\_system". A list of model\_component objects and one factor\_model object.

**Examples**

```
dat <- data.frame(y1 = rnorm(50), y2 = rnorm(50), intercept = 1)
fm <- define_factor_model(n_factors = 1)
mc1 <- define_model_component("m1", dat, "y1", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = 1)
mc2 <- define_model_component("m2", dat, "y2", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = NA_real_)
ms <- define_model_system(components = list(mc1, mc2), factor = fm)
```

---

disable\_adaptive\_quadrature\_cpp  
*Disable adaptive quadrature*

---

**Description**

Reverts to standard (non-adaptive) quadrature integration.

**Usage**

```
disable_adaptive_quadrature_cpp(fm_ptr)
```

**Arguments**

fm\_ptr            External pointer to FactorModel object

**Value**

No return value. Called for its side effect of disabling adaptive quadrature and clearing any observation weights on the FactorModel pointed to by fm\_ptr.

---

estimate\_and\_write    *Run estimation and write standard output files*

---

**Description**

Calls estimate\_model(), then writes the four expected output files:

- model\_config.csv
- meas\_par.csv
- system\_inits\_long.csv
- simulated\_data.csv

**Usage**

```
estimate_and_write(  
  model_system,  
  factor_model,  
  control,  
  data = NULL,  
  results_dir  
)
```

**Arguments**

model_system	model_system object
factor_model	factor_model object
control	estimation_control object
data	Optional data frame (to save as simulated_data.csv)
results_dir	Directory for writing output files. This argument is required; the function writes nothing without an explicit path. Use tempdir() in examples or tests.

**Value**

Invisibly returns a list with two components: results (a data frame of initial parameter values per component, as produced by estimate\_model()) and packed (a list with values and ses giving the packed parameter vector and standard errors written to meas\_par.csv). Called primarily for the side effect of writing model\_config.csv, meas\_par.csv, system\_inits\_long.csv, and optionally simulated\_data.csv into results\_dir.

---

estimate\_factorscores\_rcpp

*Estimate Factor Scores*

---

**Description**

Estimates factor scores for each observation after model estimation. The model parameters are held fixed at their estimated values, and factor scores are computed as the posterior mode for each observation.

**Usage**

```
estimate_factorscores_rcpp(
  result,
  data,
  control = NULL,
  parallel = FALSE,
  verbose = TRUE,
  include_prior = FALSE,
  id_var = NULL
)
```

**Arguments**

result	A factorana_result object from estimate_model_rcpp()
data	Data frame containing all variables (same as used in estimation)
control	Optional estimation control object. If NULL, uses default.
parallel	Whether to use parallel computation (default FALSE). When TRUE, uses the num_cores setting from control.

verbose	Whether to print progress (default TRUE)
include_prior	Whether to include the factor prior in likelihood/SE computation (default FALSE). When FALSE, matches legacy C++ behavior where SEs are based only on observation likelihood. When TRUE, includes the prior contribution to the Hessian, resulting in smaller SEs.
id_var	Optional character string specifying the name of an ID variable in data. If provided, this variable is included in the output data frame for easier merging with other datasets. The ID values are taken from the original data in the order observations were processed. Note: The ID column must be numeric (not character) since data is converted to a numeric matrix. For character IDs, use the obs_id column to merge with your original data.

## Details

Factor scores are estimated by maximizing the posterior density:

$$L(f|y_i, \theta) = p(y_i|f, \theta) \cdot \phi(f|0, \sigma^2)$$

where:

- $f$  is the vector of factor values for observation  $i$
- $y_i$  is the observed data for observation  $i$
- $\theta$  are the fixed model parameters (from previous estimation)
- $\phi(f|0, \sigma^2)$  is the normal prior on factors

Standard errors are computed from the diagonal of the inverse Hessian of the log-posterior at the mode. By default (include\_prior=FALSE), the SE is based only on the observation likelihood Hessian, matching the legacy C++ implementation. Set include\_prior=TRUE to include the prior's Hessian contribution ( $-1/\sigma^2$ ) which produces smaller SEs.

When parallel=TRUE, observations are distributed across cores using doParallel/foreach, with each worker processing a subset of observations.

## Value

A data frame with columns:

- obs\_id - Observation index (1-based)
- <id\_var> - ID variable values (if id\_var was specified)
- factor\_1, factor\_2, ... - Estimated factor scores
- se\_factor\_1, se\_factor\_2, ... - Standard errors
- converged - Whether optimization converged for this observation
- log\_posterior - Log-posterior value at the mode



**Examples**

```

# Estimate a small one-factor model, then recover factor scores
set.seed(1); n <- 200
f <- rnorm(n)
dat <- data.frame(intercept = 1,
                  y1 = 1.0 * f + rnorm(n, 0, 0.5),
                  y2 = 0.8 * f + rnorm(n, 0, 0.5))
fm <- define_factor_model(n_factors = 1)
mc1 <- define_model_component("m1", dat, "y1", fm,
                             covariates = "intercept", model_type = "linear",
                             loading_normalization = 1)
mc2 <- define_model_component("m2", dat, "y2", fm,
                             covariates = "intercept", model_type = "linear",
                             loading_normalization = NA_real_)
ms <- define_model_system(components = list(mc1, mc2), factor = fm)
ctrl <- define_estimation_control(n_quad_points = 8, num_cores = 1)
fit <- estimate_model_rcpp(ms, dat, control = ctrl,
                          optimizer = "nlminb", parallel = FALSE, verbose = FALSE)

fscores <- estimate_factorscores_rcpp(fit, dat, control = ctrl,
                                     parallel = FALSE, verbose = FALSE)

head(fscores)

```

---

estimate\_model

*Estimate model*


---

**Description**

Estimate model

**Usage**

```
estimate_model(ms, control)
```

**Arguments**

ms                    an object of class `model_system`  
control                an object of class `estimation_control`

**Value**

description

---

estimate\_model\_rcpp     *Estimate factor model using R-based optimization*

---

### Description

This function estimates a factor model by optimizing the likelihood using C++ for fast evaluation and R for optimization and parallelization.

### Usage

```
estimate_model_rcpp(
  model_system,
  data,
  init_params = NULL,
  control = NULL,
  optimizer = "nlminb",
  parallel = TRUE,
  verbose = TRUE,
  max_restarts = 5,
  factor_scores = NULL,
  factor_ses = NULL,
  factor_vars = NULL,
  init_factor_scores = NULL,
  checkpoint_file = NULL
)
```

### Arguments

model_system	A model_system object from define_model_system()
data	Data frame containing all variables
init_params	Initial parameter values (optional)
control	Estimation control object from define_estimation_control()
optimizer	Optimizer to use (default "nloptr"): <ul style="list-style-type: none"> <li>• "nloptr" - L-BFGS via nloptr (gradient only, no Hessian)</li> <li>• "optim" - L-BFGS-B via stats::optim (gradient only, no Hessian)</li> <li>• "nlminb" - Uses analytical gradient AND Hessian (more efficient!)</li> <li>• "trust" - Trust region method with analytical Hessian (requires trustOptim package)</li> </ul>
parallel	Whether to use parallel computation (default TRUE)
verbose	Whether to print progress (default TRUE)
max_restarts	Maximum number of eigenvector-based restarts for escaping saddle points (default 5). Set to 0 to disable.
factor_scores	Matrix (n_obs x n_factors) of factor score estimates from a previous stage. Used for adaptive quadrature. (default NULL)

factor_ses	Matrix (n_obs x n_factors) of factor score standard errors from a previous stage. Used for adaptive quadrature. (default NULL)
factor_vars	Named numeric vector of factor variances from a previous stage. Used for adaptive quadrature. (default NULL)
init_factor_scores	Matrix (n_obs x n_factors) of factor scores to use for initializing factor loadings. When provided, loadings are estimated by treating factor scores as regressors, giving better starting values than the default (0.5). Useful for two-stage estimation where Stage 1 factor scores can improve Stage 2 initialization. (default NULL)
checkpoint_file	Path to file for saving checkpoint parameters during optimization. When specified, parameters are saved each time the Hessian is evaluated at a point with improved likelihood. Useful for long-running estimations that may need to be restarted. The file contains parameter names and values as CSV with metadata headers. (default NULL = no checkpointing)

## Details

For maximum efficiency, use `optimizer = "nlminb"` or `optimizer = "trust"` which exploit the analytical Hessian computed in C++. The default L-BFGS methods only use the gradient and approximate the Hessian from gradient history.

When optimization fails to converge (possibly at a saddle point), the function will attempt to escape by moving in the direction of negative Hessian eigenvalues. This is controlled by the `max_restarts` parameter.

## Value

List with parameter estimates, standard errors, log-likelihood, etc.

## Examples

```
# Simulate a simple one-factor model with two linear indicators
set.seed(1); n <- 100
f <- rnorm(n)
dat <- data.frame(intercept = 1,
  y1 = 1.0 * f + rnorm(n, 0, 0.5),
  y2 = 0.8 * f + rnorm(n, 0, 0.5))
fm <- define_factor_model(n_factors = 1)
mc1 <- define_model_component("m1", dat, "y1", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = 1)
mc2 <- define_model_component("m2", dat, "y2", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = NA_real_)
ms <- define_model_system(components = list(mc1, mc2), factor = fm)
ctrl <- define_estimation_control(n_quad_points = 8, num_cores = 1)
result <- estimate_model_rcpp(ms, dat, control = ctrl,
  optimizer = "nlminb", parallel = FALSE, verbose = FALSE)
result$estimates
```

---

`evaluate_factorscore_likelihood_cpp`*Evaluate log-likelihood for a single observation at given factor values*

---

### Description

Used for factor score estimation. The model parameters are held fixed, and the factor values are treated as the parameters to optimize.

### Usage

```
evaluate_factorscore_likelihood_cpp(  
  fm_ptr,  
  iobs,  
  factor_values,  
  model_params,  
  compute_gradient = FALSE,  
  compute_hessian = FALSE,  
  include_prior = TRUE  
)
```

### Arguments

<code>fm_ptr</code>	External pointer to FactorModel object
<code>iobs</code>	Observation index (0-based)
<code>factor_values</code>	Vector of factor values (size <code>n_factors</code> )
<code>model_params</code>	Vector of ALL model parameters (from previous estimation)
<code>compute_gradient</code>	Whether to compute gradient (default FALSE)
<code>compute_hessian</code>	Whether to compute Hessian (default FALSE)
<code>include_prior</code>	Whether to include factor prior in likelihood (default TRUE). Set to FALSE to match legacy C++ behavior (observation likelihood only).

### Value

List with log-likelihood, gradient (if requested), and Hessian (if requested)

---

`evaluate_likelihood_cpp`*Evaluate log-likelihood for given parameters*

---

### Description

Evaluate log-likelihood for given parameters

### Usage

```
evaluate_likelihood_cpp(  
  fm_ptr,  
  params,  
  compute_gradient = FALSE,  
  compute_hessian = FALSE  
)
```

### Arguments

<code>fm_ptr</code>	External pointer to FactorModel object
<code>params</code>	Vector of parameters
<code>compute_gradient</code>	Whether to compute gradient (default FALSE)
<code>compute_hessian</code>	Whether to compute Hessian (default FALSE)

### Value

List with:

- `logLikelihood`: scalar log-likelihood value
- `gradient`: vector of length `n_param_free` (if requested)
- `hessian`: vector of length  $n\_param\_free * (n\_param\_free + 1) / 2$  stored as upper-triangular in row-major order (if requested). To expand to full symmetric matrix in R: `idx <- 1; for(i in 1:n) for(j in i:n) { H[i,j] <- H[j,i] <- hess[idx]; idx <- idx + 1 }`

---

`evaluate_loglik_only_cpp`*Evaluate log-likelihood only (for optimization)*

---

**Description**

Evaluate log-likelihood only (for optimization)

**Usage**

```
evaluate_loglik_only_cpp(fm_ptr, params)
```

**Arguments**

<code>fm_ptr</code>	External pointer to FactorModel object
<code>params</code>	Vector of parameters

**Value**

Log-likelihood value

---

`extract_free_params_cpp`*Extract free parameters from full parameter vector*

---

**Description**

Given a full parameter vector (including fixed parameters), extract only the free parameters based on the model's fixed parameter mask.

**Usage**

```
extract_free_params_cpp(fm_ptr, full_params)
```

**Arguments**

<code>fm_ptr</code>	External pointer to FactorModel object
<code>full_params</code>	Full parameter vector (size <code>n_param</code> )

**Value**

Vector of free parameters only (size `n_param_free`)

---

fix_coefficient	<i>Fix a coefficient in a model component</i>
-----------------	---

---

### Description

Constrains a regression coefficient (beta) to a fixed value during estimation. The coefficient will not be optimized - it remains at the specified value.

### Usage

```
fix_coefficient(component, covariate, value, choice = NULL)
```

### Arguments

component	A model_component object from define_model_component()
covariate	Character. Name of the covariate whose coefficient to fix. Must be one of the covariates specified when creating the component.
value	Numeric. The value to fix the coefficient to.
choice	Integer (optional). For multinomial logit models with num_choices > 2, specifies which choice's coefficient to fix. Must be between 1 and (num_choices - 1). Choice 0 (reference category) has no parameters. For other model types, this should be NULL (default).

### Details

Fixed coefficients are stored in the component and used during:

- Parameter initialization: fixed values are used directly (no estimation)
- Optimization: fixed parameters are excluded from the optimization
- Likelihood evaluation: fixed values are inserted into the parameter vector

Note: This function only fixes regression coefficients (betas), not:

- Factor loadings (use loading\_normalization in define\_model\_component)
- Sigma (for linear models)
- Thresholds (for ordered probit)

### Value

The modified model\_component object with the fixed coefficient constraint added.

**Examples**

```
# Build a minimal model component and fix a coefficient
dat <- data.frame(intercept = 1, x1 = rnorm(20), y = rnorm(20))
fm <- define_factor_model(n_factors = 1)
mc <- define_model_component(
  name = "y", data = dat, outcome = "y", factor = fm,
  covariates = c("intercept", "x1"), model_type = "linear",
  loading_normalization = 1
)

# Fix intercept to 0 and x1 coefficient to 0.1
mc <- fix_coefficient(mc, covariate = "intercept", value = 0.0)
mc <- fix_coefficient(mc, covariate = "x1", value = 0.1)
length(mc$fixed_coefficients) # 2 constraints stored on the component
```

---

fix\_type\_intercepts     *Fix type-specific intercepts to zero for a model component*

---

**Description**

For models with `n_types > 1`, each component has type-specific intercepts that shift the linear predictor for each non-reference type. This function constrains those intercepts to zero, effectively removing the type-specific shift for this component.

**Usage**

```
fix_type_intercepts(component, types = NULL, choice = NULL)
```

**Arguments**

<code>component</code>	A <code>model_component</code> object from <code>define_model_component()</code>
<code>types</code>	Integer vector (optional). Which types to fix. Must be between 2 and <code>n_types</code> (type 1 is the reference with no intercept). Default is <code>NULL</code> , meaning all non-reference types.
<code>choice</code>	Integer (optional). For multinomial logit models with <code>num_choices &gt; 2</code> , specifies which choice's type intercepts to fix. Must be between 1 and <code>(num_choices - 1)</code> . Default is <code>NULL</code> , meaning all choices.

**Details**

When `n_types > 1`, the model includes a latent type structure where different types can have different intercepts in each measurement equation. By default, these intercepts are freely estimated. Fixing them to zero constrains the outcome equation to have no type-specific shifts (though the type model loadings at the factor level may still differ).

This is useful when you want the type model to affect outcomes only through factor loadings, not through direct type-specific intercepts.

Fixed type intercepts are stored in the component and used during:



- Parameter initialization: fixed values are used directly (no estimation)
- Optimization: fixed parameters are excluded from the optimization
- Likelihood evaluation: fixed values are inserted into the parameter vector

### Value

The modified model\_component object with the fixed type intercept constraints added.

### Examples

```
# Build a component with n_types = 2 and fix the type-2 intercept
dat <- data.frame(intercept = 1, y = rnorm(20))
fm <- define_factor_model(n_factors = 1, n_types = 2)
mc <- define_model_component(
  name = "y", data = dat, outcome = "y", factor = fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = 1, use_types = TRUE
)
mc <- fix_type_intercepts(mc)
length(mc$fixed_type_intercepts)
```

---

gauss\_hermite\_quadrature

*Compute Gauss-Hermite quadrature nodes and weights*

---

### Description

Compute Gauss-Hermite quadrature nodes and weights

### Usage

```
gauss_hermite_quadrature(n)
```

### Arguments

n                      Number of quadrature points

### Value

List with nodes and weights

---

```
get_parameter_info_cpp
```

*Get parameter counts from FactorModel*

---

### Description

Get parameter counts from FactorModel

### Usage

```
get_parameter_info_cpp(fm_ptr)
```

### Arguments

fm\_ptr            External pointer to FactorModel object

### Value

List with parameter count information

---

```
initialize_factor_model_cpp
```

*Initialize a FactorModel C++ object from R model system*

---

### Description

Initialize a FactorModel C++ object from R model system

### Usage

```
initialize_factor_model_cpp(  
  model_system,  
  data,  
  n_quad = 8L,  
  init_params = NULL  
)
```

### Arguments

model\_system    R model\_system object  
data            Data frame or matrix with all variables  
n\_quad          Number of quadrature points  
init\_params     Optional initial parameter vector (used to set fixed parameter values)

### Value

External pointer to FactorModel object

---

initialize\_parameters *Initialize parameters for factor model estimation*

---

### Description

Estimates each model component separately in R (ignoring factors) to obtain good starting values. Also checks factor identification.

### Usage

```
initialize_parameters(model_system, data, factor_scores = NULL, verbose = TRUE)
```

### Arguments

model_system	A model_system object from define_model_system()
data	Data frame containing all variables
factor_scores	Optional matrix of factor scores (nobs x n_factors). When provided, factor loadings are estimated by including factor scores as regressors in the initialization regressions. This is useful for two-stage estimation where factor scores from Stage 1 can be used to initialize loadings in Stage 2 outcomes.
verbose	Whether to print progress (default TRUE)

### Details

Factor identification: For each factor, if NO component has a non-zero fixed loading, then the factor variance is not identified and must be fixed to 1.0.

When factor\_scores is provided, the initialization will estimate loadings by treating factor scores as additional regressors. For example, for a linear model:  $Y \sim X + \text{factor\_scores}$  and the coefficients on factor\_scores become the loading estimates. This typically provides better starting values than the default (0.5).

### Value

List with:

- init\_params - Initial parameter values
- factor\_variance\_fixed - Logical vector indicating which factor variances must be fixed

print.components\_table

*Print method for components\_table*

---

### Description

Print method for components\_table

### Usage

```
## S3 method for class 'components_table'  
print(x, ...)
```

### Arguments

x	A components_table object
...	Additional arguments (ignored)

### Value

Invisibly returns x. Called for its side effect of printing a components-as-columns view of model estimates (one column per model component, factor parameters in a separate section) to the console.

---

print.factorana\_factorscores

*Print method for factorana\_factorscores*

---

### Description

Print method for factorana\_factorscores

### Usage

```
## S3 method for class 'factorana_factorscores'  
print(x, n = 10, ...)
```

### Arguments

x	A factorana_factorscores object
n	Number of rows to print (default 10)
...	Additional arguments (ignored)

### Value

Invisibly returns x. Called for its side effect of printing a summary of the factor score estimates (convergence rate, per-factor summary statistics, and the first n rows) to the console.

---

```
print.factorana_result
```

*Print and Summary Methods for Factor Model Results*

---

### Description

Functions for displaying and exporting factor model estimation results in formatted tables, including LaTeX output. Print method for factorana\_result objects

### Usage

```
## S3 method for class 'factorana_result'
print(x, digits = 4, ...)
```

### Arguments

x	A factorana_result object from estimate_model_rcpp()
digits	Number of decimal places (default 4)
...	Additional arguments (ignored)

### Value

Invisibly returns x. Called for its side effect of printing a formatted summary (convergence status, log-likelihood, and a parameter table with estimates and standard errors) to the console.

---

```
print.factorana_table Print method for factorana_table
```

---

### Description

Print method for factorana\_table

### Usage

```
## S3 method for class 'factorana_table'
print(x, ...)
```

### Arguments

x	A factorana_table object
...	Additional arguments (ignored)

### Value

Invisibly returns x. Called for its side effect of printing a side-by-side model-comparison table (parameters grouped by component, with estimates and standard errors) to the console.

---

```
print.model_component Print method for model_component objects
```

---

**Description**

Print method for model\_component objects

**Usage**

```
## S3 method for class 'model_component'  
print(x, ...)
```

**Arguments**

x	An object of class "model_component".
...	Not used.

**Value**

Invisibly returns x. Called for its side effect of printing a human-readable summary of the component to the console.

---

```
print.summary.factorana_result  
Print method for summary.factorana_result
```

---

**Description**

Print method for summary.factorana\_result

**Usage**

```
## S3 method for class 'summary.factorana_result'  
print(x, digits = 4, ...)
```

**Arguments**

x	A summary.factorana_result object
digits	Number of decimal places (default 4)
...	Additional arguments (ignored)

**Value**

Invisibly returns x. Called for its side effect of printing a detailed estimation summary (convergence, log-likelihood, and a coefficient table with z-values and significance stars) to the console.



```

mc2 <- define_model_component("m2", dat, "y2", fm,
  covariates = "intercept", model_type = "linear",
  loading_normalization = NA_real_)
ms <- define_model_system(components = list(mc1, mc2), factor = fm)
ctrl <- define_estimation_control(n_quad_points = 8, num_cores = 1)
fit <- estimate_model_rcpp(ms, dat, control = ctrl,
  optimizer = "nlminb", parallel = FALSE, verbose = FALSE)

results_table(fit, model_names = "Baseline")

```

---

results\_to\_latex

*Export results table to LaTeX*


---

## Description

Creates a LaTeX table from factorana results, suitable for academic papers.

## Usage

```

results_to_latex(
  ...,
  model_names = NULL,
  digits = 3,
  file = NULL,
  caption = NULL,
  label = NULL,
  stars = TRUE,
  note = NULL,
  booktabs = TRUE,
  include_loglik = TRUE,
  param_labels = NULL
)

```

## Arguments

...	One or more factorana_result objects, or a list of them
model_names	Optional character vector of model names for column headers
digits	Number of decimal places (default 3)
file	Optional file path to write the LaTeX table
caption	Table caption (optional)
label	Table label for cross-referencing (optional)
stars	Add significance stars (default TRUE)
note	Footnote text (optional, default includes significance codes)
booktabs	Use booktabs package formatting (default TRUE)
include_loglik	Include log-likelihood row (default TRUE)
param_labels	Optional named list mapping parameter names to display labels



**Value**

A character string containing the LaTeX table code

**Examples**

```
# Estimate a small model and export the result as a LaTeX table
set.seed(1); n <- 200
f <- rnorm(n)
dat <- data.frame(intercept = 1,
                  y1 = 1.0 * f + rnorm(n, 0, 0.5),
                  y2 = 0.8 * f + rnorm(n, 0, 0.5))
fm <- define_factor_model(n_factors = 1)
mc1 <- define_model_component("m1", dat, "y1", fm,
                              covariates = "intercept", model_type = "linear",
                              loading_normalization = 1)
mc2 <- define_model_component("m2", dat, "y2", fm,
                              covariates = "intercept", model_type = "linear",
                              loading_normalization = NA_real_)
ms <- define_model_system(components = list(mc1, mc2), factor = fm)
ctrl <- define_estimation_control(n_quad_points = 8, num_cores = 1)
fit <- estimate_model_rcpp(ms, dat, control = ctrl,
                          optimizer = "nlminb", parallel = FALSE, verbose = FALSE)

# Return LaTeX as a character string
tex <- results_to_latex(fit, model_names = "Baseline")

# Or write to a file inside tempdir()
results_to_latex(fit,
                 model_names = "Baseline",
                 file = file.path(tempdir(), "results_table.tex"))
```

---

set\_adaptive\_quadrature\_cpp

*Set up adaptive quadrature based on factor scores and standard errors*

---

**Description**

Enables adaptive integration where the number of quadrature points varies by observation based on the precision of factor score estimates. When factor scores are well-determined (small SE), fewer integration points are used. Importance sampling weights are computed automatically.

**Usage**

```
set_adaptive_quadrature_cpp(
  fm_ptr,
  factor_scores,
  factor_ses,
  factor_vars,
```

```

    threshold = 0.5,
    max_quad = 16L,
    verbose = TRUE
  )

```

### Arguments

fm_ptr	External pointer to FactorModel object
factor_scores	Matrix (n_obs x n_factors) of factor score estimates
factor_ses	Matrix (n_obs x n_factors) of standard errors
factor_vars	Vector (n_factors) of factor variances from previous stage
threshold	Threshold for determining quadrature points (default 0.5, matching legacy)
max_quad	Maximum quadrature points per factor (default 16)
verbose	Whether to print summary of adaptive quadrature setup (default TRUE)

### Value

No return value. Called for its side effect of enabling adaptive quadrature on the FactorModel pointed to by fm\_ptr and (when verbose = TRUE) printing a summary of the per-observation integration-point distribution.

---

set\_observation\_weights\_cpp

*Set observation weights for weighted likelihood estimation*

---

### Description

Sets per-observation weights for the likelihood calculation. When weights are set, each observation's contribution to the log-likelihood is multiplied by its weight. This is used for importance sampling in adaptive integration.

### Usage

```
set_observation_weights_cpp(fm_ptr, weights)
```

### Arguments

fm_ptr	External pointer to FactorModel object
weights	Numeric vector of observation weights (length = n_obs)

### Value

No return value. Called for its side effect of setting the per-observation likelihood weights on the FactorModel pointed to by fm\_ptr.

---

```
summary.factorana_result
```

*Summary method for factorana\_result objects*

---

### Description

Summary method for factorana\_result objects

### Usage

```
## S3 method for class 'factorana_result'
summary(object, ...)
```

### Arguments

object	A factorana_result object from estimate_model_rcpp()
...	Additional arguments (ignored)

### Value

A summary object with formatted tables

---

```
write_model_config_csv
```

*Write a single CSV with all configuration rows*

---

### Description

Write a single CSV with all configuration rows

### Usage

```
write_model_config_csv(model_system, factor_model, estimation_control, file)
```

### Arguments

model_system	a model_system object
factor_model	a factor_model object
estimation_control	an estimation_control object
file	Path to CSV to write. Required; pass an explicit path (use tempdir() in examples or tests).

### Value

Invisibly returns the data frame of configuration rows that was written to file (columns section, component, key, value, dtype). Called primarily for its side effect of writing a CSV.

# Index

`build_dynamic_previous_stage`, [2](#), [7](#)

`cleanup_parallel_workers`, [3](#)  
`components_table`, [4](#)  
`components_to_latex`, [5](#)

`define_dynamic_measurement`, [3](#), [6](#)  
`define_estimation_control`, [8](#)  
`define_factor_model`, [9](#)  
`define_model_component`, [10](#)  
`define_model_system`, [2](#), [3](#), [12](#)  
`disable_adaptive_quadrature_cpp`, [14](#)

`estimate_and_write`, [14](#)  
`estimate_factorscores_rcpp`, [15](#)  
`estimate_model`, [17](#)  
`estimate_model_rcpp`, [7](#), [18](#)  
`evaluate_factorscore_likelihood_cpp`,  
[20](#)  
`evaluate_likelihood_cpp`, [21](#)  
`evaluate_loglik_only_cpp`, [22](#)  
`extract_free_params_cpp`, [22](#)

`fix_coefficient`, [23](#)  
`fix_type_intercepts`, [24](#)

`gauss_hermite_quadrature`, [25](#)  
`get_parameter_info_cpp`, [26](#)

`initialize_factor_model_cpp`, [26](#)  
`initialize_parameters`, [27](#)

`print.components_table`, [28](#)  
`print.factorana_factorscores`, [28](#)  
`print.factorana_result`, [29](#)  
`print.factorana_table`, [29](#)  
`print.model_component`, [30](#)  
`print.summary.factorana_result`, [30](#)

`results_table`, [31](#)  
`results_to_latex`, [32](#)

`set_adaptive_quadrature_cpp`, [33](#)  
`set_observation_weights_cpp`, [34](#)  
`summary.factorana_result`, [35](#)

`write_model_config_csv`, [35](#)