Package: medsim (via r-universe)

June 12, 2026

Type Package

Title Simulation Infrastructure for Mediation Analysis

Version 0.2.1.9000

Maintainer Davood Tofighi <dtofighi@gmail.com>

Description Provides standardized infrastructure for conducting Monte Carlo simulation studies in mediation analysis. Features include environment-aware execution (local vs HPC cluster), parallel processing with progress reporting, ground truth caching, automated result analysis, and publication-ready visualization. Designed to work seamlessly with the mediationverse ecosystem: 'medfit', 'probmed', 'RMediation', and 'medrobust'.

License GPL (>= 3)

URL https://github.com/Data-Wise/medsim, https://data-wise.github.io/medsim/

BugReports https://github.com/Data-Wise/medsim/issues

Depends R (>= 4.1.0)

Imports parallel, pbapply, ggplot2, dplyr

Suggests medfit (>= 0.1.0), RMediation, mice, mitml, mvtnorm, MASS, tictoc, here, renv, pak, testthat (>= 3.0.0), knitr, rmarkdown, quarto, covr, withr, gridExtra, patchwork, RColorBrewer

Remotes Data-Wise/medfit

Config/Needs/website pkgdown, quarto

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

VignetteBuilder quarto

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3
Repository https://data-wise.r-universe.dev
Date/Publication 2026-06-12 00:22:56 UTC
RemoteUrl https://github.com/Data-Wise/medsim
RemoteRef dev
RemoteSha fe5e50ea79ddc6c49245e7f039ca72474697c112

Contents

`medsim_amputate` *Insert missing values under MCAR / MAR / MNAR*

Description

Thin, documented amputer aligning `mice::ampute` semantics to the medsim DGM contract. Returns the input frame with `NA`s inserted in `target` column(s); column names/order are preserved (DGM data contract).

Usage

```
medsim_amputate(
  data,
  target,
  mechanism = c("MCAR", "MAR", "MNAR"),
  prop = 0.2,
  predictors = NULL,
  weights = NULL,
  type = c("RIGHT", "LEFT", "MID", "TAIL")
)
```

Arguments

`data`	data.frame with columns `X, M, Y` (+ optional covariates).
`target`	Character vector of column(s) to set `NA` (e.g. `"M"`, or `c("M","Y")`).
`mechanism`	One of `"MCAR"`, `"MAR"`, `"MNAR"`. MCAR: constant prob; MAR: logistic on observed `predictors`; MNAR: logistic including the `target` itself (self-mechanism).

`prop`	Target overall missingness proportion in `target`.
`predictors`	Columns driving missingness for MAR/MNAR (defaults to all non-`target`).
`weights`	Optional named weights for the missingness logistic.
`type`	Amputation tail type, `mice::ampute`-style: one of `"RIGHT","LEFT","MID","TAIL"`.

Value

`data` with `NA`s inserted in `target`; names/order preserved.

`medsim_analyze` *Analyze Simulation Results*

Description

Computes comprehensive accuracy metrics for simulation results including bias, mean absolute error (MAE), root mean squared error (RMSE), and relative efficiency.

Usage

```
medsim_analyze(results, metrics = "all", by_scenario = TRUE)
```

Arguments

`results`	A medsim_results object from `medsim_run()`
`metrics`	Character vector: which metrics to compute. Options: • "bias": Mean estimation error • "mae": Mean absolute error • "rmse": Root mean squared error • "median_ae": Median absolute error • "max_ae": Maximum absolute error • "relative_bias": Bias as percentage of truth • "all": Compute all metrics (default)
`by_scenario`	Logical: Compute metrics separately for each scenario (default TRUE)

Details

Metrics Computed:

For each estimated parameter with known ground truth:

- **Bias**: Mean estimation error (estimate - truth)
- **MAE**: Mean absolute error, robust to outliers
- **RMSE**: Root mean squared error, penalizes large errors
- **Median AE**: Median absolute error, very robust
- **Max AE**: Maximum absolute error, identifies worst case
- **Relative Bias**: Bias as percentage of true value

Interpretation:

- Bias close to 0: unbiased estimator
- MAE/RMSE small: accurate estimator
- RMSE > MAE: presence of outliers/large errors
- Relative bias: standardized bias metric

Value

A list with class "medsim_analysis" containing:

- `accuracy`: data.frame with accuracy metrics
- `by_scenario`: data.frame with metrics by scenario (if by_scenario=TRUE)
- `summary`: Overall summary statistics

See Also

`medsim_analyze_coverage()`, `medsim_compare_methods()`

Examples

```
## Not run:
# Run simulation
results <- medsim_run(method, scenarios, config)

# Analyze accuracy
analysis <- medsim_analyze(results)

# View overall metrics
print(analysis$accuracy)

# View by-scenario metrics
print(analysis$by_scenario)

# Custom metrics
analysis <- medsim_analyze(
  results,
  metrics = c("bias", "mae", "rmse"),
  by_scenario = FALSE
)

## End(Not run)
```

`medsim_analyze_coverage` *Analyze Coverage Rates*

Description

Computes coverage rates for confidence intervals from simulation results. Requires that the simulation method returns confidence interval bounds (e.g., `ci_lower`, `ci_upper`).

Usage

```
medsim_analyze_coverage(
  results,
  ci_levels = c(0.9, 0.95, 0.99),
  ci_suffix = "_ci",
  by_scenario = TRUE
)
```

Arguments

`results`	A medsim_results object from `medsim_run()`
`ci_levels`	Numeric vector: nominal confidence levels to check (default: c(0.90, 0.95, 0.99))
`ci_suffix`	Character: suffix for CI columns (default: "_ci") E.g., if parameter is "indirect", looks for "indirect_ci_lower" and "indirect_ci_upper"
`by_scenario`	Logical: Compute coverage separately for each scenario (default TRUE)

Details

Coverage Interpretation:
Coverage rate is the proportion of confidence intervals that contain the true parameter value. For a 95% CI, expect ~95% coverage in large samples.

- Coverage < nominal: CI too narrow (anti-conservative)
- Coverage nominal: CI has correct width
- Coverage > nominal: CI too wide (conservative)

Column Naming Conventions:
The function looks for columns named:

- `{parameter}_ci_lower` and `{parameter}_ci_upper` (default)
- Or custom suffix: `{parameter}{ci_suffix}_lower` and `{parameter}{ci_suffix}_upper`

Value

A list with class "medsim_coverage" containing:

- `coverage`: data.frame with coverage rates
- `by_scenario`: data.frame with coverage by scenario (if by_scenario=TRUE)
- `summary`: Overall summary statistics

See Also

medsim_analyze(), medsim_analyze_power()

Examples

```
## Not run:
# Method that returns CIs
my_method <- function(data, params) {
  # ... estimation
  list(
    indirect = estimate,
    indirect_ci_lower = ci[1],
    indirect_ci_upper = ci[2]
  )
}

results <- medsim_run(my_method, scenarios, config)

# Analyze coverage
coverage <- medsim_analyze_coverage(results)

print(coverage$coverage)
#>   parameter  coverage_90  coverage_95  coverage_99
#>   indirect   0.902        0.951        0.991

## End(Not run)
```

---

medsim_analyze_power *Analyze Statistical Power*

Description

Computes empirical power (rejection rate under alternative hypothesis) from simulation results. Requires that the simulation method returns p-values.

Usage

```
medsim_analyze_power(
  results,
  alpha = 0.05,
  p_suffix = "_p",
  null_value = 0,
  by_scenario = TRUE
)
```

Arguments

`results`	A medsim_results object from `medsim_run()`
`alpha`	Numeric: significance level (default: 0.05)
`p_suffix`	Character: suffix for p-value columns (default: "_p") E.g., if parameter is "indirect", looks for "indirect_p"
`null_value`	Numeric: null hypothesis value (default: 0)
`by_scenario`	Logical: Compute power separately for each scenario (default TRUE)

Details

Power Interpretation:

Power is the probability of correctly rejecting the null hypothesis when it is false (i.e., detecting a true effect).

- Power = proportion of replications where p < alpha
- Higher power = better ability to detect effects
- Power depends on: effect size, sample size, alpha level

Notes:

This function computes **empirical power** from simulations, not theoretical power. For scenarios where the null is true (e.g., indirect effect = 0), the rejection rate represents Type I error rate, not power.

Value

A list with class "medsim_power" containing:

- `power`: data.frame with power rates
- `by_scenario`: data.frame with power by scenario (if by_scenario=TRUE)
- `summary`: Overall summary statistics

See Also

`medsim_analyze()`, `medsim_analyze_coverage()`

Examples

```
## Not run:
# Method that returns p-values
my_method <- function(data, params) {
  # ... estimation and testing
  list(
    indirect = estimate,
    indirect_p = p_value
  )
}

results <- medsim_run(my_method, scenarios, config)
```

```
# Analyze power
power <- medsim_analyze_power(results, alpha = 0.05)

print(power$power)
#>   parameter  power    n_valid
#>   indirect   0.847    1000

## End(Not run)
```

`medsim_cache_clear` *Clear Cache*

Description

Deletes cache files from a directory. Can delete all files or files matching a pattern.

Usage

```
medsim_cache_clear(
  cache_dir,
  pattern = "*.rds",
  max_age = NULL,
  confirm = TRUE
)
```

Arguments

`cache_dir`	Character: Directory containing cache files
`pattern`	Character: Optional pattern to match files (e.g., "truth_*.rds")
`max_age`	Numeric: Delete only files older than this many days. If NULL, deletes all matching files.
`confirm`	Logical: Ask for confirmation before deleting (default TRUE)

Value

Invisibly returns number of files deleted

Examples

```
## Not run:
# Clear all cache
medsim_cache_clear("cache")

# Clear old cache only
medsim_cache_clear("cache", max_age = 30)

# Clear specific pattern
```

```
medsim_cache_clear("cache", pattern = "truth_*.rds")

# Without confirmation
medsim_cache_clear("cache", confirm = FALSE)

## End(Not run)
```

`medsim_cache_exists` *Check if Cache Exists*

Description

Checks if a cache file exists and is readable.

Usage

```
medsim_cache_exists(file)
```

Arguments

`file`	Character: Path to cache file

Value

Logical: TRUE if cache exists and is readable, FALSE otherwise

Examples

```
## Not run:
if (medsim_cache_exists("cache/truth.rds")) {
  truth <- medsim_cache_load("cache/truth.rds")
} else {
  truth <- list(indirect = 0.09)  # Compute your truth values
  medsim_cache_save(truth, "cache/truth.rds")
}

## End(Not run)
```

`medsim_cache_info` *Get Cache Info*

Description

Retrieves metadata about a cached object without loading the full object.

Usage

```
medsim_cache_info(file)
```

Arguments

`file`	Character: Path to cache file

Value

List with cache metadata:

- `exists`: Logical - file exists
- `size_mb`: Numeric - file size in MB
- `modified`: POSIXct - last modified time
- `age_days`: Numeric - age in days
- `timestamp`: POSIXct - creation time (if available)
- `r_version`: Character - R version used (if available)

Examples

```
# Save a cached object to a temp file, then inspect it
cache_file <- tempfile(fileext = ".rds")
medsim_cache_save(list(value = 42), cache_file)
info <- medsim_cache_info(cache_file)
print(info)
unlink(cache_file)
```

medsim_cache_init *Initialize Cache Directory*

Description

Creates a cache directory with proper structure and a README file explaining the cache system.

Usage

```
medsim_cache_init(cache_dir, create_readme = TRUE)
```

Arguments

cache_dir	Character: Path to cache directory
create_readme	Logical: Create README.md explaining cache (default TRUE)

Value

Invisibly returns cache directory path

Examples

```
cache_dir <- file.path(tempdir(), "medsim_cache_example")
medsim_cache_init(cache_dir)
unlink(cache_dir, recursive = TRUE)
```

medsim_cache_list *List Cache Files*

Description

Lists all cache files in a directory with their metadata.

Usage

```
medsim_cache_list(cache_dir, pattern = "*.rds")
```

Arguments

cache_dir	Character: Directory containing cache files
pattern	Character: Pattern to match files (default "*.rds")

Value

data.frame with columns:

- `file`: File name
- `path`: Full path
- `size_mb`: Size in MB
- `age_days`: Age in days
- `modified`: Last modified time

Examples

```
# Create a small cache in a temp directory, then list it
cache_dir <- file.path(tempdir(), "medsim_cache_list_example")
dir.create(cache_dir, showWarnings = FALSE)
medsim_cache_save(list(a = 1), file.path(cache_dir, "result1.rds"))
medsim_cache_save(list(b = 2), file.path(cache_dir, "result2.rds"))

cache_list <- medsim_cache_list(cache_dir)
print(cache_list)

unlink(cache_dir, recursive = TRUE)
```

`medsim_cache_load` *Load Object from Cache*

Description

Loads a previously cached R object. Returns NULL if cache file doesn't exist.

Usage

```
medsim_cache_load(file, check_expiry = FALSE, max_age = 30, verbose = TRUE)
```

Arguments

`file`	Character: Path to cache file
`check_expiry`	Logical: Check if cache has expired (default FALSE)
`max_age`	Numeric: Maximum age of cache in days (default 30). Only used if `check_expiry = TRUE`.
`verbose`	Logical: Print messages about cache status (default TRUE)

Details

Cache Validation:

When loading, the cache is checked for:

- File exists
- File is readable
- Age is within max_age (if check_expiry = TRUE)

Handling Missing Cache:

If cache doesn't exist:

- Returns NULL (no error)
- Calling code should compute and cache the result

Example pattern:

```
result <- medsim_cache_load("cache/truth.rds")
if (is.null(result)) {
  result <- expensive_computation()
  medsim_cache_save(result, "cache/truth.rds")
}
```

Value

Cached object if file exists and is valid, NULL otherwise

See Also

medsim_cache_save(), medsim_cache_exists()

Examples

```
## Not run:
# Load from cache
truth <- medsim_cache_load("cache/truth_scenario1.rds")

if (is.null(truth)) {
  # Cache miss - compute and cache
  truth <- compute_expensive_truth()
  medsim_cache_save(truth, "cache/truth_scenario1.rds")
}

# Load with expiry check
result <- medsim_cache_load(
  "cache/old_result.rds",
  check_expiry = TRUE,
  max_age = 7  # 7 days
)

## End(Not run)
```

`medsim_cache_save` *Save Object to Cache*

Description

Saves an R object to a cache file for later retrieval. Uses RDS format for efficient storage and compression.

Usage

```
medsim_cache_save(object, file, compress = TRUE, overwrite = TRUE)
```

Arguments

`object`	Any R object to cache
`file`	Character: Path to cache file. Parent directory will be created if it doesn't exist.
`compress`	Logical: Compress the cached object (default TRUE). Can significantly reduce file size for large objects.
`overwrite`	Logical: Overwrite existing cache file (default TRUE)

Details

Cache File Format:

Cache files are stored in RDS format with metadata:

- Original object
- Timestamp of creation
- R version used
- Package version (if available)

File Organization:

Recommended cache directory structure:

```
cache/
|-- truth_scenario_1.rds
|-- truth_scenario_2.rds
+-- ...
```

Value

Invisibly returns the file path

See Also

`medsim_cache_load()`, `medsim_cache_clear()`

Examples

```
## Not run:
# Cache simulation truth
truth <- compute_expensive_truth()
medsim_cache_save(truth, "cache/truth_scenario1.rds")

# Cache with metadata
result <- list(value = 42, timestamp = Sys.time())
medsim_cache_save(result, "cache/my_result.rds")

## End(Not run)
```

---

medsim_check_results *Check Results for Errors*

Description

Checks a list of results from parallel execution for errors and provides a summary.

Usage

```
medsim_check_results(results, stop_on_error = FALSE)
```

Arguments

`results`	List of results from `medsim_run_parallel()`
`stop_on_error`	Logical: If TRUE, stops with error if any task failed

Value

Invisibly returns indices of failed tasks

Examples

```
## Not run:
results <- medsim_run_parallel(...)
medsim_check_results(results)

## End(Not run)
```

medsim_compare_configs

Compare Multiple Configurations

Description

Creates a comparison table showing differences between test, local, and cluster configurations.

Usage

```
medsim_compare_configs()
```

Value

A data.frame with configuration comparison

Examples

```
comparison <- medsim_compare_configs()
print(comparison)
```

medsim_compare_methods

Compare Multiple Methods

Description

Compares performance of multiple estimation methods across scenarios. Combines results from multiple simulation runs and computes relative performance metrics.

Usage

```
medsim_compare_methods(..., metrics = "all")
```

Arguments

...	Named medsim_results objects (e.g., method1 = results1, method2 = results2)
metrics	Character vector: which comparison metrics to compute. Options: "accuracy", "timing", "coverage", "all" (default)

Details

Comparison Metrics:

- **Accuracy**: MAE, RMSE, bias across methods
- **Timing**: Mean/median execution time
- **Coverage**: CI coverage rates (if CIs available)
- **Relative Efficiency**: Ratio of MSE (lower = better)

Notes:

All results must:

- Use the same scenarios (matched by name)
- Use the same configuration (n_replications, seed)
- Have ground truth available (for accuracy comparison)

Value

A list with class "medsim_comparison" containing:

- `accuracy_comparison`: data.frame comparing accuracy metrics
- `timing_comparison`: data.frame comparing timing
- `coverage_comparison`: data.frame comparing coverage (if applicable)
- `summary`: Overall comparison summary

See Also

`medsim_analyze()`, `medsim_analyze_coverage()`

Examples

```
## Not run:
# Run multiple methods
results_proposed <- medsim_run(method_proposed, scenarios, config)
results_delta <- medsim_run(method_delta, scenarios, config)
results_boot <- medsim_run(method_boot, scenarios, config)

# Compare
comparison <- medsim_compare_methods(
  proposed = results_proposed,
  delta = results_delta,
  bootstrap = results_boot
)

# View accuracy comparison
print(comparison$accuracy_comparison)

# View timing comparison
print(comparison$timing_comparison)

## End(Not run)
```

`medsim_config` *Create Simulation Configuration*

Description

Creates a configuration object for simulation studies. Automatically detects whether running on local machine or HPC cluster and adjusts parameters accordingly.

Usage

```
medsim_config(
  mode = "auto",
  n_replications = NULL,
  n_cores = NULL,
  scenarios = NULL,
  output_dir = NULL,
  seed = 12345,
  ...
)
```

Arguments

`mode`	Character: "auto", "test", "local", or "cluster" • "auto": Detect based on environment variables (SLURM, PBS, LSF) • "test": Quick validation (~30 seconds) • "local": Development on local machine (~15 minutes) • "cluster": Production on HPC cluster (hours)
`n_replications`	Integer: Number of Monte Carlo replications. If NULL, uses mode defaults (test=20, local=100, cluster=1000)
`n_cores`	Integer: Number of CPU cores for parallel processing. If NULL, autodetects (all cores - 2 on local, SLURM_CPUS_PER_TASK on cluster)
`scenarios`	Character: "all" or "test". Use "test" for single challenging scenario during development
`output_dir`	Character: Directory for saving results
`seed`	Integer: Random seed for reproducibility
`...`	Additional custom parameters

Details

Execution Modes:

Mode	Replications	Cores	Runtime	Use Case
test	20	4	~30s	Quick validation
local	100	auto	~15m	Development

cluster 1000 SLURM hours Production

Environment Detection:

When mode = "auto", checks for:

- SLURM_JOB_ID (SLURM scheduler)
- PBS_JOBID (PBS/Torque scheduler)
- LSB_JOBID (LSF scheduler)

If any are found, uses "cluster" mode. Otherwise, uses "local" mode.

Custom Parameters:

You can add custom parameters via `...`:

```
config <- medsim_config(
  mode = "local",
  n_bootstrap = 5000,
  alpha = 0.05,
  custom_param = "value"
)
```

Value

A list with simulation configuration parameters

See Also

`medsim_detect_environment()`, `print.medsim_config()`

Examples

```
# Auto-detect environment
config <- medsim_config(mode = "auto")

# Explicit test mode for quick validation
config_test <- medsim_config(mode = "test")

# Local mode with custom replications
config_local <- medsim_config(
  mode = "local",
  n_replications = 500
)

# Cluster mode (auto-detects SLURM cores)
config_cluster <- medsim_config(mode = "cluster")

# Custom parameters
config_custom <- medsim_config(
  mode = "local",
  n_bootstrap = 1000,
  ci_level = 0.95
```

```
)

# Print configuration
print(config)
```

`medsim_detect_cores` *Detect Number of Available Cores*

Description

Detects the number of CPU cores available for parallel processing. On HPC clusters, uses scheduler-allocated cores. On local machines, uses all cores minus 2 to avoid overwhelming the system.

Usage

```
medsim_detect_cores()
```

Details

On clusters, checks for:

- SLURM_CPUS_PER_TASK (SLURM)
- PBS_NUM_PPN (PBS/Torque)
- LSB_DJOB_NUMPROC (LSF)

On local machines, uses `parallel::detectCores() - 2`.

Value

Integer: Number of cores to use

Examples

```
n_cores <- medsim_detect_cores()
message(sprintf("Using %d cores", n_cores))
```

`medsim_detect_environment`
Detect Computing Environment

Description

Detects whether code is running on local machine or HPC cluster by checking for scheduler environment variables.

Usage

```
medsim_detect_environment()
```

Details

Checks for the following environment variables:

- SLURM_JOB_ID (SLURM scheduler)
- PBS_JOBID (PBS/Torque scheduler)
- LSB_JOBID (LSF scheduler)

If any are found, returns "cluster". Otherwise, returns "local".

Value

Character: "local" or "cluster"

Examples

```
env <- medsim_detect_environment()
if (env == "cluster") {
  message("Running on HPC cluster")
} else {
  message("Running on local machine")
}
```

`medsim_estimate_speedup`
Estimate Parallel Speedup

Description

Estimates the expected speedup from parallel processing based on number of cores and task characteristics.

Usage

```
medsim_estimate_speedup(n_tasks, n_cores, overhead = 0.1)
```

Arguments

`n_tasks`	Integer: Number of tasks
`n_cores`	Integer: Number of cores
`overhead`	Numeric: Overhead per core (default 0.1 = 10%)

Details

Speedup is estimated using Amdahl's law with overhead:

Speedup = n_cores / (1 + overhead * n_cores)

Efficiency = Speedup / n_cores

Value

Named list with:

- `speedup`: Expected speedup factor
- `efficiency`: Parallel efficiency (0-1)
- `recommendation`: Character recommendation

Examples

```
# For 100 tasks on 10 cores
est <- medsim_estimate_speedup(100, 10)
cat(sprintf("Expected speedup: %.1fx\n", est$speedup))
cat(sprintf("Efficiency: %.0f%%\n", est$efficiency * 100))
```

`medsim_figures` *Generate all standard figures from simulation results*

Description

Convenience wrapper that produces the standard set of medsim plots (coverage, error boxplot, timing) and writes them to `output_dir`.

Usage

```
medsim_figures(
  results,
  output_dir = "figures",
  format = "pdf",
  width = 8,
  height = 6,
  ...
)
```

Arguments

`results`	A `medsim_results` object from `medsim_run()`.
`output_dir`	Directory to write figures to. Created if it doesn't exist.
`format`	File extension passed to `ggplot2::ggsave()` (e.g. `"pdf"`, `"png"`).
`width, height`	Figure dimensions in inches.
`...`	Additional arguments passed to `ggplot2::ggsave()`.

Value

Invisibly, a named character vector of file paths written.

`medsim_get_optimal_cores` *Get Optimal Number of Cores*

Description

Determines the optimal number of cores to use for parallel processing based on system capabilities and task requirements.

Usage

```
medsim_get_optimal_cores(n_tasks, reserve = 2, max_cores = NULL)
```

Arguments

`n_tasks`	Integer: Number of tasks to process. If fewer tasks than cores, returns `n_tasks`.
`reserve`	Integer: Number of cores to reserve for system (default 2). Only applied on local machines, not clusters.
`max_cores`	Integer: Maximum number of cores to use. If NULL, uses all available (minus reserve).

Value

Integer: Optimal number of cores to use

Examples

```
# For 100 tasks
medsim_get_optimal_cores(n_tasks = 100)

# For few tasks (returns n_tasks)
medsim_get_optimal_cores(n_tasks = 3)

# With maximum limit
medsim_get_optimal_cores(n_tasks = 1000, max_cores = 8)
```

medsim_method_ipw *Thin IPW estimator adapter (robustness appendix)*

Description

Inverse-probability-weighting comparator. Positioned as a robustness appendix, not a co-equal main-results arm (decided 2026-06-11).

Usage

```
medsim_method_ipw(model, ...)
```

Arguments

`model`	Mediation model spec.
`...`	Passed to the underlying weighting / fitting routines.

Value

A `function(data, params)` returning the shared method contract (`branch_switch = NA`).

medsim_method_mbco_mi *MBCO-MI estimator adapter*

Description

Multiple imputation (via `mice`) + **D4-stacked MBCO** likelihood-ratio inference for the union null `H0: ab = 0`. Ports the validated `Missing Effect/code/prototype-d4-mbco.R` (phase-2, exact match vs `mitml::testModels(method = "D4")`): per-imputation MBCO LRTs are pooled with the Reiter/Chan-Meng D4 rule (Grund et al. 2021) into an F reference.

Usage

```
medsim_method_mbco_mi(model, m = 20L, ...)
```

Arguments

`model`	Mediation model spec (accepted for API symmetry; the estimator reads the `X`, `M`, `Y` (+ optional `C*`) columns of `data`).
`m`	Integer number of imputations.
`...`	Reserved for future MI / MBCO options.

Details

The point estimate is the Rubin-pooled `a*b`; the CI is a Monte-Carlo product interval on the pooled paths; the p-value is the D4-pooled MBCO test. With no missingness (or without `mice`) it degrades to the complete-case MBCO LRT with a chi-square reference — never to a Sobel normal approximation for the test.

Value

A `function(data, params)` returning the shared method contract (see file header).

`medsim_method_mc_ci` *Monte-Carlo CI estimator adapter*

Description

Multiple imputation (via `mice`) + Monte-Carlo confidence interval for the indirect effect (via `RMediation::medci()` when available, else a base-R product-of-normals draw on the Rubin-pooled paths).

Usage

```
medsim_method_mc_ci(model, m = 20L, ...)
```

Arguments

`model`	Mediation model spec (accepted for API symmetry; the estimator reads the `X`, `M`, `Y` (+ optional `C*`) columns of `data`).
`m`	Integer number of imputations.
`...`	Reserved for future MI / MBCO options.

Value

A `function(data, params)` returning the shared method contract (`branch_switch = NA`).

`medsim_plot_combined_panel` *Create Combined Multi-Panel Figure*

Description

Creates a combined figure with multiple panels for manuscript submission. Useful for showing multiple aspects of simulation results in one figure.

Usage

```
medsim_plot_combined_panel(
  results,
  panels = c("error", "timing"),
  layout = NULL,
  ...
)
```

Arguments

`results`	A medsim_results object or named list of results objects
`panels`	Character vector: which panels to include. Options: "error", "timing", "coverage" (default: c("error", "timing"))
`layout`	Character: layout specification like "2x1", "1x2", "2x2" (default: auto-determined from number of panels)
`...`	Additional arguments passed to individual plotting functions

Details

Requires either `patchwork` or `gridExtra` package for combining plots.

Value

A combined plot object (patchwork or gridExtra)

Examples

```
## Not run:
# Two-panel figure
p <- medsim_plot_combined_panel(
  results,
  panels = c("error", "timing"),
  layout = "1x2"
)

# Four-panel figure for manuscript
p <- medsim_plot_combined_panel(
  list(Proposed = res1, Delta = res2),
  panels = c("error", "timing", "coverage"),
  layout = "2x2"
)

## End(Not run)
```

`medsim_plot_coverage` *Plot Coverage Rates*

Description

Creates a plot showing confidence interval coverage rates across scenarios and methods.

Usage

```
medsim_plot_coverage(
  coverage,
  expected_coverage = 0.95,
  tolerance = 0.02,
  title = NULL,
  ...
)
```

Arguments

`coverage`	A medsim_coverage object from `medsim_analyze_coverage()`, or a named list of coverage objects for method comparison
`expected_coverage`	Numeric: expected coverage rate to show as reference line (default: 0.95)
`tolerance`	Numeric: tolerance band around expected coverage (default: 0.02)
`title`	Character: plot title (default: auto-generated)
`...`	Additional arguments passed to ggplot2::theme()

Value

A ggplot2 object

Examples

```
## Not run:
coverage <- medsim_analyze_coverage(results)
p <- medsim_plot_coverage(coverage, expected_coverage = 0.95)

## End(Not run)
```

`medsim_plot_error_boxplot` *Plot Error Distribution Boxplots*

Description

Creates boxplots showing the distribution of estimation errors across scenarios. Useful for comparing accuracy of different methods.

Usage

```
medsim_plot_error_boxplot(
  results,
  parameter = "indirect",
  log_scale = FALSE,
  color_palette = NULL,
  title = NULL,
  ...
)
```

Arguments

`results`	A medsim_results object from `medsim_run()`, or a list of named results objects for method comparison
`parameter`	Character: which parameter to plot (default: "indirect"). If NULL, plots all parameters with available ground truth.
`log_scale`	Logical: use log10 scale for y-axis (default: FALSE)
`color_palette`	Character: color palette name from RColorBrewer or "viridis" (default: NULL uses ggplot2 defaults)
`title`	Character: plot title (default: auto-generated)
`...`	Additional arguments passed to ggplot2::theme()

Details

Interpreting Boxplots:

- Box: IQR (25th to 75th percentile)
- Line in box: Median error
- Whiskers: 1.5 × IQR
- Points: Outliers beyond whiskers

Tighter boxes = more consistent estimates

Usage Patterns:

```
# Single method
p <- medsim_plot_error_boxplot(results)

# Compare methods
p <- medsim_plot_error_boxplot(list(
  Proposed = results1,
  Delta = results2,
  Bootstrap = results3
))

# Customize
p <- medsim_plot_error_boxplot(
  results,
  log_scale = TRUE,
  color_palette = "Set2"
)
p + ggplot2::theme_minimal()
```

Value

A ggplot2 object

Examples

```
## Not run:
results <- medsim_run(method, scenarios, config)
p <- medsim_plot_error_boxplot(results)
print(p)

# Save to file
ggplot2::ggsave("error_boxplot.pdf", p, width = 10, height = 6)

## End(Not run)
```

---

medsim_plot_timing *Plot Timing Comparison*

Description

Creates bar plots comparing computational time across methods and scenarios.

Usage

```
medsim_plot_timing(
  results,
  metric = "mean",
  log_scale = FALSE,
```

```
  title = NULL,
  ...
)
```

Arguments

results	A named list of medsim_results objects (required for comparison)
metric	Character: which timing metric to plot. Options: "mean", "median", "total" (default: "mean")
log_scale	Logical: use log10 scale for y-axis (default: FALSE)
title	Character: plot title (default: auto-generated)
...	Additional arguments passed to ggplot2::theme()

Value

A ggplot2 object

Examples

```
## Not run:
p <- medsim_plot_timing(list(
  Proposed = results1,
  Delta = results2,
  Bootstrap = results3
))

## End(Not run)
```

---

medsim_rnonnormal *Draw nonnormal values with a target marginal skew/kurtosis*

Description

Fleishman / Vale–Maurelli power-method draws. Pure base R; no hard dependency. Warns/errors on (skew, kurtosis) outside the Fleishman-feasible region.

Usage

```
medsim_rnonnormal(n, mean = 0, sd = 1, skew = 0, kurtosis = 0)
```

Arguments

n	Integer sample size.
mean, sd	Target marginal mean and SD (applied after standardizing).
skew	Target marginal skewness (third standardized moment).
kurtosis	Target marginal excess kurtosis (fourth standardized moment minus 3).

Value

Numeric vector of length `n` with the requested marginal moments.

`medsim_run` *Run Simulation Study*

Description

Executes a complete simulation study by running a user-defined method across multiple scenarios and replications. Supports parallel processing and automatic result aggregation.

Usage

```
medsim_run(
  method,
  scenarios,
  config,
  compute_truth = NULL,
  parallel = TRUE,
  verbose = TRUE
)
```

Arguments

`method`	Function: User-defined simulation method. Must accept two arguments: `data` (data.frame) and `params` (list). Should return a named list of results.
`scenarios`	List of scenario objects from `medsim_scenario()` or `medsim_scenarios_mediation()`
`config`	Configuration object from `medsim_config()`
`compute_truth`	Function: Optional function to compute ground truth. Takes same arguments as `method`. If NULL, no ground truth computed.
`parallel`	Logical: Use parallel processing (default TRUE)
`verbose`	Logical: Print progress messages (default TRUE)

Details

Method Function Requirements:

The `method` function must:

- Accept `data` (data.frame) as first argument
- Accept `params` (list) as second argument
- Return a named list with at least one numeric element

Example:

```
my_method <- function(data, params) {
  fit_m <- lm(M ~ X, data = data)
  fit_y <- lm(Y ~ X + M, data = data)

  a <- coef(fit_m)["X"]
  b <- coef(fit_y)["M"]

  list(
    indirect = a * b,
    a_path = a,
    b_path = b,
    se_indirect = ...,  # Optional
    ci_lower = ...,     # Optional
    ci_upper = ...      # Optional
  )
}
```

Parallel Processing:

When `parallel = TRUE`:

- Uses `medsim_run_parallel()` internally
- Number of cores from `config$n_cores`
- Progress bars shown (local) or suppressed (cluster)
- All necessary objects exported to workers

Ground Truth:

If `compute_truth` is provided:

- Computed once per scenario (cached automatically)
- Used to calculate errors/bias in results
- Can use Monte Carlo or analytical methods

Value

A medsim_results object (list) containing:

- `results`: data.frame with all simulation results
- `summary`: data.frame with summary statistics
- `truth`: data.frame with ground truth values (if compute_truth provided)
- `config`: configuration used
- `scenarios`: scenarios used
- `method_name`: name of method function
- `timestamp`: when simulation was run

See Also

`medsim_config()`, `medsim_scenario()`, `medsim_run_parallel()`

Examples

```
## Not run:
# Define method
my_method <- function(data, params) {
  fit_m <- lm(M ~ X, data = data)
  fit_y <- lm(Y ~ X + M, data = data)
  list(indirect = coef(fit_m)["X"] * coef(fit_y)["M"])
}

# Configure and run
config <- medsim_config("local")
scenarios <- medsim_scenarios_mediation()

results <- medsim_run(
  method = my_method,
  scenarios = scenarios,
  config = config
)

# View results
print(results)
summary(results)

## End(Not run)
```

`medsim_run_parallel` *Run Tasks in Parallel*

Description

Executes a list of tasks in parallel using multiple CPU cores. Provides progress bars, proper error handling, and automatic cluster management.

Usage

```
medsim_run_parallel(
  tasks,
  fun,
  n_cores = NULL,
  progress = TRUE,
  export = NULL,
  packages = NULL,
  cluster_type = NULL
)
```

Arguments

`tasks`	Vector or list: Tasks to process (e.g., 1:100, list of params)
`fun`	Function: Function to apply to each task. Should accept one argument (the task) and return a result.
`n_cores`	Integer: Number of CPU cores to use. If NULL, uses all available cores minus 2.
`progress`	Logical: Show progress bar (default TRUE). Automatically suppressed when running on cluster (batch jobs).
`export`	Character vector: Names of objects to export to workers. These objects must be available in the calling environment.
`packages`	Character vector: Names of packages to load on each worker. Use this if `fun` requires specific packages.
`cluster_type`	Character: Type of cluster ("PSOCK" or "FORK"). FORK is more efficient but only works on Unix. Auto-detected by default.

Details

Cluster Types:

- **PSOCK** (default on Windows): Creates separate R sessions. Requires explicit export of objects and packages. Works on all platforms.
- **FORK** (default on Unix): Copies current R session. More efficient, automatic object sharing, but Unix-only.

Progress Bars:

Progress bars are shown in interactive sessions and local execution, but suppressed on HPC clusters to avoid cluttering log files.

Error Handling:

If a task fails:

- Error is caught and returned as an error object
- Other tasks continue processing
- Check results with `sapply(results, inherits, "error")`

Memory Considerations:

Each worker process requires memory. For large tasks:

- Reduce `n_cores` if running out of memory
- Process tasks in chunks
- Use FORK cluster (Unix) which shares memory

Value

List: Results from applying `fun` to each task (same length as tasks)

See Also

`parallel::makeCluster()`, `parallel::parLapply()`, `pbapply::pblapply()`

Examples

```
## Not run:
# Simple parallel computation
results <- medsim_run_parallel(
  tasks = 1:100,
  fun = function(i) {
    Sys.sleep(0.1)  # Simulate work
    i^2
  },
  n_cores = 4
)

# With exports
my_data <- data.frame(x = 1:10, y = 11:20)

results <- medsim_run_parallel(
  tasks = 1:10,
  fun = function(i) {
    mean(my_data$x[1:i])
  },
  export = "my_data"
)

# With packages
results <- medsim_run_parallel(
  tasks = 1:10,
  fun = function(i) {
    dplyr::tibble(x = i, y = i^2)
  },
  packages = "dplyr"
)

# Check for errors
errors <- sapply(results, inherits, "error")
if (any(errors)) {
  cat("Errors occurred in:", which(errors), "\n")
}

## End(Not run)
```

`medsim_scenario` *Create Custom Simulation Scenario*

Description

Define a custom scenario for simulation studies. A scenario consists of a name, description, data generation function, and population parameters.

Usage

```
medsim_scenario(name, description = "", data_generator, params = list())
```

Arguments

`name`	Character: Descriptive name for the scenario
`description`	Character: Detailed description (optional)
`data_generator`	Function: Takes n (sample size) and returns data.frame
`params`	List: Known population parameters for validation

Details

Data Generator Function:

The data_generator function must:

- Accept `n` (sample size) as first argument
- Return a data.frame with at least: X (treatment), M (mediator), Y (outcome)
- Can include additional variables or covariates

Parameters List:

The params list should include known population values:

- `a`: X -> M path coefficient
- `b`: M -> Y path coefficient
- `c_prime`: X -> Y direct effect
- `indirect`: True indirect effect (a * b)
- Additional parameters as needed

Value

A scenario object (list with class "medsim_scenario")

See Also

`medsim_scenarios_mediation()` for standard scenarios

Examples

```
# Simple custom scenario
my_scenario <- medsim_scenario(
  name = "Large Effects",
  description = "Both paths have large effects",
  data_generator = function(n = 200) {
    X <- rnorm(n)
    M <- 0.7 * X + rnorm(n)
    Y <- 0.7 * M + rnorm(n)
    data.frame(X = X, M = M, Y = Y)
  },
  params = list(
```

```
    a = 0.7,
    b = 0.7,
    indirect = 0.49
  )
)

# Generate data
data <- my_scenario$data_generator(n = 100)

# Scenario with covariates
covariate_scenario <- medsim_scenario(
  name = "With Covariates",
  data_generator = function(n = 200) {
    C1 <- rnorm(n)
    C2 <- rbinom(n, 1, 0.5)
    X <- rnorm(n)
    M <- 0.3 * X + 0.2 * C1 + rnorm(n)
    Y <- 0.3 * M + 0.2 * C2 + rnorm(n)
    data.frame(X = X, M = M, Y = Y, C1 = C1, C2 = C2)
  },
  params = list(
    a = 0.3,
    b = 0.3,
    indirect = 0.09,
    gamma_m = 0.2,
    gamma_y = 0.2
  )
)
```

`medsim_scenario_missing` *Construct a missing-data mediation scenario*

Description

Wraps `medsim_scenario()` with a `data_generator` that (1) draws complete `X -> M -> Y` with optional nonnormal residuals (`medsim_rnonnormal()`), then (2) amputes `target` under `mechanism` (`medsim_amputate()`).

Usage

```
medsim_scenario_missing(
  name,
  true_params,
  mechanism,
  prop = 0.2,
  target = "M",
  nonnormal = NULL
)
```

Arguments

`name`	Scenario name.
`true_params`	List of true generating parameters: `a`, `b`, `cp`, residual SDs.
`mechanism`	One of `"MCAR"`, `"MAR"`, `"MNAR"`.
`prop`	Target missingness proportion.
`target`	Column(s) to make missing (default `"M"`).
`nonnormal`	`NULL` for normal residuals, or `list(skew=, kurtosis=)`.

Value

A `medsim_scenario` object.

`medsim_scenario_missing_grid`

Build the full factorial of missing-data scenarios

Description

Convenience: expand `SPEC-simulation-design` cells (mechanism × prop × n × effect × nonnormality) into a list of `medsim_scenario_missing()` objects in one call.

Usage

```
medsim_scenario_missing_grid(
  true_params_list,
  mechanisms,
  props,
  nonnormal_list = list(NULL)
)
```

Arguments

`true_params_list`
: List of `true_params` lists (one per effect-size cell).

`mechanisms`, `props`
: Character/numeric vectors crossed factorially.

`nonnormal_list`
: List of `nonnormal` specs (incl. `NULL` for normal).

Value

A list of `medsim_scenario` objects.

`medsim_scenarios_mediation` *Create Standard Mediation Scenarios*

Description

Creates a list of standard mediation scenarios for simulation studies. These scenarios cover common patterns in mediation analysis including independent paths, various correlation structures, suppression effects, and non-standard conditions.

Usage

```
medsim_scenarios_mediation()
```

Details

Standard Scenarios:

1. **Independent Paths**: No correlation between X, M, Y
2. **Moderate Correlation**: rho = 0.3 between all pairs
3. **High Correlation**: rho = 0.7 between all pairs
4. **Suppression**: Mixed positive and negative correlations
5. **Non-zero Effects**: Small to moderate true effects
6. **Unequal Variances**: Different residual variances

Each scenario generates data with:

- Sample size n (default: 200)
- Treatment X, Mediator M, Outcome Y
- Known population parameters for validation

Value

A list of scenario objects, each with name, description, and data_generator function

See Also

`medsim_scenario()` for creating custom scenarios

Examples

```
# Get all standard scenarios
scenarios <- medsim_scenarios_mediation()

# See scenario names
sapply(scenarios, function(s) s$name)

# Generate data from first scenario
data <- scenarios[[1]]$data_generator(n = 100)
head(data)
```

```
# Access scenario parameters
scenarios[[1]]$params
```

`medsim_summarize_branch_switch`
Summarize the MBCO branch-switch rate per scenario

Description

Reads the `branch_switch` column emitted by `medsim_method_mbco_mi()` (the union-null / ab=0 diagnostic from MEMO-MBCO-MI-derivation) and returns a per-scenario summary suitable for joining as a column onto a `medsim_table_coverage()` table. Non-converged rows (`converged == 0`) are dropped before summarizing.

Usage

```
medsim_summarize_branch_switch(results, by = "scenario")
```

Arguments

`results`	A `medsim_results` object from `medsim_run()`.
`by`	Grouping columns (default `"scenario"`).

Value

A data.frame: one row per group with `branch_switch_rate` and `n_valid`.

`medsim_table_accuracy`
Generate Accuracy Table

Description

Creates a publication-ready LaTeX table showing accuracy metrics (bias, MAE, RMSE) from simulation analysis results.

Usage

```
medsim_table_accuracy(
  analysis,
  digits = 3,
  metrics = c("bias", "mae", "rmse"),
  by_scenario = TRUE,
  caption = NULL,
  label = "tab:accuracy"
)
```

Arguments

analysis	A medsim_analysis object from medsim_analyze()
digits	Integer: number of decimal places (default: 3)
metrics	Character vector: which metrics to include. Options: "bias", "mae", "rmse", "median_ae", "max_ae", "relative_bias" (default: c("bias", "mae", "rmse"))
by_scenario	Logical: include scenario breakdown (default: TRUE)
caption	Character: table caption (default: auto-generated)
label	Character: LaTeX label for cross-referencing (default: "tab:accuracy")

Value

A medsim_table object (character vector with LaTeX code)

Examples

```
## Not run:
results <- medsim_run(method, scenarios, config)
analysis <- medsim_analyze(results)

# Generate table
table <- medsim_table_accuracy(analysis)
print(table)  # Preview

# Write to file
medsim_write_table(table, "tables/accuracy.tex")

## End(Not run)
```

---

medsim_table_comparison

Generate Method Comparison Table

Description

Creates a publication-ready LaTeX table comparing multiple methods across accuracy, timing, and coverage metrics.

Usage

```
medsim_table_comparison(
  comparison,
  metrics = c("accuracy", "timing"),
  caption = NULL,
  label = "tab:comparison"
)
```

Arguments

`comparison`	A medsim_comparison object from `medsim_compare_methods()`
`metrics`	Character vector: which metrics to include ("accuracy", "timing", "coverage")
`caption`	Character: table caption
`label`	Character: LaTeX label

Value

A medsim_table object

`medsim_table_coverage` *Generate Coverage Table*

Description

Creates a publication-ready LaTeX table showing confidence interval coverage rates.

Usage

```
medsim_table_coverage(
  coverage,
  expected = 0.95,
  by_scenario = TRUE,
  caption = NULL,
  label = "tab:coverage"
)
```

Arguments

`coverage`	A medsim_coverage object from `medsim_analyze_coverage()`
`expected`	Numeric: expected/nominal coverage rate (default: 0.95)
`by_scenario`	Logical: show coverage by scenario (default: TRUE)
`caption`	Character: table caption
`label`	Character: LaTeX label

Value

A medsim_table object

medsim_table_power *Generate Power Table*

Description

Creates a publication-ready LaTeX table showing empirical power (rejection rates) from simulation analysis.

Usage

```
medsim_table_power(
  power,
  by_scenario = TRUE,
  caption = NULL,
  label = "tab:power"
)
```

Arguments

power	A medsim_power object from medsim_analyze_power()
by_scenario	Logical: show power by scenario (default: TRUE)
caption	Character: table caption
label	Character: LaTeX label

Value

A medsim_table object

medsim_table_timing *Generate Timing Comparison Table*

Description

Creates a publication-ready LaTeX table comparing computational time across methods.

Usage

```
medsim_table_timing(
  results_list,
  metric = "mean",
  include_speedup = TRUE,
  reference_method = NULL,
  by_scenario = FALSE,
  caption = NULL,
  label = "tab:timing"
)
```

Arguments

`results_list`	Named list of medsim_results objects, one per method
`metric`	Character: timing metric to use ("mean", "median", "total")
`include_speedup`	Logical: include speedup column relative to slowest method
`reference_method`	Character: method name to use as reference for speedup (default: NULL uses slowest method)
`by_scenario`	Logical: show timing by scenario (default: FALSE)
`caption`	Character: table caption
`label`	Character: LaTeX label

Value

A medsim_table object

Examples

```
## Not run:
table <- medsim_table_timing(list(
  Proposed = results1,
  Delta = results2,
  Bootstrap = results3
))

## End(Not run)
```

`medsim_tables` *Generate all standard LaTeX tables from simulation results*

Description

Convenience wrapper around `medsim_tables_workflow()` with sensible defaults.

Usage

```
medsim_tables(results, output_dir = "tables", format = "latex", ...)
```

Arguments

`results`	A `medsim_results` object from `medsim_run()`.
`output_dir`	Directory to write tables to. Created if it doesn't exist.
`format`	Output format passed to `medsim_tables_workflow()` (e.g. `"latex"`, `"markdown"`).
`...`	Additional arguments passed to `medsim_tables_workflow()`.

Value

Invisibly, the result of `medsim_tables_workflow()`.

`medsim_tables_workflow` *Generate All Tables*

Description

Convenience function to generate all tables from simulation results and save them to an output directory.

Usage

```
medsim_tables_workflow(
  results,
  output_dir,
  tables = "all",
  format = "latex",
  prefix = "table_"
)
```

Arguments

`results`	A medsim_results object from `medsim_run()`
`output_dir`	Character: output directory for tables
`tables`	Character vector: which tables to generate ("accuracy", "coverage", "power", "all")
`format`	Character: output format ("latex", "csv", "markdown")
`prefix`	Character: prefix for file names (default: "table_")

Value

Invisibly returns a list of generated file paths

Examples

```
## Not run:
results <- medsim_run(method, scenarios, config)

# Generate all tables
files <- medsim_tables_workflow(
  results,
  output_dir = "manuscript/tables",
  format = "latex"
)

## End(Not run)
```

medsim_validate_scenario *Validate Scenario*

Description

Checks that a scenario object is valid and can generate data correctly.

Usage

```
medsim_validate_scenario(scenario, n = 10)
```

Arguments

scenario	A medsim_scenario object
n	Sample size to test (default: 10)

Value

TRUE if valid, throws error otherwise

Examples

```
scenarios <- medsim_scenarios_mediation()
medsim_validate_scenario(scenarios[[1]])
```

medsim_workflow *Generate analysis, figures, and tables from simulation results*

Description

End-to-end output pipeline: runs `medsim_analyze()`, `medsim_figures()`, and `medsim_tables()` and writes everything to `output_dir`.

Usage

```
medsim_workflow(
  results,
  output_dir = "results",
  figures_format = "pdf",
  tables_format = "latex",
  ...
)
```

Arguments

results	A medsim_results object from medsim_run().
output_dir	Top-level directory for all outputs. Subdirectories figures/ and tables/ are created underneath. Created if it doesn't exist.
figures_format	File format for figures (e.g. "pdf", "png").
tables_format	Output format for tables (e.g. "latex", "markdown").
...	Additional arguments passed to medsim_figures().

Value

Invisibly, a list with elements analysis, figures (file paths), and tables (return value of medsim_tables_workflow()).

Examples

```
## Not run:
config <- medsim_config("test")
scenarios <- medsim_scenarios_mediation()
results <- medsim_run(my_method, scenarios, config)
medsim_workflow(results, output_dir = "results")

## End(Not run)
```

---

medsim_write_table *Write Table to File*

Description

Writes a medsim_table object to a file in LaTeX, CSV, or Markdown format.

Usage

```
medsim_write_table(table, file, format = NULL)
```

Arguments

table	A medsim_table object
file	Character: output file path
format	Character: output format ("latex", "csv", "markdown") If NULL, inferred from file extension.

Value

Invisibly returns the file path

Examples

```
## Not run:
table <- medsim_table_accuracy(analysis)

# Write LaTeX
medsim_write_table(table, "tables/accuracy.tex")

# Write CSV
medsim_write_table(table, "tables/accuracy.csv", format = "csv")

## End(Not run)
```

---

print.medsim_analysis

Print Analysis Results

Description

Print Analysis Results

Usage

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

Arguments

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

Value

Invisibly returns x

---

print.medsim_comparison

Print Method Comparison

Description

Print Method Comparison

Usage

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

Arguments

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

Value

Invisibly returns x

---

print.medsim_config *Print Configuration Summary*

Description

Print Configuration Summary

Usage

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

Arguments

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

Value

Invisibly returns x

---

print.medsim_coverage *Print Coverage Results*

Description

Print Coverage Results

Usage

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

Arguments

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

Value

Invisibly returns x

---

print.medsim_power *Print Power Results*

Description

Print Power Results

Usage

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

Arguments

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

Value

Invisibly returns x

---

print.medsim_results *Print Simulation Results*

Description

Print Simulation Results

Usage

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

Arguments

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

Value

Invisibly returns x

print.medsim_scenario *Print Scenario Summary*

Description

Print Scenario Summary

Usage

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

Arguments

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

Value

Invisibly returns x

print.medsim_table *Print medsim_table*

Description

Print medsim_table

Usage

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

Arguments

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

Value

Invisibly returns x

`summary.medsim_results` *Summarize Simulation Results*

Description

Summarize Simulation Results

Usage

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

Arguments

`object`	A medsim_results object
`...`	Additional arguments (ignored)

Value

Summary data.frame

Index