# AnalysesFunction

Tathagata Chattopadhyay

January 2026

## 1 Unit Test Coverage Report

This section documents how the unit tests you wrote exercise each function. For every function, we list the behaviours/branches that are explicitly covered, and where helpful, we also note relevant gaps.

Throughout, test names refer to the `test_that()` descriptions in your test file.

### 1.1 `applicablePreviousTrials`

**Main responsibilities**

- Check that previous analyses exist and have the right structure.
- Check that method names, quantiles and number of cohorts agree with the new analysis.
- Optionally check that all required `p_diff_*` columns (from `calc_differences`) are present in the stored posterior quantiles.
- Return a single logical flag indicating whether previous trials can be reused.

**Covered behaviours**

- **Happy path, no differences** – test `"applicablePreviousTrials returns TRUE when all conditions are met (no differences)"`:
  - Previous analyses exist (via `continueRecruitment()`).
  - Method names in `scenario_list` and input `method_names` agree.
  - Quantiles and number of cohorts match.
  - `calc_differences = NULL` so the function takes the "no differences" branch.
  - Return value is a logical scalar `TRUE`.

- **Happy path with differences** – test `"applicablePreviousTrials returns TRUE when required diff columns exist"`:
  - `calc_differences` is a matrix and the corresponding `p_diff_*` columns exist in all relevant `post_quantiles`.
  - The function checks for required `p_diff_*` columns and returns `TRUE`.
- **Missing `p_diff_*` column** – test `"applicablePreviousTrials returns FALSE when required diff columns are missing"`:
  - A required difference column is manually removed from `previous_analyses`*post_quantiles*.
  - `The function detects the missing p_diff_* column and returns FALSE.`
- **Different method names across scenarios** – test `"applicablePreviousTrials returns FALSE when method_names differ across scenarios"`:
  - Two scenarios are created where the second has different method names in `previous_analyses`*post_quantiles*(*prefixedwith*`alt_`).
  - `The function detects inconsistent method names across scenarios and returns FALSE.`

## 1.2 `performAnalyses`

**Main responsibilities**

- Orchestrate Bayesian analyses for each scenario and each method.
- Build and store common analysis parameters: method names, quantiles, prior parameters.
- Call lower-level posterior and quantile computation routines.

**Covered behaviours**

- **Basic structure of `analysis_list`** – test `"performAnalyses returns a well-formed analysis_list"`:
  - Return object has class `analysis_list`.
  - Length equals number of input scenarios.
  - Names follow the `"scenario_X"` convention.
  - Each scenario entry contains `quantiles_list`, `scenario_data`, and `analysis_parameters`.
  - `scenario_data` is identical to the input scenario.

- **Sorting of** `method_names` – test `"performAnalyses sorts method_names and stores them in analysis_parameters"`:
  - The function sorts `method_names` alphabetically inside `analysis_parameters`.
  - The names of `quantiles_list` match this sorted order.
- **Quantiles construction** – test `"performAnalyses constructs quantiles from defaults and evidence_levels"`:
  - The function combines a default set of quantiles and `evidence_levels` via `1 - c(defaults, evidence_levels)`.
  - Values are rounded, made unique, and sorted.
  - The resulting vector contains all expected quantiles.
- **Automatic prior generation** – test `"performAnalyses fills prior_parameters_list when not supplied"`:
  - When `prior_parameters_list = NULL`, the function calls `getPriorParameters()`.
  - The resulting list is stored in `analysis_parameters`*prior_parameters_listandcontainsentriesforall*
- **Verbose progress message** – test `"performAnalyses prints a progress message when verbose = TRUE"`:
  - The branch printing "`Performing Analyses`" is executed when `verbose = TRUE`, and the message is emitted.

## 1.3 `performJags`

### Main responsibilities

- Wrap `rjags` to run JAGS models and return posterior samples as a matrix.

### Covered behaviours

- Test `"performJags runs a simple Bernoulli model and returns a sensible sample matrix"`:
  - Model file handling: JAGS model read from a temporary `.bug` file.
  - Data passing: `data` list with `N` and `y`.
  - Iterations and burn-in: uses `n_chains`, `n_iter`, `n_burnin`.
  - Output structure: returns a matrix with correct number of rows (`n_chains * (n_iter - n_burnin)`) and correct column names.
  - Basic sanity: posterior mean lies in $(0, 1)$ for a Bernoulli parameter.

### 1.4 `getPosteriors`

**Main responsibilities**

- Call `performJags()` and post-process the resulting sample matrix.
- Clean column names (remove `[ ]`, internal indices).
- For EXNEX models, aggregate JAGS exchangeability weights into `w_j` columns.

**Covered behaviours**

- **Berry case, no weights** – test `"getPosteriors: basic posterior sampling and name cleaning (no exch weights)"`:
  - Real Berry JAGS model is prepared via `prepareAnalysis()`.
  - `r` and `n` are populated from a small `simulateScenarios` dataset.
  - The function returns a numeric matrix with only finite values.
  - Square brackets are removed from column names.
  - No `w_` or `exch` columns appear, confirming that the weights-handling branch is skipped for Berry.
- **EXNEX case, with weights** – test `"getPosteriors: exNEX exchangeability weights are renamed to w_j and extras dropped"`:
  - EXNEX JAGS model is prepared and run.
  - Output still a numeric matrix with finite entries.
  - No `"exch"` substring remains in column names.
  - There is one `w_j` column per cohort.
  - JAGS-style index `",1"` is removed from all column names.

### 1.5 `getPostQuantiles`

**Main responsibilities**

- For each trial realisation, compute posterior quantiles (and mean/SD) for:
  - Pooled Beta model.
  - Stratified (per-cohort) Beta model.
  - (Potentially) BHM methods via JAGS (not explicitly tested here).
- Optionally compute difference parameters `p_diff_*`.

**Covered behaviours**

- **Pooled Beta backend** – test "getPostQuantiles (pooled): uses pooled backend and matches Beta posterior":
  - Uses `foreach::registerDoSEQ()` to force sequential execution.
  - Accepts `n_subjects` and `n_responders` as matrices (one trial row).
  - Uses only `a` and `b` from `j_data` and fills in `r,n`.
  - Returns a list of length 1 (one trial), each a matrix with rows `quantiles + Mean + SD` and columns `p_1, p_2`.
  - Quantiles, mean and SD agree with the theoretical Beta posterior Beta$(a + \sum r, b + \sum n - \sum r)$.
- **Stratified Beta backend with differences** – test "getPostQuantiles (stratified): multiple trials and calc_differences produce p_j and p_diff_*":
  - Uses matrices with two trials and two cohorts.
  - Priors given via `j_data`$a\_j, b\_j$.
  - `calc_differences requests p_diff_12.`
  - `Output is a list of length 2 (two trial outcome combinations), each matrix containing columns p_1, p_2, p_diff_12.`
  - `Row names equal the specified quantiles plus Mean, SD.`
  - `Ensures numeric and finite entries for p_1, p_2 and non-trivial values in p_diff_12.`

## 1.6 `loadAnalyses`

**Main responsibilities**

- Validate input arguments (`scenario_numbers`, `analysis_numbers`, `load_path`).
- Load previously saved analysis objects from disk.
- Reconstruct an `analysis_list` with consistent naming.

**Covered behaviours**

- **Happy path** – test "loadAnalyses: loads saved analyses and sets class/names correctly":
  - Two `.rds` files are created and then loaded.
  - Return object has class `analysis_list`.
  - Names are "scenario_1" and "scenario_2".
  - Contents match the originally saved objects.

- **Validation of** `scenario_numbers` – test `"loadAnalyses: scenario_numbers must be positive integers"`:
  - Non-numeric and non-positive inputs trigger errors matching `"scenario_numbers"`.
- **Validation of** `analysis_numbers` – test `"loadAnalyses: analysis_numbers must be positive integers of same length"`:
  - Length mismatch with `scenario_numbers` triggers an error.
  - Non-integerish or negative values trigger an error mentioning `"analysis_numbers"`.
- **Validation of** `load_path` – test `"loadAnalyses: load_path must be a single character string"`:
  - Non-character or vector-valued `load_path` trigger an error mentioning `"load_path"`.

### 1.7 `print.analysis_list`

**Main responsibilities**

- Provide a readable summary of an `analysis_list`.
- Print header summarising number of scenarios and methods.
- Print per-scenario blocks with key estimates per method.
- Honour the `digits` argument.

**Covered behaviours**

- **Basic structure and footer** – test `"print.analysis_list: prints header, scenario blocks, method label, and numeric estimates"`:
  - Header line: `"analysis_list of 2 scenarios with 1 method"`.
  - Scenario labels: `"  - scenario_1"`, `"  - scenario_2"`.
  - Method name printed with capitalised label (e.g. `"Pooled"`).
  - Numeric Mean and SD for one scenario appear in the output with expected rounding.
  - Footer contains lines mentioning MCMC iterations and available evidence levels.
- **Different scenarios produce different estimates** – test `"print.analysis_list: multiple scenarios print distinct scenario-specific estimates"`:
  - Two scenarios with different underlying response rates are analysed.
  - The printed output contains distinct Mean values for `p_1` for each scenario.

- `digits` **argument** – test `"print.analysis_list: digits argument controls printed numeric precision"`:
  - For `digits = 2`, numbers appear rounded to 2 decimal places.
  - For `digits = 4`, a more precise 4-decimal representation appears.

## 1.8 `saveAnalyses`

**Main responsibilities**

- Validate inputs (`analysis_list` object, `save_path`).
- Write each scenario analysis to disk in a consistent file naming scheme.
- Return metadata specifying which files were written.

**Covered behaviours**

- **Happy path** – test `"saveAnalyses: saves analysis_list to disk and loadAnalyses can read it back"`:
  - Valid `analysis_list` with two scenarios is saved to `tempdir()`.
  - Returned metadata includes matching `scenario_numbers`, `analysis_numbers` and `path`.
  - All expected `analysis_data_X_Y.rds` files exist on disk.
  - `loadAnalyses()` can reconstruct an equivalent `analysis_list`.
- **Class validation** – test `"saveAnalyses: non-analysis_list input triggers a class error"`:
  - Passing a plain `list` leads to an error mentioning `"analysis_list"`.
- `save_path` **validation** – test `"saveAnalyses: save_path must be a character vector of length 1"`:
  - A character vector of length greater than one triggers an error mentioning `"save_path"`.

## 1.9 `calcDiffsMCMC`

**Main responsibilities**

- Given a posterior sample matrix and a matrix of index pairs, compute differences $p_i - p_j$ for all requested pairs.
- Append the resulting columns to the original sample matrix with names `p_diff_ij`.

**Covered behaviours**

- Test `"calcDiffsMCMC: adds correctly named difference columns with correct values"`:
  - Original posterior columns `p_1, p_2, p_3, mu` remain present.
  - New columns `p_diff_12` and `p_diff_31` are added and named correctly.
  - Values match manual computation `posterior_samples[, p_i] - posterior_samples[, p_j]`.

## 1.10 `prepareAnalysis`

**Main responsibilities**

- Build (method-specific) JAGS data lists, parameter vectors, and model file paths.
- For `stratified` and `pooled`, pass through Beta prior parameters without using JAGS.
- Validate `method_name`.

**Covered behaviours**

- **Berry branch** – test `"prepareAnalysis: berry branch builds correct j_data and parameters"`:
  - `mean_mu`, `precision_mu`, `precision_tau`, and `p_t` in `j_data` match the prior parameters and target rates.
  - `J` equals #cohorts.
  - `j_parameters = c("p", "mu", "tau")` and model file exists.
- **EXNEX branch** – test `"prepareAnalysis: exnex branch builds mixture priors and pMix"`:
  - `Nexch`, `Nmix`, `Nstrata` computed correctly.
  - `mu_mean`, `mu_prec`, `tau_HN_scale`, `nex_mean`, `nex_prec` are correctly derived from priors.
  - `pMix` has length `Nexch + 1` and sums to 1.
  - `j_parameters = c("p", "mu", "tau", "exch")` and model file exists.
- **EXNEX_ADJ branch** – test `"prepareAnalysis: exnex_adj branch includes p_target and uses exnex_adj model"`:

  - `j_data`$p\_targetequals$`target_rates`.
  - `j_parameters identical to EXNEX branch.`
  - `Model file path contains "exnex_adj" and exists.`
- **Stratified and pooled** – test `"prepareAnalysis: stratified and pooled use dummy JAGS info and pass priors through"`:
  - Both return a dummy JAGS model path and dummy parameter string.
  - `j_data` for each method is identical to the corresponding prior parameters from `getPriorParameters()`.
- **Invalid method** – test `"prepareAnalysis: invalid method_name throws an error"`:
  - Unrecognised `method_name` raises an error mentioning acceptable method names.

### 1.11 `getUniqueRows`

**Main responsibilities**

- Given a numeric matrix, return its unique rows with the same number of columns.

**Covered behaviours**

- Test `"getUniqueRows: returns unique row combinations with correct columns"`:
  - Output has the same number of columns as input.
  - All rows in the output are unique.
  - A sorted comparison between `getUniqueRows(mat)` and `unique(mat)` shows identical values (ignoring internal names).

### 1.12 `getUniqueTrials`

**Main responsibilities**

- Collect all trial realisations from a `scenario_list`: responders, subjects, and go/no-go flags.
- Return the set of unique combinations across all scenarios.

**Covered behaviours**

- Test `"getUniqueTrials: combines scenarios and returns unique responder/subject/go rows"`:
  - Builds the same combined matrix internally as reconstructed in the test (responders, subjects, `go_flag`).
  - Output has 5 columns: 2 responders + 2 subjects + 1 go-flag.
  - All rows are unique.
  - The set of rows equals the set of `unique(combined)` (after sorting).

## 1.13 `mapUniqueTrials`

**Main responsibilities**

- Given unique trial configurations and method-specific quantiles, map these back to each scenario and each trial realisation.
- Optionally respect previous analyses and update only GO trials.

**Covered behaviours**

- **No previous trials** – test `"mapUniqueTrials: without previous trials, maps unique trial quantiles back per scenario"`:
  - `applicable_previous_trials = FALSE`.
  - One scenario with two trials, two cohorts.
  - Quantiles from `method_quantiles_list` are correctly mapped back: `scenario_1`*pooled*[[1]] == *q*1*and*`[[2]] == q2`.
- **With previous trials and GO flags** – test `"mapUniqueTrials: with previous trials, only GO trials are updated from hash tables"`:
  - `applicable_previous_trials = TRUE`.
  - `previous_analyses`*post_quantilesinitiallycontain*`"prev1"`*and*`"prev2"`.
  - `go_decisions vector marks first trial as GO, second as NoGo.`
  - `After mapping:`
    - `GO trial (row 1) quantiles are updated to new_q1.`
    - `NoGo trial (row 2) quantiles remain prev2.`

## 1.14 `posteriors2Quantiles`

**Main responsibilities**

- Convert posterior sample matrices into summary matrices: requested quantiles plus posterior mean and standard deviation, per column.

**Covered behaviours**

- Test `"posteriors2Quantiles: computes quantiles, mean, and sd for each column"`:
  - Posterior samples generated from a Normal distribution.
  - Output is a matrix with: rownames `"25%"`, `"50%"`, `"75%"`, `"Mean"`, `"SD"` and column `"theta"`.
  - Quantiles `25%, 50%, 75%` match `stats::quantile()` to Monte Carlo tolerance.
  - `Mean` and `SD` agree with sample mean and SD up to a small numerical tolerance.