

Package: DeCovarT (via r-universe)

October 1, 2026

Title Covariance-Aware Deconvolution of Bulk Transcriptomic Samples

Version 2.3.1

Description Estimates cell-type proportions in bulk transcriptomic samples with a probabilistic convolution model that integrates the gene-gene covariance structure of purified reference profiles. Cellular ratios are recovered by maximum likelihood under a multivariate Gaussian convolution, using analytic gradients and Hessians, an isometric log-ratio reparametrisation (Helmert basis) that enforces the simplex constraint, and Marquardt-Levenberg or Newton-type optimisers. The methodology is described in Chassagnol, Nuel and Becht (2023) <[doi:10.48550/arXiv.2309.09557](https://doi.org/10.48550/arXiv.2309.09557)>.

License MIT + file LICENSE

URL <https://github.com/bastienchassagnol/DeCovarT>,
<https://bastienchassagnol.github.io/DeCovarT/>

BugReports <https://github.com/bastienchassagnol/DeCovarT/issues>

Depends R (>= 4.1.0)

Imports dplyr, ggplot2 (>= 3.5.0), marqLevAlg, MASS, Matrix, purrr,
Rdpack (>= 0.7), rlang, tensor, tibble, tidyr

Suggests circlize, cli, ComplexHeatmap, compositions, cowplot, e1071,
EMMIXmfa, flextable, forcats, funkyheatmap, furrr, future,
ggdendro, ggdist, ggtext, glmnet, gridExtra, htmlwidgets,
igraph, knitr, limSolve, litedown, mclust, Metrics, MixSim,
mvtnorm, nnls, numDeriv, patchwork, pkgdown, plotly, png, ps,
qrrg, quarto, qqplotr, reactable (>= 0.4.4), readr, rgl,
rmarkdown, spelling, stringr, testthat (>= 3.0.0), tinytable
(>= 0.15.0), viridis, withr

VignetteBuilder knitr, quarto

RdMacros Rdpack

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE, roclets = c("`namespace", "`rd",
 "`srr::srr_stats_roclet"))

RoxygenNote 8.1.0

Config/pak/sysreqs libicu-dev

Repository <https://ropensci-staging.r-universe.dev>

Date/Publication 2026-10-01 15:38:11 UTC

RemoteUrl <https://github.com/bastienchassagnol/DeCovarT>

RemoteRef HEAD

RemoteSha 5245f2cf91276ed01a14d60aa4944ef0fcf58e54

Contents

additive_logistic	3
algorithm_similarity	4
bootstrap_decovart	5
check_true_theta	7
chi_bar_square_pvalue	8
composition_from_entropy	10
compute_average_jeffreys	10
compute_average_overlap	11
compute_average_riemannian	12
compute_benchmark_metrics	13
compute_glmnet_gene_scores	15
compute_shannon_entropy	17
confint_profile_decovart	18
coverage_mc_interval	20
deconvolute_ratios	20
deconvolute_ratios_cibersort	23
deconvolute_ratios_gls	26
describe_simulation_scenario	27
equicorrelation_gram	28
expected_fisher_unconstrained	29
fit_decovart	30
fixed_gls_covariance	33
generate_mean_signature_matrix	33
helmert_basis	35
hessian_isometric_logistic	36
isometric_logistic	36
jacobian_isometric_logistic	37
lrt_decovart	38
multistart_decovart	40
overlap_gaussian_mc	41
pivot_mc_estimates	42

plot_algorithm_similarity	43
plot_correlation_Heatmap	44
plot_mc_forest	46
plot_mc_metric_dots	47
plot_mc_raincloud	49
reference_bootstrap_decovart	51
repair_simplex	54
run_simulation_benchmark	55
scale_covariance_array	57
simulate_bulk_mixture	58
simulate_hierarchical_grn_moments	59
spd_affine_invariant_distance	61
starting_simplex	62
theme_decovart_facets	63
vcov_ilmr_delta	64
Index	66

additive_logistic	<i>Additive logistic transform (unconstrained coordinates to the simplex)</i>
-------------------	---

Description

Inverse additive log-ratio map $\psi : \boldsymbol{\rho} \mapsto \boldsymbol{p}$ of Aitchison (alr^{-1}): a softmax with the last category J pinned as reference ($\rho_J \equiv 0$). Solvers and `vcov.decovart_fit()` use the isometric log-ratio chart `isometric_logistic()` instead. This ALR helper is retained for the vignette appendix.

Recovers the unconstrained additive log-ratio coordinates $\rho_j = \ln(p_j/p_J)$ for $j = 1, \dots, J-1$, with the last part p_J as reference. This is Aitchison's additive log-ratio (alr) transform, equivalently the multinomial-logit link with reference category J (see `compositions::alr()` and `vignette("theory-decovart-generation", package = "DeCovart")`).

Usage

```
additive_logistic(rho)

additive_log_ratio(p)
```

Arguments

rho	Numeric vector $\boldsymbol{\rho} \in \mathbb{R}^{J-1}$ of unconstrained additive log-ratio coordinates (reference cell type J).
p	Numeric vector $\boldsymbol{p}$ on the open simplex.

Details

With $A = \sum_{k=1}^{J-1} e^{\rho_k} + 1$,

$$p_j = \frac{e^{\rho_j}}{A} \quad (j < J), \quad p_J = \frac{1}{A}.$$

Equivalently, $\mathbf{p} = \boldsymbol{\psi}(\boldsymbol{\rho})$ with $\boldsymbol{\psi}(\boldsymbol{\rho}) \propto (e^{\rho_1}, \dots, e^{\rho_{J-1}}, 1)^\top$. Jacobians and Hessians of both maps are derived in the package vignette `vignette("theory-decovart-generative-model", package = "DeCovart")`. See also `compositions::alrInv()`.

Value

Numeric vector $\mathbf{p} \in \mathbb{R}^J$ on the unit simplex ($\mathbf{1}^\top \mathbf{p} = 1$, $\mathbf{p} \geq \mathbf{0}$).

Numeric vector $\boldsymbol{\rho} \in \mathbb{R}^{J-1}$.

Numerical stability

Evaluated through the log-sum-exp shift $p_i \propto \exp(\tilde{\rho}_i - \max_k \tilde{\rho}_k)$ with $\tilde{\boldsymbol{\rho}} = (\rho_1, \dots, \rho_{J-1}, 0)$, which is algebraically identical to the ratio above but never forms e^{ρ_i} directly. The naive quotient overflows to NaN for $\rho_i \gtrsim 710$, which unconstrained ascent can reach when the MLE approaches a simplex face; the shifted form returns an exact zero instead and keeps $\boldsymbol{\Sigma}(\mathbf{p})$ positive definite.

See Also

The inverse map (additive log-ratio) is documented as `additive_log_ratio()` on this help page.

Examples

```
rho <- c(0.2, -0.5)
p <- additive_logistic(rho)
sum(p)
additive_log_ratio(p)
# Stable far out in ALR space, where exp(rho) would overflow.
additive_logistic(c(800, 1200))
```

algorithm_similarity *Algorithm-similarity correlation from a Monte Carlo benchmark*

Description

Pearson correlations $r_{ab} = \text{cor}(\hat{p}_a, \hat{p}_b)$ across Monte Carlo replicates (and cell types). This is **behavioural similarity**: two solvers can correlate near 1 while remaining systematically biased. Hierarchical order uses Ward D2 on $d_{ab} = 1 - r_{ab}$.

Usage

```
algorithm_similarity(benchmark, facet_rows = NULL, facet_cols = NULL)
```

Arguments

`benchmark` List from `run_simulation_benchmark()`.
`facet_rows, facet_cols` Optional scenario columns. When supplied, correlations are computed inside each facet cell.

Value

A tibble with `algorithm_x`, `algorithm_y`, `correlation`, and any facet columns.

See Also

`plot_algorithm_similarity()`, `pivot_mc_estimates()`

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
out <- run_simulation_benchmark(
  tibble::tibble(
    true_theta = list(list(p = c(0.5, 0.5), mu = mu, sigma = Sigma))
  ),
  deconvolution_functions = list(
    "nnls" = list(FUN = deconvolute_ratios_nnls),
    "rlm" = list(FUN = deconvolute_ratios_rlm)
  ),
  n = 4,
  cores = 1
)
algorithm_similarity(out)
```

bootstrap_decovart *Parametric bootstrap for DeCovarT proportions and boundary tests*

Description

The conditional law of the bulk profile is fully specified, so the asymptotic calibrations of `lrt_decovart()` can be replaced by simulation. Given plug-in moments and a composition $\mathbf{p}$, the function draws $\mathbf{Y}^{(b)} \sim \mathcal{N}_G(\boldsymbol{\mu}\mathbf{p}, \boldsymbol{\Sigma}(\mathbf{p}))$, refits DeCovarT, and returns the empirical distribution of $\hat{\mathbf{p}}^{(b)}$ together with percentile intervals.

Usage

```
bootstrap_decovart(
  y = NULL,
  mean_signature_matrix,
  Sigma,
  bulk_expression = NULL,
  p = NULL,
  null_value = NULL,
  n_boot = 500L,
  level = 0.95,
  n_replicates = NULL,
  epsilon = 10^-8,
  itmax = 500L
)
```

Arguments

y	Numeric vector (or one-column matrix) $\mathbf{y} \in \mathbb{R}^G$.
mean_signature_matrix	Numeric matrix $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ (plug-in means).
Sigma	Array of cell-type covariances in $\mathcal{M}_{G \times G \times J}$.
bulk_expression	Numeric matrix $\mathbf{Y} \in \mathcal{M}_{G \times N}$ of replicate bulk profiles sharing one composition, used instead of y for the pooled (population) test.
p	Composition to simulate from. Defaults to the MLE computed from y / bulk_expression.
null_value	Named numeric vector of hypothesised ratios, e.g. c(celltype_2 = 0).
n_boot	Number of bootstrap replicates.
level	Confidence level for percentile intervals.
n_replicates	Number of bulk columns simulated per replicate; defaults to the number of observed columns.
epsilon, itmax	Relative convergence tolerance and iteration budget passed to <code>stats::optim()</code> .

Details

Supplying null_value switches to a restricted bootstrap: replicates are generated under the restricted MLE and the likelihood-ratio statistic is recomputed on each, giving a Monte Carlo p-value that does not rely on the 50:50 asymptotic weights. For a composite null the remaining nuisance proportions are profiled at their restricted estimate, so the procedure is a plug-in (approximate) rather than an exact bootstrap.

Reference uncertainty is **not** propagated here: $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}_j$ are treated as known, exactly as in the frequentist likelihood. To resample the purified observations that produced those plug-in moments, use `reference_bootstrap_decovart()`.

Value

A list with estimates ($J \times n_boot$ matrix), interval (percentile bounds), p_simulated, and, when null_value is supplied, statistic (observed D), null_statistics, and p_value.

See Also

[lrt_decovart\(\)](#), [chi_bar_square_pvalue\(\)](#), [reference_bootstrap_decovart\(\)](#)

Other decovart_inference: [boundary_diagnostics\(\)](#), [chi_bar_square_pvalue\(\)](#), [confint_profile_decovart\(\)](#), [equivariance_check_decovart\(\)](#), [lrt_decovart\(\)](#), [multistart_decovart\(\)](#), [profile_loglik_decovart\(\)](#), [reference_bootstrap_decovart\(\)](#), [restricted_mle_decovart\(\)](#)

Examples

```
mu <- matrix(c(20, 40, 15, 40, 20, 25), nrow = 3)
colnames(mu) <- paste0("ct", 1:2)
Sigma <- array(c(diag(3), diag(3)), dim = c(3, 3, 2))
y <- drop(mu %*% c(0.6, 0.4))
set.seed(1)
bootstrap_decovart(y, mu, Sigma, n_boot = 20)$interval
```

check_true_theta	<i>Validate generative-model parameters θ</i>
------------------	---

Description

Checks that true_theta stores the DeCovarT mixture parameters:

- p: mixing proportions of length J , or a matrix $J \times N$ of sample-wise proportions (each column on the simplex);
- mu: mean signature $\mu \in \mathcal{M}_{G \times J}$ (also accepts $J \times G$);
- sigma and/or Theta: array $G \times G \times J$ of covariances and/or precision matrices.

Usage

```
check_true_theta(
  true_theta,
  require_p = TRUE,
  J = NULL,
  second_moment = c("either", "sigma", "Theta")
)
```

Arguments

true_theta	Named list with the fields above.
require_p	Logical; if TRUE (default), p must be present and valid. If FALSE, a missing p is allowed (callers may default it).
J	Optional expected number of cell types. When NULL, inferred from mu / the third dimension of sigma or Theta.
second_moment	Which second-moment field to require: "sigma", "Theta", or "either" (default: at least one of sigma / Theta). Matching is case-insensitive.

Value

TRUE invisibly if the structure is valid; otherwise stops with an informative error.

See Also

`compute_average_overlap()`, `compute_average_jeffreys()`

Examples

```
theta <- list(
  p = c(0.5, 0.5),
  mu = cbind(c(0, 0), c(3, 0)),
  sigma = array(c(diag(2), diag(2)), dim = c(2, 2, 2))
)
check_true_theta(theta)
```

chi_bar_square_pvalue *Chi-bar-square tail probability for boundary likelihood-ratio tests*

Description

Upper-tail probability of the mixture

$$\bar{\chi}^2 = \sum_{i=0}^q w_i \chi_{s+i}^2, \quad w_i = \binom{q}{i} 2^{-q},$$

with q constraints active on the boundary of the parameter space and s interior constraints. This is Case 9 of (Self and Liang 1987), which generalises the $\frac{1}{2}\chi_0^2 + \frac{1}{2}\chi_1^2$ result of (Chernoff 1954). For $q = 0$ the mixture collapses to the ordinary χ_s^2 of (Wilks 1938).

Usage

```
chi_bar_square_pvalue(
  statistic,
  n_boundary = 1L,
  df_interior = 0L,
  weights = NULL
)
```

Arguments

statistic	Numeric vector of observed likelihood-ratio statistics $D = 2(\hat{\ell}_1 - \hat{\ell}_0)$.
n_boundary	Integer q : number of constraints active on the boundary (e.g. null ratios set to zero).
df_interior	Integer s : number of interior (two-sided) constraints tested simultaneously.
weights	Optional numeric vector of length n_boundary + 1 giving the mixing weights $w_0, \dots, w_q$. Defaults to the binomial weights.

Details

The binomial weights hold when the block of the Fisher information corresponding to the active constraints is diagonal after orthogonalisation against the nuisance block. Under correlated constraints the mixing weights depend on the geometry of the approximating tangent cone and the binomial choice is only an approximation; Self and Liang also exhibit a nuisance-parameter configuration whose limit is **not** a chi-square mixture. Supply weights explicitly, or calibrate by simulation with `bootstrap_decovart()`, whenever $q > 1$ and the constrained cell types share signal.

Value

Numeric vector of upper-tail probabilities.

Atom at zero

χ_0^2 is a point mass at the origin, so its strict upper tail $\Pr(\chi_0^2 > D)$ is zero for every $D \geq 0$. The component therefore only removes weight: with one active constraint and no interior constraint the p-value is $\frac{1}{2} \Pr(\chi_1^2 > D)$, capped at 0.5 when $D = 0$. Note `stats::pchisq()` switches to the *closed* tail at exactly $q = 0$ for $df = 0$; the strict convention is used here so the p-value stays continuous in statistic.

References

- Self SG, Liang K (1987). “Asymptotic Properties of Maximum Likelihood Estimators and Likelihood Ratio Tests under Nonstandard Conditions.” *Journal of the American Statistical Association*, **82**(398), 605–610. ISSN 0162-1459. doi:10.1080/01621459.1987.10478472.
- Chernoff H (1954). “On the Distribution of the Likelihood Ratio.” *The Annals of Mathematical Statistics*, **25**(3), 573–578. ISSN 0003-4851. doi:10.1214/aoms/1177728725.
- Wilks SS (1938). “The Large-Sample Distribution of the Likelihood Ratio for Testing Composite Hypotheses.” *The Annals of Mathematical Statistics*, **9**(1), 60–62. ISSN 0003-4851. doi:10.1214/aoms/1177732360.

See Also

`lrt_decovart()`

Other `decovart_inference`: `bootstrap_decovart()`, `boundary_diagnostics()`, `confint_profile_decovart()`, `equivariance_check_decovart()`, `lrt_decovart()`, `multistart_decovart()`, `profile_loglik_decovart()`, `reference_bootstrap_decovart()`, `restricted_mle_decovart()`

Examples

```
# One active boundary constraint: 50:50 mixture of chi2_0 and chi2_1.
chi_bar_square_pvalue(2.71, n_boundary = 1L)
# Interior (Wilks) calibration.
chi_bar_square_pvalue(3.84, n_boundary = 0L, df_interior = 1L)
```

composition_from_entropy

One-dominant composition with a target normalised Shannon entropy

Description

Returns a length- J simplex vector of the form $(1 - (J - 1)q, q, \dots, q)$ whose normalised Shannon entropy `compute_shannon_entropy()` equals `h_star`. The uniform composition (`h_star = 1`) and a Dirac mass (`h_star = 0`) are returned exactly.

Usage

```
composition_from_entropy(h_star, n_celltypes = 3L, nms = NULL)
```

Arguments

<code>h_star</code>	Target $H^* \in [0, 1]$.
<code>n_celltypes</code>	Integer $J \geq 2$.
<code>nms</code>	Optional names for the returned vector.

Value

Numeric simplex vector of length J .

See Also

`compute_shannon_entropy()`

Examples

```
p <- composition_from_entropy(0.5, n_celltypes = 3L)
compute_shannon_entropy(p)
```

compute_average_jeffreys

Average pairwise Jeffreys divergence of a Gaussian mixture

Description

Returns the $p_j p_\ell$ -weighted mean of pairwise Jeffreys (symmetrised KL) divergences between purified Gaussians

$$\bar{J} = \frac{\sum_{1 \leq j < \ell \leq J} p_j p_\ell J(f_j, f_\ell)}{\sum_{1 \leq j < \ell \leq J} p_j p_\ell} \in [0, \infty),$$

with $f_j = \mathcal{N}_G(\boldsymbol{\mu}_{\cdot j}, \boldsymbol{\Sigma}_j)$. If `p` is omitted it defaults to the equi-balanced vector $1/J$, which recovers the uniform pairwise average $2/(J(J - 1)) \sum_{j < \ell} J(f_j, f_\ell)$.

Usage

```
compute_average_jeffreys(true_theta, J = NULL)
```

Arguments

`true_theta` List validated by `check_true_theta()`: μ ($G \times J$), σ ($G \times G \times J$), and optionally p (length J or $J \times N$). If p is missing it is set to $(1/J, \dots, 1/J)$.

`J` Number of cell types. Defaults to the third dimension of σ .

Value

Scalar average pairwise Jeffreys divergence.

See Also

`check_true_theta()`, `compute_average_overlap()`, `compute_glmnet_gene_scores()`

Examples

```
set.seed(1)
theta <- list(
  mu = cbind(c(0, 0), c(3, 0)),
  sigma = array(c(diag(2), diag(2)), dim = c(2, 2, 2))
)
compute_average_jeffreys(theta)
```

```
compute_average_overlap
```

Average pairwise overlap of a Gaussian mixture

Description

Returns MixSim's **BarOmega**: the unweighted mean of pairwise overlaps

$$\bar{\omega} = \frac{2}{J(J-1)} \sum_{1 \leq j < \ell \leq J} (\Omega_{j\ell} + \Omega_{\ell j}) \in [0, 1]$$

(up to the MixSim numerical convention), where $\Omega_{j\ell} = \Pr_{X \sim f_j}(X \text{ classified as } \ell)$ already uses the mixture weights p inside the Bayes / MAP rule of `MixSim::overlap()`. Do **not** multiply the directional masses by p_j again.

For $G \geq 4$ MixSim's Davies quadrature becomes expensive. The helper then switches to `overlap_gaussian_mc()`: stratified Sobol draws, inverse-transform sampling through precomputed Cholesky factors, and log-density MAP comparisons (`n_mc` draws per component; default 10,000).

Usage

```
compute_average_overlap(
  true_theta,
  J = NULL,
  n_mc = 10000L,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

true_theta	List validated by check_true_theta() : p (length J or $J \times N$), μ ($G \times J$), σ ($G \times G \times J$).
J	Number of cell types (components). Defaults to the third dimension of σ .
n_mc	Monte Carlo draws per component when $G \geq 4$.
seed	Optional seed forwarded to overlap_gaussian_mc() .
verbose	If TRUE (default), announce the MixSim-to-MC switch with cli when it is installed.

Value

Scalar average pairwise overlap (MixSim BarOmega).

See Also

[check_true_theta\(\)](#), [overlap_gaussian_mc\(\)](#)

Examples

```
set.seed(1)
theta <- list(
  p = c(0.5, 0.5),
  mu = cbind(c(0, 0), c(3, 0)),
  sigma = array(c(diag(2), diag(2)), dim = c(2, 2, 2))
)
compute_average_overlap(theta)
```

compute_average_riemannian

Mean pairwise AIRM distance of component covariances

Description

Mean pairwise AIRM distance of component covariances

Usage

```
compute_average_riemannian(true_theta, J = NULL)
```

Arguments

true_theta List validated by [check_true_theta\(\)](#): μ ($G \times J$), σ ($G \times G \times J$), and optionally p (length J or $J \times N$). If p is missing it is set to $(1/J, \dots, 1/J)$.

J Number of cell types. Defaults to the third dimension of σ .

Value

Scalar average of $d_R(\Sigma_j, \Sigma_\ell)$ over $j < \ell$.

See Also

[spd_affine_invariant_distance\(\)](#)

Examples

```
a <- diag(2)
b <- matrix(c(2, 0.5, 0.5, 1), nrow = 2)
theta <- list(
  mu = cbind(c(0, 0), c(1, 0)),
  sigma = array(c(a, b), dim = c(2, 2, 2))
)
compute_average_riemannian(theta)
# J = 2 has a single pair, so the average equals the pairwise AIRM.
all.equal(
  compute_average_riemannian(theta),
  spd_affine_invariant_distance(a, b)
)
```

```
compute_benchmark_metrics
```

Compute deconvolution benchmark metrics

Description

Returns three blocks: composition / regression scores, Monte Carlo summaries (ADEMP-style), and optimisation / runtime diagnostics. When p^* is a matrix $J \times N$ (or $\hat{p}$ is), cell-type Pearson correlation and Monte Carlo summaries are computed **across samples**, not within a single composition.

Usage

```
compute_benchmark_metrics(
  y,
  mean_signature_matrix,
  estimated_p,
  true_ratios = NULL,
  se = NULL,
  lower = NULL,
  upper = NULL,
  elapsed = NULL,
  memory_bytes = NULL,
  kkt_residual = NULL,
  numerical_converged = NULL,
  loglik_hat = NULL,
  loglik_true = NULL,
  converged = NULL,
  local_maximum = NULL,
  min_eigenvalue = NULL,
  max_eigenvalue = NULL,
  presence_threshold = 1e-04,
  level = 0.95,
  coverage_interval = "wilson",
  algorithm = NA_character_
)
```

Arguments

<code>y</code>	Numeric vector (or one-column matrix) $\mathbf{y} \in \mathbb{R}^G$.
<code>mean_signature_matrix</code>	Numeric matrix $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ (plug-in means).
<code>estimated_p</code>	Estimated proportions: a length- J vector or a numeric matrix $J \times N$.
<code>true_ratios</code>	Optional ground truth with the same layout. When omitted, reconstitution scores compare $\hat{\mathbf{y}} = \boldsymbol{\mu}\hat{\mathbf{p}}$ with $\mathbf{y}$.
<code>se</code>	Optional standard errors matching <code>estimated_p</code> . Convolution-likelihood solvers fill these from <code>vcov_ilr_delta()</code> (expected Fisher information pulled back through the ILR chart). Mean-only solvers (NNLS, LSEI) leave them missing.
<code>lower, upper</code>	Optional interval bounds matching <code>estimated_p</code> (used for coverage and mean width). When omitted but <code>se</code> is supplied, Wald intervals at <code>level</code> are used.
<code>elapsed, memory_bytes, kkt_residual</code>	Optional per-sample runtime diagnostics (recycled to N).
<code>numerical_converged</code>	Optional logical per sample: the solver returned a finite simplex vector.
<code>loglik_hat, loglik_true</code>	Optional per-sample log-likelihoods used for theoretical convergence (regret $\ell(\mathbf{p}^*) - \ell(\hat{\mathbf{p}})$).

converged	Optional list of length N with homogenised optimiser fields (iterations, criterion, code, rdm). Closed-form solvers store NA entries.
local_maximum	Optional logical per sample: ILR score small and Hessian negative definite ($\lambda_{\min} < 0$ and $\lambda_{\max} < 0$).
min_eigenvalue, max_eigenvalue	Optional ILR Hessian extrema.
presence_threshold	Threshold ε for presence / F1 / false-positive mass (default $1e-4$).
level	Wald coverage level when lower / upper are omitted (default 0.95).
coverage_interval	Interval for the Monte Carlo coverage rate $\hat{\pi}$: "wilson" (default), "wald", or "agresti_coull". This is not the interval for p_j itself.
algorithm	Character label stored on each metrics row (typically the solver name). Recycled to length one.

Value

A named list with:

- regression: global (one row per sample) and cell_type (one row per cell type);
- monte_carlo: one row per cell type, including theoretical_se (filled later from expected Fisher at p^*);
- optimisation: one row per sample.

Examples

```
mu <- matrix(c(20, 22, 22, 20), nrow = 2,
             dimnames = list(paste0("g", 1:2), paste0("ct", 1:2)))
y <- drop(mu %*% c(0.4, 0.6))
compute_benchmark_metrics(y, mu, estimated_p = c(0.45, 0.55),
                        true_ratios = c(0.4, 0.6))
```

compute_glmnet_gene_scores

Gene scores from multinomial elastic-net cell-type classification

Description

Fits a multinomial (or binomial) elastic net (`glmnet::glmnet()`) that predicts cell type from expression features. Inputs are purified expression profiles $\mathbf{X} \in \mathcal{M}_{G \times J \times N}$ (genes $\times$ cell types $\times$ samples) and length- J cell-type labels. Variability across samples replaces synthetic isotropic noise. Gene scores are the sum over classes of absolute coefficients at a chosen λ (intercept excluded). For nested / CV selection of λ , see the experimental `compute_glmnet_gene_scores_cv()` helper (not shipped in the package build).

Usage

```
compute_glmnet_gene_scores(
  expression_profiles,
  celltype_labels,
  alpha = 0.5,
  lambda = NULL,
  ...
)
```

Arguments

expression_profiles	Numeric array $G \times J \times N$ of purified profiles.
celltype_labels	Character or factor labels of length J (one per cell-type slice).
alpha	Elastic-net mixing parameter in $[0, 1]$ (default 0.5).
lambda	Optional penalty value at which coefficients are extracted. When NULL, uses the smallest λ on the fitted path.
...	Additional arguments forwarded to <code>glmnet::glmnet()</code> .

Value

Named numeric vector of length G (gene scores; larger means stronger multinomial signal).

See Also

`compute_average_jeffreys()`, `compute_average_overlap()`

Examples

```
## Not run:
set.seed(1)
G <- 4L
J <- 3L
N <- 12L
profiles <- array(0, dim = c(G, J, N))
for (j in seq_len(J)) {
  profiles[j, j, ] <- 5 + stats::rnorm(N, sd = 0.2)
}
labels <- paste0("ct", seq_len(J))
scores <- compute_glmnet_gene_scores(profiles, labels)
names(scores)[which.max(scores)]

## End(Not run)
```

compute_shannon_entropy

Normalised Shannon entropy of a discrete distribution

Description

For a probability vector $\mathbf{p} \in \Delta^{J-1}$ over J classes (here: cell types), the Shannon entropy is

$$H(\mathbf{p}) = - \sum_{j=1}^J p_j \log p_j,$$

with the convention $0 \log 0 = 0$. Dividing by the maximum entropy $\log J$ (uniform over all J classes) yields normalised Shannon entropy

$$H^*(\mathbf{p}) = \frac{H(\mathbf{p})}{\log J} \in [0, 1],$$

so $H^* = 0$ for a Dirac mass on one type and $H^* = 1$ for the uniform distribution over the J cell types. Zero masses are dropped only inside the sum; the normaliser still uses the original class count J .

This is preferable to changing the logarithm base to the number of *positive* masses J' , which would renormalise only on the support and hide sparsity relative to the full panel of cell types.

Usage

```
compute_shannon_entropy(ratios)
```

Arguments

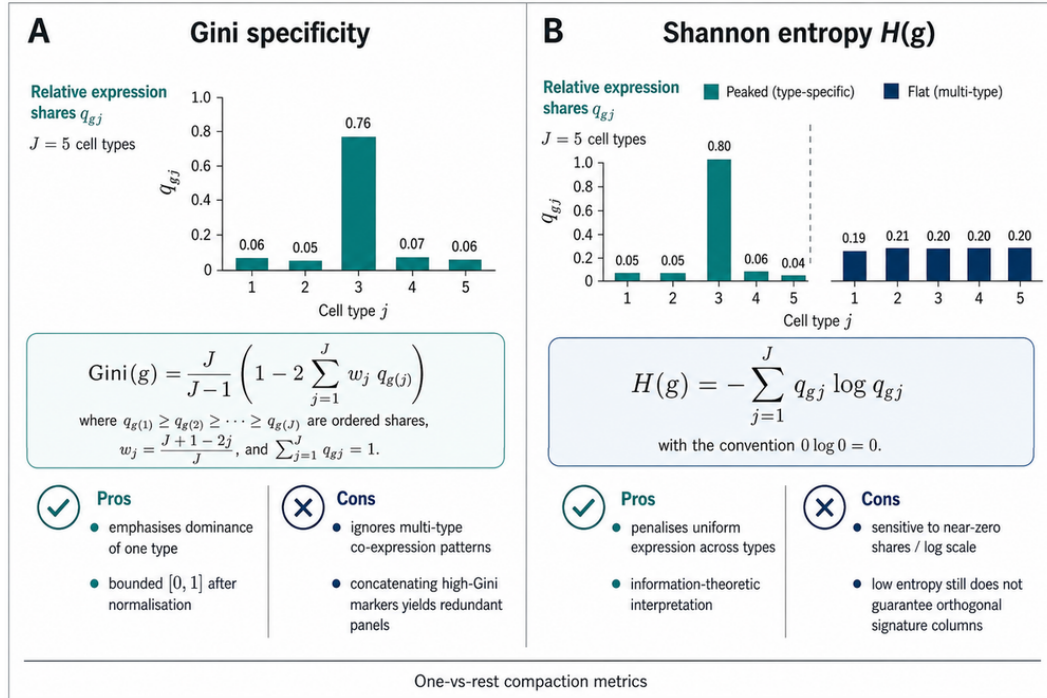
ratios Numeric vector of cellular proportions $\mathbf{p} \in \Delta^{J-1}$ (length J).

Value

Scalar normalised entropy $H^* \in [0, 1]$.

Illustration

Panel B of the figure contrasts a peaked (type-specific, low entropy) share vector with a flat (multi-type, high entropy) one for $J = 5$ classes.



See Also

[composition_from_entropy\(\)](#)

Examples

```
compute_shannon_entropy(c(1, 0, 0)) # Dirac -> 0
compute_shannon_entropy(rep(1 / 3, 3)) # uniform -> 1
```

confint_profile_decovart

Profile-likelihood confidence intervals for cellular ratios

Description

Inverts the likelihood-ratio test of [lrt_decovart\(\)](#) coordinate by coordinate: the interval collects every c with $2\{\ell(\hat{p}) - \ell_j^{\text{prof}}(c)\} \leq \chi_{1,1-\alpha}^2$. Because likelihood ratios are invariant under reparametrisation, the interval respects $0 \leq p_j \leq 1$ without the delta-method linearisation that makes Wald intervals unreliable near a simplex face.

Usage

```
confint_profile_decovart(
  y = NULL,
```

```

    mean_signature_matrix,
    Sigma,
    bulk_expression = NULL,
    level = 0.95,
    celltypes = NULL,
    epsilon = 10^-8,
    itmax = 500L
  )

```

Arguments

y Numeric vector (or one-column matrix) $\mathbf{y} \in \mathbb{R}^G$.

mean_signature_matrix Numeric matrix $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ (plug-in means).

Sigma Array of cell-type covariances in $\mathcal{M}_{G \times G \times J}$.

bulk_expression Numeric matrix $\mathbf{Y} \in \mathcal{M}_{G \times N}$ of replicate bulk profiles sharing one composition, used instead of $\mathbf{y}$ for the pooled (population) test.

level Confidence level $1 - \alpha$.

celltypes Cell types to profile; defaults to all.

epsilon, itmax Relative convergence tolerance and iteration budget passed to `stats::optim()`.

Value

A matrix with one row per cell type and columns estimate, lower, upper.

See Also

`confint.decovart_fit()` for the Wald analogue, `lrt_decovart()`

Other `decovart_inference`: `bootstrap_decovart()`, `boundary_diagnostics()`, `chi_bar_square_pvalue()`, `equivariance_check_decovart()`, `lrt_decovart()`, `multistart_decovart()`, `profile_loglik_decovart()`, `reference_bootstrap_decovart()`, `restricted_mle_decovart()`

Examples

```

mu <- matrix(c(20, 40, 15, 40, 20, 25), nrow = 3)
colnames(mu) <- paste0("ct", 1:2)
Sigma <- array(c(diag(3), diag(3)), dim = c(3, 3, 2))
y <- drop(mu %*% c(0.6, 0.4))
confint_profile_decovart(y, mu, Sigma)

```

coverage_mc_interval *Monte Carlo interval for an empirical coverage probability*

Description

Interval for a binomial coverage rate $\hat{\pi} = X/N$ from independent coverage indicators. Wilson is the default because the parameter is bounded in $[0, 1]$ and Wald intervals behave poorly near 0 or 1.

Usage

```
coverage_mc_interval(
  covered,
  conf_level = 0.95,
  method = c("wilson", "wald", "agresti_coull")
)
```

Arguments

covered	Logical or 0/1 coverage indicators (one per replicate).
conf_level	Confidence level for the interval around $\hat{\pi}$ (default 0.95).
method	"wilson" (default), "wald", or "agresti_coull".

Value

A list with n, successes, coverage, mcse, lower, upper, and method.

Examples

```
coverage_mc_interval(c(TRUE, TRUE, TRUE, FALSE))
```

deconvolute_ratios *Parallel deconvolution of a bulk expression matrix*

Description

For each column $\mathbf{y}_{\cdot i}$ of the bulk matrix $\mathbf{Y} \in \mathcal{M}_{G \times N}$, estimates $\hat{\mathbf{p}}_{\cdot i}$ with every supplied algorithm. Samples are iterated with `furrr` when `cores > 1`; algorithms are sequential, so workers are never nested. When covariance information is provided, DeCovarT methods maximise $\ell_{\mathbf{y}|\boldsymbol{\zeta}}(\mathbf{p})$ under $\mathbf{y} | (\boldsymbol{\zeta}, \mathbf{p}) \sim \mathcal{N}_G(\boldsymbol{\mu}\mathbf{p}, \boldsymbol{\Sigma}(\mathbf{p}))$.

Usage

```
deconvolute_ratios(
  signature_matrix,
  bulk_expression,
  true_ratios = NULL,
  Sigma = NULL,
  deconvolution_functions = NULL,
  standardise = FALSE,
  scaled = FALSE,
  cores = getOption("mc.cores", 1L),
  coverage_interval = "wilson",
  verbose = FALSE,
  progress_every = 10L
)
```

Arguments

signature_matrix	Mean signature $\mu \in \mathcal{M}_{G \times J}$ (rownames = genes, colnames = cell types).
bulk_expression	Bulk matrix $Y \in \mathcal{M}_{G \times N}$ (rownames = genes, colnames = samples).
true_ratios	Optional ground-truth proportions: a length- J numeric vector (recycled over samples) or a numeric matrix $J \times N$ (or $N \times J$).
Sigma	Optional array $(\Sigma_j)_j \in \mathcal{M}_{G \times G \times J}$ of cell-type covariances (numeric; off-diagonals may be negative).
deconvolution_functions	Named list; each element has FUN and optional additional_parameters.
standardise	If TRUE, apply a gene-wise affine z-score computed once from μ to bulk, means and covariances (see Details). Cell-type-wise or sample-wise transforms are not supported.
scaled	Deprecated. TRUE (log2 mixing) always errors.
cores	Number of furrr workers for the sample loop. Defaults to getOption("mc.cores", 1L). Use cores = 1 to stay sequential (CRAN examples and nested callers). With cores > 1, .map_samples() uses furrr_options(seed = TRUE), which assigns an independent L'Ecuyer-CMRG stream per worker (R 4.1 / future streams). Do not offset a shared seed by the sample index.
coverage_interval	Passed to compute_benchmark_metrics() ("wilson", "wald", or "agresti_coull").
verbose	If TRUE, emit progress for sequential sample loops (every progress_every samples). Parallel furrr runs log a one-line notice instead of per-sample ticks.
progress_every	Sample-progress interval when verbose is TRUE and cores = 1. Defaults to 10L.

Value

A named list from [compute_benchmark_metrics\(\)](#) with regression (global and cell-type sub-tables), monte_carlo, and optimisation (per-sample elapsed time, memory, KKT residual, and

$\hat{p}$). First-generation solvers still call `repair_simplex()`; the three native DeCovarT maps (ILR or $p/\sum p$) already lie on the simplex. When Sigma is supplied, Newton-Raphson, Marquardt-Levenberg, L-BFGS-B, BFGS (gradient), and simulated annealing attach ILR Wald standard errors from `vcov_ilr_delta()` so `monte_carlo` can report coverage, mean model SE, and the SE/SD ratio. Those SEs are the expected-Fisher delta-method bounds used by `confint.decovart_fit()`, not diagonal entries of the observed Hessian. Mean-only solvers (NNLS, LSEI) leave Wald columns missing: those estimators do not use the convolution likelihood.

References

Erkkilä T, Lehmusvaara S, Ruusuvaara P, Visakorpi T, Shmulevich I, Lähdesmäki H (2010). “Probabilistic Analysis of Gene Expression Measurements from Heterogeneous Tissues.” *Bioinformatics*, **26**. doi:10.1093/bioinformatics/btq406. Ahn J, Yuan Y, Parmigiani G, Suraokar MB, Diao L, Wistuba II, Wang W (2013). “DeMix: Deconvolution for Mixed Cancer Transcriptomes Using Raw Measured Data.” *Bioinformatics (Oxford, England)*, **29**. doi:10.1093/bioinformatics/btt301. Wang Z, Cao S, Morris JS, Ahn J, Liu R, Tyekucheva S, Gao F, Li B, Lu W, Tang X, Wistuba II, Bowden M, Mucci L, Loda M, Parmigiani G, Holmes CC, Wang W (2018). “Transcriptome Deconvolution of Heterogeneous Tumor Samples with Immune Infiltration.” *iScience*, **9**. doi:10.1016/j.isci.2018.10.028. Anghel CV, Quon G, Haider S, Nguyen F, Deshwar AG, Morris QD, Boutros PC (2015). “ISOpureR: An R Implementation of a Computational Purification Algorithm of Mixed Tumour Profiles.” *BMC Bioinformatics*, **16**. doi:10.1186/s128590150597x. Ogundijo OE, Wang X (2017). “A Sequential Monte Carlo Approach to Gene Expression Deconvolution.” *PLOS ONE*, **12**. doi:10.1371/journal.pone.0186167. Hafemeister C, Satija R (2019). “Normalization and variance stabilization of single-cell RNA-seq data using regularized negative binomial regression.” *Genome Biology*, **20**(1), 296. doi:10.1186/s1305901918741. Chion M, Leroy A (2023). “A Bayesian Framework for Multivariate Differential Analysis Accounting for Missing Data.” arXiv. doi:10.48550/arxiv.2307.08975. Newman A, Liu C, Green M, Gentles A, Feng W, Xu Y, Hoang C, Diehn M, Alizadeh A (2015). “Robust Enumeration of Cell Subsets from Tissue Expression Profiles.” *Nature methods*, **12**. doi:10.1038/nmeth.3337.

See Also

`deconvolute_ratios_Marquardt_Levenberg()`

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
Y <- simulate_bulk_mixture(mu, Sigma, p = c(0.5, 0.5), n = 2)$Y
deconvolute_ratios(
  signature_matrix = mu,
  bulk_expression = Y,
  true_ratios = c(0.5, 0.5),
```

```

Sigma = Sigma,
deconvolution_functions = list(
  "nnls" = list(FUN = deconvolute_ratios_nnls)
),
cores = 1
)

```

deconvolute_ratios_cibersort

DeCovarT MLE of cellular proportions for one bulk sample

Description

Estimates $\hat{\mathbf{p}} = \arg \max_{\mathbf{p}} \ell_{\mathbf{y}|\zeta}(\mathbf{p})$ under the Gaussian convolution model

$$\mathbf{y} | (\zeta, \mathbf{p}) \sim \mathcal{N}_G(\boldsymbol{\mu}\mathbf{p}, \boldsymbol{\Sigma}(\mathbf{p})), \quad \boldsymbol{\Sigma}(\mathbf{p}) = \sum_{j=1}^J p_j^2 \boldsymbol{\Sigma}_j,$$

subject to the simplex constraint $\mathbf{1}^T \mathbf{p} = 1, \mathbf{p} \geq \mathbf{0}$. Optimisation is performed in unconstrained ILR coordinates $\mathbf{z} \in \mathbb{R}^{J-1}$ via $\mathbf{p} = \text{softmax}(\mathbf{V}\mathbf{z})$ (Marquardt–Levenberg default; see other methods below and vignette("theory-decovart-generative-model", package = "DeCovarT")).

Usage

```

deconvolute_ratios_cibersort(y, mean_signature_matrix)

deconvolute_ratios_rlm(y, mean_signature_matrix)

deconvolute_ratios_nnls(y, mean_signature_matrix)

deconvolute_ratios_deconrnaseq(y, mean_signature_matrix)

deconvolute_ratios_Marquardt_Levenberg(
  y,
  mean_signature_matrix,
  Sigma,
  epsilon = 10^-4,
  itmax = 200,
  return_model = FALSE,
  initial_p = NULL,
  dirichlet_alpha = 1
)

deconvolute_ratios_simulated_annealing(
  y,
  mean_signature_matrix,
  Sigma,

```

```

    epsilon = 10^-4,
    itmax = 200,
    return_model = FALSE,
    initial_p = NULL,
    dirichlet_alpha = 1
)

deconvolute_ratios_L_BFGS_B(
  y,
  mean_signature_matrix,
  Sigma,
  epsilon = 10^-4,
  itmax = 200,
  return_model = FALSE,
  initial_p = NULL,
  dirichlet_alpha = 1
)

deconvolute_ratios_Newton_Raphson(
  y,
  mean_signature_matrix,
  Sigma,
  epsilon = 10^-4,
  itmax = 200,
  return_model = FALSE,
  initial_p = NULL,
  dirichlet_alpha = 1
)

deconvolute_ratios_gradient_descent(
  y,
  mean_signature_matrix,
  Sigma,
  epsilon = 10^-4,
  itmax = 200,
  return_model = FALSE,
  initial_p = NULL,
  dirichlet_alpha = 1
)

```

Arguments

y	Numeric vector (or one-column matrix) $\mathbf{y} \in \mathbb{R}^G$.
mean_signature_matrix	Numeric matrix $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ (plug-in means).
Sigma	Array of cell-type covariances in $\mathcal{M}_{G \times G \times J}$.
epsilon, itmax	Absolute convergence tolerance and maximum number of iterations for the optimiser (same roles as reltol / maxit in <code>stats::optim()</code>).

return_model	If TRUE, return a named list with coefficients, ILR coordinates, log-likelihood and optimiser diagnostics instead of the proportion vector.
initial_p	Optional start: NULL / "barycentre" (default equi-balanced), a length- J numeric vector, "dirichlet", or "qp" (mean-only simplex QP). See starting_simplex() . Starts on a simplex face are nudged into the interior so the ILR map is defined (ILR methods) and so L-BFGS-B does not start on a degenerate $\Sigma(p)$.
dirichlet_alpha	Dirichlet concentration when initial_p = "dirichlet" (default 1: uniform on the simplex; $\alpha > 1$ centre-biased; $\alpha < 1$ face-biased). Independent draws are independent restarts; use multistart_decovart() to sequence several.

Details

Plug-in signature. Argument mean_signature_matrix is the mean μ , used as a proxy for the unobserved cell-type profiles $x_{.j}$. This is the frequentist plug-in; recovering sample-specific latents is a Bayesian / MAP problem ([.map_gaussian_convolution\(\)](#)).

Value

Named numeric vector $\hat{p}$ on the simplex (ILR methods), or that list when return_model = TRUE. Benchmark metrics are computed by [deconvolute_ratios\(\)](#).

Functions

- [deconvolute_ratios_cibersort\(\)](#): Linear baseline $\hat{y} = \mu\hat{p}$ via nu-SVR (CIBERSORT-style); no covariance prior is used.
- [deconvolute_ratios_rlm\(\)](#): Robust linear model $y \approx \mu p$ ([MASS::rlm\(\)](#)), as in Monaco et al. (2019).
- [deconvolute_ratios_nnls\(\)](#): Non-negative least squares for $y \approx \mu p$ ([nnls::nnls\(\)](#)), then simplex projection.
- [deconvolute_ratios_deconrnaseq\(\)](#): Equality- and inequality-constrained least squares on the simplex ([limSolve::lsei\(\)](#)), in the spirit of deconRNASeq.
- [deconvolute_ratios_simulated_annealing\(\)](#): Simulated annealing on ILR coordinates z ([stats::optim\(\)](#) with method = "SANN").
- [deconvolute_ratios_L_BFGS_B\(\)](#): Box-constrained L-BFGS-B directly in p ([stats::optim\(\)](#) method = "L-BFGS-B"). The box keeps each coordinate in $[0, 1]$; the returned vector is closed by $p / \sum p$ (no [repair_simplex\(\)](#) clipping). Wald standard errors still use the ILR expected-Fisher map [vcov_ilr_delta\(\)](#), not a p -space Hessian.
- [deconvolute_ratios_Newton_Raphson\(\)](#): Newton–Raphson / [nlminb](#) on ILR coordinates z using analytic gradient and Hessian ([stats::nlminb\(\)](#)).
- [deconvolute_ratios_gradient_descent\(\)](#): BFGS quasi-Newton ascent on ILR coordinates z ([stats::optim\(\)](#) method = "BFGS").

References

Monaco G, Lee B, Xu W, Mustafah S, Hwang YY, Carré C, Burdin N, Visan L, Ceccarelli M, Poidinger M, Zippelius A, Pedro de Magalhães J, Larbi A (2019). “RNA-Seq Signatures Normalized by mRNA Abundance Allow Absolute Deconvolution of Human Immune Cell Types.” *Cell Reports*, **26**. doi:10.1016/j.celrep.2019.01.041.

See Also

`deconvolute_ratios()`, `isometric_logistic()`, `starting_simplex()`

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
y <- drop(mu %*% c(0.6, 0.4) + rnorm(2, sd = 0.1))
deconvolute_ratios_Marquardt_Levenberg(y, mu, Sigma, itmax = 50)
```

deconvolute_ratios_gls

Generalised least squares with a fixed residual covariance

Description

Fits $y \approx \mu p$ by `MASS::lm.gls()` with a **known** $G \times G$ residual covariance W that does not depend on p , then `repair_simplex()`. Use `fixed_gls_covariance()` for the global-diagonal competitor $\text{diag}\{\sum_j \bar{p}_j^2 \Sigma_j\}$. This is the mean-only GLS gold standard for covariance-structure benchmarks, not the DeCovarT convolution $\Sigma(p) = \sum_j p_j^2 \Sigma_j$.

Usage

```
deconvolute_ratios_gls(y, mean_signature_matrix, W, inverse = TRUE)
```

Arguments

<code>y</code>	Numeric bulk vector of length G .
<code>mean_signature_matrix</code>	Numeric matrix $G \times J$.
<code>W</code>	Residual covariance (when <code>inverse = TRUE</code>) or precision (when <code>inverse = FALSE</code>), of size $G \times G$.
<code>inverse</code>	Passed to <code>MASS::lm.gls()</code> ; TRUE means W is a covariance matrix.

Value

Named simplex vector of length J .

See Also

[fixed_gls_covariance\(\)](#), [deconvolute_ratios_deconrnaseq\(\)](#)

Examples

```
mu <- cbind(ct1 = c(20, 40), ct2 = c(40, 20))
y <- drop(mu %*% c(0.4, 0.6))
Sigma <- array(c(diag(2), 2 * diag(2)), dim = c(2, 2, 2))
deconvolute_ratios_gls(y, mu, fixed_gls_covariance(Sigma))
```

describe_simulation_scenario

Describe a Gaussian-convolution simulation scenario

Description

Summarises the generating law $\mathbf{y} \mid (\boldsymbol{\zeta}, \mathbf{p}) \sim \mathcal{N}_G(\boldsymbol{\mu}\mathbf{p}, \sum_j p_j^2 \boldsymbol{\Sigma}_j)$ at five layers: composition, mean geometry, covariance / SPD diagnostics, network sparsity, and tangent Fisher information (mean versus covariance split). MixSim BarOmega and average pairwise Hellinger of the component Gaussians are kept in descriptors (Hellinger is the unweighted pairwise mean; hellinger_weighted is optional). Jeffreys / symmetrised KL is returned in supplementary. Mean-geometry columns include the highest pairwise cosine and the mean Euclidean (CLD) gap.

Usage

```
describe_simulation_scenario(
  true_theta,
  adjacency = NULL,
  active_tol = 1e-08,
  include_mixsim = TRUE
)
```

Arguments

true_theta	List with p, mu, and sigma (and optionally Theta precision). Parsed by check_true_theta() / .parse_true_theta() .
adjacency	Optional $G \times G \times J$ adjacency array, or a length- J list of adjacencies, used for network density when supplied.
active_tol	Threshold for counting an active simplex component.
include_mixsim	Logical. If TRUE (default), compute MixSim BarOmega via compute_average_overlap() . Set FALSE to skip.

Value

A list with:

- `theta_true`: the convolution parameters p, μ, σ ;
- `descriptors`: one-row tibble of kept scenario statistics in six families (composition, mean geometry, SPD of $\Sigma(p)$, tangent Fisher, network, component overlap). MixSim BarOmega uses `compute_average_overlap()` (Davies quadrature for $G < 4$, Sobol Monte Carlo otherwise). Average pairwise Hellinger and affine-invariant Riemannian distance of the Σ_j are kept alongside. `hellinger_weighted` is optional. SPD columns include both $\kappa\{\Sigma(p)\}$ (`kappa_sigma_p`) and the reciprocal $\lambda_{\min}/\lambda_{\max}$ (`kappa_sigma_reciprocal`);
- `supplementary`: one-row tibble of Jeffreys / symmetrised KL, recorded but not treated as a primary score;
- `call`: the matched call (`match.call()`).

See Also

`run_simulation_benchmark()`, `expected_fisher_unconstrained()`, `compute_shannon_entropy()`, `composition_from_entropy()`, `helmert_basis()`, `compute_average_overlap()`, `compute_average_riemannian()`

Examples

```
theta <- list(
  p = c(0.5, 0.5),
  mu = cbind(c(10, 0), c(0, 10)),
  sigma = array(c(diag(2), diag(2)), dim = c(2, 2, 2))
)
out <- describe_simulation_scenario(theta)
out$descriptors$h_star
out$call
```

`equicorrelation_gram` *Equicorrelation (constant-correlation) Gram matrix*

Description

$R = (1 - \rho)I + \rho \mathbf{1}\mathbf{1}^\top$. Positive semidefinite for $\rho \in [-1/(J - 1), 1]$.

Usage

```
equicorrelation_gram(n_celltypes, target_cosine = 0)
```

Arguments

`n_celltypes` Integer $J \geq 2$.
`target_cosine` Pairwise cosine ρ .

Value

Symmetric $J \times J$ matrix with unit diagonal.

Examples

```
equicorrelation_gram(3L, 0.4)
```

```
expected_fisher_unconstrained
```

Expected Fisher information of unconstrained $\mathbf{p}$

Description

For the multivariate-normal mean–covariance map of the DeCovarT convolution, $\mathbf{y} \sim \mathcal{N}_G(\boldsymbol{\mu}\mathbf{p}, \boldsymbol{\Sigma}(\mathbf{p}))$ with $\boldsymbol{\Sigma}(\mathbf{p}) = \sum_j p_j^2 \boldsymbol{\Sigma}_j$ and precision $\boldsymbol{\Omega}(\mathbf{p}) = \boldsymbol{\Sigma}(\mathbf{p})^{-1}$, the expected Fisher information has entries

$$I(\mathbf{p})_{jk} = \boldsymbol{\mu}_{\cdot j}^\top \boldsymbol{\Omega}(\mathbf{p}) \boldsymbol{\mu}_{\cdot k} + 2p_j p_k \operatorname{tr}(\boldsymbol{\Omega}(\mathbf{p}) \boldsymbol{\Sigma}_j \boldsymbol{\Omega}(\mathbf{p}) \boldsymbol{\Sigma}_k).$$

The first summand is the mean contribution (an $\boldsymbol{\Omega}$ -inner product of signature columns); the second is the covariance contribution of the quadratic map $\mathbf{p} \mapsto \boldsymbol{\Sigma}(\mathbf{p})$. See the multivariate-normal formula on https://en.wikipedia.org/wiki/Fisher_information#Multivariate_normal_distribution.

Usage

```
expected_fisher_unconstrained(p, mean_signature_matrix, Sigma)
```

Arguments

`p` Numeric proportions on the open simplex.
`mean_signature_matrix` Mean signature $\boldsymbol{\mu}$ ($G \times J$).
`Sigma` Cell-type covariances $G \times G \times J$.

Value

Symmetric $J \times J$ expected Fisher information matrix $I(\mathbf{p})$.

See Also

[vcov_ilmr_delta\(\)](#), [vcov.decovart_fit\(\)](#), [confint.decovart_fit\(\)](#), [.inner_product\(\)](#)

Examples

```
# Two cell types, bivariate Gaussians: I(p) is 2 x 2.
p <- c(0.6, 0.4)
mu <- cbind(c(0, 0), c(3, 0))
Sigma <- array(c(diag(2), diag(2)), dim = c(2, 2, 2))
I <- expected_fisher_unconstrained(p, mu, Sigma)
I
```

fit_decovart

*Fit the DeCovart Gaussian-convolution model***Description**

Maximum-likelihood estimator of cellular proportions $\mathbf{p}_i$ under the multivariate Gaussian convolution

$$\mathbf{y}_i \mid (\boldsymbol{\mu}, \boldsymbol{\Sigma}_j, \mathbf{p}_i) \sim \mathcal{N}_G(\boldsymbol{\mu} \mathbf{p}_i, \boldsymbol{\Sigma}(\mathbf{p}_i)), \quad \boldsymbol{\Sigma}(\mathbf{p}) = \sum_{j=1}^J p_j^2 \boldsymbol{\Sigma}_j.$$

This is a **variance / likelihood specification**, not ordinary least squares: cell-type covariances enter the residual law and cannot be written as extra columns of $\boldsymbol{\mu}$. There is therefore no formula / lm interface, and no predict() method for forecasting bulk expression.

The returned object is an S3 class decovart_fit with accessors `coef()`, `fitted()`, `residuals()`, `vcov()`, `nobs()`, `confint()`, `print()`, `summary()` and `plot()`. Residuals are $\mathbf{Y} - \boldsymbol{\mu} \hat{\mathbf{P}}$; they are **not** OLS residuals. Goodness of fit is the convolution log-likelihood, not residual sum of squares.

Usage

```
fit_decovart(
  signature_matrix,
  bulk_expression,
  true_ratios = NULL,
  Sigma = NULL,
  method = c("Marquardt-Levenberg", "L-BFGS-B", "Newton-Raphson"),
  epsilon = 10^-4,
  itmax = 200,
  standardise = FALSE,
  scaled = FALSE,
  n_starts = 0L,
  boundary_tol = 1e-08
)

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

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

## S3 method for class 'summary.decovart_fit'
print(x, ...)

## S3 method for class 'decovart_fit'
coef(object, ...)

## S3 method for class 'decovart_fit'
fitted(object, ...)
```

```
## S3 method for class 'decovart_fit'
residuals(object, ...)

## S3 method for class 'decovart_fit'
vcov(object, ...)

## S3 method for class 'decovart_fit'
nobs(object, ...)

## S3 method for class 'decovart_fit'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'decovart_fit'
plot(x, ...)
```

Arguments

signature_matrix	Mean signature $\mu \in \mathcal{M}_{G \times J}$ (rownames = genes, colnames = cell types).
bulk_expression	Bulk matrix $Y \in \mathcal{M}_{G \times N}$ (rownames = genes, colnames = samples).
true_ratios	Optional ground-truth proportions (J vector or $J \times N$ matrix). Stored on the fit; not used by the MLE.
Sigma	Array $(\Sigma_j)_j \in \mathcal{M}_{G \times G \times J}$ of cell-type covariances.
method	Optimiser; one of "Marquardt-Levenberg", "L-BFGS-B", "Newton-Raphson" (case-insensitive). These three maps already land on the simplex (ILR or $p/\sum p$); they do not call <code>repair_simplex()</code> .
epsilon, itmax	Absolute convergence tolerance and iteration budget, in the same roles as <code>reltol</code> / <code>maxit</code> in <code>stats::optim()</code> .
standardise	If TRUE, apply a gene-wise affine z-score computed once from μ to bulk, means and covariances (see Details). Cell-type-wise or sample-wise transforms are not supported.
scaled	Deprecated. TRUE (log2 mixing) always errors.
n_starts	Number of additional random Dirichlet starts per sample. With <code>n_starts > 0</code> the best log-likelihood is kept and <code>multistart_decovart()</code> records the spread of attained optima, the only direct probe of multimodality (the realised log-likelihood is not globally concave).
boundary_tol	Threshold on $\min_j \hat{p}_j$ below which a sample is flagged near_boundary by <code>boundary_diagnostics()</code> . Wald intervals are unreliable there; prefer <code>confint_profile_decovart()</code> or <code>lrt_decovart()</code> .
x, object	A <code>decovart_fit</code> .
...	Passed to <code>graphics::plot()</code> (plot method) or ignored.
parm	Kept for compatibility with <code>stats::confint()</code> ; all simplex coordinates are always returned (subsetting by <code>parm</code> is not implemented). Removing this formal would break the S3 method contract with the generic.

level Confidence level $1 - \alpha$ (default 0.95). Wald intervals use asymptotic normality of the MLE with standard errors from `vcov_ilr_delta()` / `expected_fisher_unconstrained()` (see Details of `fit_decovart()` and of `vcov_ilr_delta()`): $\hat{p}_j \pm z_{1-\alpha/2} \text{SE}_{E_j}$.

Details

Standardisation. CIBERSORT requires non-negative expression, no missing values, and a non-log linear scale (Newman et al. 2015). A logarithm is concave, so Jensen’s inequality shifts first and second moments and $\log(\mu p) \neq (\log \mu)p$. A gene-wise affine map $x \mapsto D^{-1}(x - m)$ with the same D, m on Y, μ and $\Sigma_j^* = D^{-1}\Sigma_j D^{-1}$ leaves the MLE of p unchanged (equivariance).

Wald covariance. Let $\Omega(p) = \Sigma(p)^{-1}$. The expected Fisher information of the unconstrained mean–covariance map (multivariate normal; see e.g. the Wikipedia entry *Fisher information*, multivariate normal) is

$$I(p)_{jk} = \mu_{.j}^\top \Omega(p) \mu_{.k} + 2p_j p_k \text{tr}(\Omega(p) \Sigma_j \Omega(p) \Sigma_k).$$

Cramer–Rao gives $\text{Var}(\hat{z}) \succeq I_z^{-1}$ in ILR coordinates, with $I_z = \mathbf{J}_\psi^\top I(p) \mathbf{J}_\psi$ and $\mathbf{J}_\psi = \mathbf{S}(p) \mathbf{V}$ (`jacobian_isometric_logistic()`). The delta method maps the bound back to the simplex:

$$\text{Var}(\hat{p}) = \mathbf{J}_\psi I_z^{-1} \mathbf{J}_\psi^\top.$$

Value

An object of class `decovart_fit`.

References

Newman A, Liu C, Green M, Gentles A, Feng W, Xu Y, Hoang C, Diehn M, Alizadeh A (2015). “Robust Enumeration of Cell Subsets from Tissue Expression Profiles.” *Nature methods*, **12**. doi:10.1038/nmeth.3337.

See Also

`deconvolute_ratios()`, `deconvolute_ratios_Marquardt_Levenberg()`, `expected_fisher_unconstrained()`, `vcov_ilr_delta()`, `coef.decovart_fit()`, `vcov.decovart_fit()`, `confint.decovart_fit()`

Examples

```
toy <- readRDS(system.file(
  "extdata", "toy_deconvolution.rds",
  package = "DeCovart")
))
fit <- fit_decovart(
  signature_matrix = toy$signature_matrix,
  bulk_expression = toy$bulk_expression,
  Sigma = toy$Sigma,
  itmax = 40
)
coef(fit)
nobs(fit)
```

fixed_gls_covariance *Fixed residual covariance for a GLS competitor*

Description

Evaluates the convolution covariance $\Sigma(\mathbf{p}) = \sum_j p_j^2 \Sigma_j$ at a **known** composition (default the barycentre $p_j = 1/J$) and optionally keeps only the diagonal. That matrix does **not** depend on the unknown $\mathbf{p}$ at fit time. Do not copy it into every slice of a DeCovarT tensor: that would yield $\|\mathbf{p}\|_2^2 \Sigma_{\text{GLS}}$, which still depends on $\mathbf{p}$.

Usage

```
fixed_gls_covariance(Sigma, p = NULL, diagonal = TRUE)
```

Arguments

Sigma	Array $G \times G \times J$ of cell-type covariances.
p	Optional composition of length J ; default $(1/J, \dots, 1/J)$.
diagonal	If TRUE (default), return $\text{diag}\{\Sigma(\mathbf{p})\}$.

Value

A $G \times G$ covariance matrix.

See Also

[deconvolute_ratios_gls\(\)](#)

Examples

```
Sigma <- array(c(diag(2), 2 * diag(2)), dim = c(2, 2, 2))
fixed_gls_covariance(Sigma)
```

generate_mean_signature_matrix

Generate mean profiles from a target Gram matrix

Description

Builds $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ whose *unit* columns realise a prescribed Gram matrix R (pairwise cosines) via the symmetric square root,

$$\boldsymbol{\mu} = s \mathbf{Q} R^{1/2}, \quad \mathbf{Q}^\top \mathbf{Q} = \mathbf{I}_J.$$

Then $\boldsymbol{\mu}^\top \boldsymbol{\mu} = s^2 R$ whenever R has unit diagonal. The default R is the equicorrelation matrix $(1-\rho)\mathbf{I} + \rho \mathbf{1}\mathbf{1}^\top$ ([equicorrelation_gram\(\)](#)). Supply `target_gram` to set unequal pairwise cosines (for example two close pairs and a distant background), provided R is symmetric nonnegative-definite.

Usage

```
generate_mean_signature_matrix(
  n_genes,
  n_celltypes,
  mean_scale = 10,
  target_cosine = 0,
  target_gram = NULL,
  seed = NULL,
  nonnegative = FALSE,
  gene_names = NULL,
  celltype_names = NULL
)
```

Arguments

n_genes	Integer G ; must be at least n_celltypes.
n_celltypes	Integer $J \geq 2$.
mean_scale	Positive scalar s (centroid norms). Default 10. Hold fixed when studying cosine / collinearity alone.
target_cosine	Numeric pairwise cosine ρ for the equicorrelation Gram. Ignored when target_gram is supplied. Must lie in $[-1/(J-1), 1]$.
target_gram	Optional symmetric $J \times J$ cosine matrix (unit diagonal). Overrides target_cosine.
seed	Optional integer. When supplied, the orthonormal frame Q is drawn from a Gaussian QR; otherwise it is deterministic. Ignored when nonnegative = TRUE.
nonnegative	If TRUE, use a disjoint-support nonnegative frame so that $\mu \geq 0$ whenever $R^{1/2} \geq 0$ (required by deconvolute_ratios()). The Gram is unchanged.
gene_names	Optional character vector of length G .
celltype_names	Optional character vector of length J .

Details

Realisations are unique up to an orthogonal transformation of the gene space: replacing Q by QU with $U^T U = I_J$ leaves the Gram unchanged. By default Q is a deterministic thin QR frame; pass seed for a Haar-like Gaussian frame. Cholesky $R = LL^T$ is an alternative square root used in Monte Carlo simulation; see the how-to-build-synthetic-scenarios vignette appendix.

Value

Numeric matrix μ with dimensions $G \times J$.

Examples

```
generate_mean_signature_matrix(
  n_genes = 6L,
  n_celltypes = 2L,
  target_cosine = 0.5
)
```

```
k <- matrix(c(1, 0.98, 0.2, 0.98, 1, 0.2, 0.2, 0.2, 1), 3, 3)
generate_mean_signature_matrix(
  n_genes = 8L,
  n_celltypes = 3L,
  target_gram = k
)
```

helmert_basis	<i>Helmert contrast matrix for isometric log-ratio coordinates</i>
---------------	--

Description

Returns the $J \times (J - 1)$ Helmert sub-matrix $\mathbf{V}$ with

$$\mathbf{V}^T \mathbf{V} = \mathbf{I}_{J-1}, \quad \mathbf{V}^T \mathbf{1} = \mathbf{0}.$$

Column k contrasts the first k parts against part $k + 1$. This is the standard ILR basis used with [isometric_log_ratio\(\)](#) / [isometric_logistic\(\)](#); any other valid ILR basis is $\mathbf{V}Q$ for an orthogonal Q , which rotates coordinates but leaves simplex geometry, eigenvalues of quadratic forms, and $\text{Var}(\hat{p})$ unchanged.

Usage

```
helmert_basis(n_parts)
```

Arguments

`n_parts` Integer $J \geq 2$ (number of simplex parts).

Value

Numeric matrix $J \times (J - 1)$.

Examples

```
V <- helmert_basis(3L)
crossprod(V)
drop(crossprod(V, rep(1, 3)))
```

hessian_isometric_logistic

Hessian tensor of the isometric logistic map

Description

Slice i is

$$\mathbf{H}_{\psi_i} = p_i(\mathbf{q}_i \mathbf{q}_i^\top - \mathbf{C}_V),$$

where $\bar{\mathbf{v}} = \mathbf{V}^\top \mathbf{p}$, $\mathbf{q}_i = \mathbf{V}_{i\cdot} - \bar{\mathbf{v}}$ and $\mathbf{C}_V = \mathbf{V}^\top \mathbf{S}(\mathbf{p}) \mathbf{V}$. Array shape $(J-1) \times (J-1) \times J$.

Usage

```
hessian_isometric_logistic(z, V = NULL)
```

Arguments

$\mathbf{z}$ Numeric vector $\mathbf{z} \in \mathbb{R}^{J-1}$.
 $\mathbf{V}$ Optional ILR basis; see [isometric_logistic\(\)](#).

Value

Numeric array used in the constrained Hessian chain rule.

Examples

```
hessian_isometric_logistic(c(0.2, -0.5))
```

isometric_logistic

Isometric logistic transform (ILR coordinates to the simplex)

Description

Inverse isometric log-ratio map $\psi : \mathbf{z} \mapsto \mathbf{p}$ with a Helmert basis $\mathbf{V}$,

$$\mathbf{p} = \text{softmax}(\mathbf{V}\mathbf{z}) = \mathcal{C}(\exp(\mathbf{V}\mathbf{z})).$$

No cell type is pinned as a reference (unlike [additive_logistic\(\)](#)). Evaluated with a log-sum-exp shift so large $\|\mathbf{z}\|$ cannot overflow.

$$\mathbf{z} = \mathbf{V}^\top \log \mathbf{p} = \mathbf{V}^\top \text{clr}(\mathbf{p}),$$

which is well-defined because $\mathbf{V}^\top \mathbf{1} = 0$. Requires a strictly positive composition (open simplex).

Usage

```
isometric_logistic(z, V = NULL)

isometric_log_ratio(p, V = NULL)
```

Arguments

z Numeric vector $z \in \mathbb{R}^{J-1}$.
V Optional ILR basis; see [isometric_logistic\(\)](#).
p Numeric vector on the open simplex.

Value

Numeric vector p on the unit simplex.
 Numeric vector $z \in \mathbb{R}^{J-1}$.

See Also

[isometric_log_ratio\(\)](#), [helmert_basis\(\)](#), [additive_logistic\(\)](#)

Examples

```
z <- c(0.2, -0.5)
p <- isometric_logistic(z)
sum(p)
isometric_log_ratio(p)
isometric_logistic(c(800, 1200))
```

jacobian_isometric_logistic

Jacobian of the isometric logistic map

Description

$$\mathbf{J}_\psi(z) = \frac{\partial \mathbf{p}}{\partial \mathbf{z}^\top} = \mathbf{S}(\mathbf{p})\mathbf{V},$$

with $\mathbf{S}(\mathbf{p}) = \text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$.

Usage

```
jacobian_isometric_logistic(z, V = NULL)
```

Arguments

z Numeric vector $z \in \mathbb{R}^{J-1}$.
V Optional ILR basis; see [isometric_logistic\(\)](#).

Value

Numeric matrix $J \times (J - 1)$.

Examples

```
jacobian_isometric_logistic(c(0.2, -0.5))
```

lrt_decovart

Likelihood-ratio test for cellular ratios

Description

Tests $H_0 : p_j = c_j, j \in A$ against the unrestricted simplex with the profile likelihood-ratio statistic

$$D = 2\{\ell(\hat{\mathbf{p}}) - \ell(\hat{\mathbf{p}}^{(0)})\},$$

where $\hat{\mathbf{p}}^{(0)}$ is the restricted MLE of `restricted_mle_decovart()`. Unlike the Wald interval of `confint_decovart_fit()`, D is invariant under the additive log-ratio reparametrisation, because a smooth bijection leaves likelihood ratios unchanged.

Usage

```
lrt_decovart(
  y = NULL,
  mean_signature_matrix,
  Sigma,
  null_value,
  bulk_expression = NULL,
  n_boundary = NULL,
  boundary_tol = 1e-06,
  epsilon = 10^-8,
  itmax = 500L
)
```

Arguments

y	Numeric vector (or one-column matrix) $\mathbf{y} \in \mathbb{R}^G$.
mean_signature_matrix	Numeric matrix $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ (plug-in means).
Sigma	Array of cell-type covariances in $\mathcal{M}_{G \times G \times J}$.
null_value	Named numeric vector of hypothesised ratios, e.g. <code>c(celltype_2 = 0)</code> .
bulk_expression	Numeric matrix $\mathbf{Y} \in \mathcal{M}_{G \times N}$ of replicate bulk profiles sharing one composition, used instead of <code>y</code> for the pooled (population) test.
n_boundary	Optional integer override for the number of active boundary constraints; <code>NULL</code> infers it from <code>null_value</code> .
boundary_tol	Distance from 0 or 1 below which a null value counts as being on the boundary.
epsilon, itmax	Relative convergence tolerance and iteration budget passed to <code>stats::optim()</code> .

Details

Interior null. If every tested ratio lies strictly inside $(0, 1)$, D is asymptotically $\chi^2_{|A|}$ (Wilks 1938).

Boundary null. A null such as $p_j = 0$ lets the parameter move in one direction only, so the local parameter space is a half-line rather than a vector space and Wilks' theorem fails. The limit is the chi-bar-square mixture of `chi_bar_square_pvalue()`: for a single active constraint, $D \rightsquigarrow \frac{1}{2}\chi_0^2 + \frac{1}{2}\chi_1^2$ Chernoff (1954), generalised to several active constraints by Self and Liang (1987). Boundary calibration is selected automatically when any tested value is within `boundary_tol` of 0 or 1; override with `n_boundary`.

Replication. Both calibrations are *population* statements: they need replicate bulk samples that genuinely share one composition $\mathbf{p}$. A single G -vector supplies no replication (genes are dependent through $\Sigma(\mathbf{p})$, and adding bulk samples with distinct compositions adds parameters as fast as observations). Use `bulk_expression` with several replicate columns, or calibrate by `bootstrap_decovart()`.

Value

A one-row data frame with columns `statistic`, `n_boundary`, `df_interior`, `p_value`, `loglik_full`, `loglik_null`, and `calibration`.

References

Wilks SS (1938). "The Large-Sample Distribution of the Likelihood Ratio for Testing Composite Hypotheses." *The Annals of Mathematical Statistics*, 9(1), 60–62. ISSN 0003-4851. doi:10.1214/aoms/1177732360.

Self SG, Liang K (1987). "Asymptotic Properties of Maximum Likelihood Estimators and Likelihood Ratio Tests under Nonstandard Conditions." *Journal of the American Statistical Association*, 82(398), 605–610. ISSN 0162-1459. doi:10.1080/01621459.1987.10478472.

See Also

`chi_bar_square_pvalue()`, `confint_profile_decovart()`, `bootstrap_decovart()`

Other `decovart_inference`: `bootstrap_decovart()`, `boundary_diagnostics()`, `chi_bar_square_pvalue()`, `confint_profile_decovart()`, `equivariance_check_decovart()`, `multistart_decovart()`, `profile_loglik_decovart()`, `reference_bootstrap_decovart()`, `restricted_mle_decovart()`

Examples

```
mu <- matrix(c(20, 22, 18, 22, 20, 24), nrow = 2)
colnames(mu) <- paste0("ct", 1:3)
Sigma <- array(c(diag(2), diag(2), diag(2)), dim = c(2, 2, 3))
y <- drop(mu %**% c(0.5, 0.5, 0))
lrt_decovart(y, mu, Sigma, null_value = c(ct3 = 0))
```

multistart_decovart *Refit DeCovart from several random starts to probe multimodality*

Description

The realised DeCovart log-likelihood is not globally concave, so a successful convergence code, a small relative-distance-to-minimum and a negative-definite local Hessian together establish a **local** maximum only. Refitting from independent $\text{Dirichlet}(1, \dots, 1)$ starts and comparing the attained log-likelihoods is the cheapest direct probe of a second, equally good or better mode.

Usage

```
multistart_decovart(
  y,
  mean_signature_matrix,
  Sigma,
  n_starts = 5L,
  solver = deconvolute_ratios_Marquardt_Levenberg,
  loglik_tol = 1e-04,
  dirichlet_alpha = 1,
  ...
)
```

Arguments

y	Numeric vector (or one-column matrix) $\mathbf{y} \in \mathbb{R}^G$.
mean_signature_matrix	Numeric matrix $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ (plug-in means).
Sigma	Array of cell-type covariances in $\mathcal{M}_{G \times G \times J}$.
n_starts	Number of Dirichlet restarts (in addition to the equi-balanced start).
solver	Solver accepting initial_p and return_model; defaults to deconvolute_ratios_Marquardt_Levenberg .
loglik_tol	Two starts count as reaching the same mode when their log-likelihoods differ by less than this amount.
dirichlet_alpha	Concentration for those random starts (default 1). See starting_simplex() .
...	Passed to solver (e.g. epsilon, itmax).

Value

A list with coefficients (best fit), loglik, loglik_range, starts (matrix of per-start estimates), logliks, and multimodal.

See Also

`boundary_diagnostics()`, `fit_decovart()`

Other `decovart_inference`: `bootstrap_decovart()`, `boundary_diagnostics()`, `chi_bar_square_pvalue()`, `confint_profile_decovart()`, `equivariance_check_decovart()`, `lrt_decovart()`, `profile_loglik_decovart()`, `reference_bootstrap_decovart()`, `restricted_mle_decovart()`

Examples

```
mu <- matrix(c(20, 40, 15, 40, 20, 25), nrow = 3)
colnames(mu) <- paste0("ct", 1:2)
Sigma <- array(c(diag(3), diag(3)), dim = c(3, 3, 2))
y <- drop(mu %*% c(0.6, 0.4))
set.seed(1)
multistart_decovart(y, mu, Sigma, n_starts = 3L)$multimodal
```

overlap_gaussian_mc	<i>MixSim-style pairwise overlap via stratified Sobol Monte Carlo</i>
---------------------	---

Description

Estimates the MixSim Omega map $\Omega_{j\ell} = \Pr_{X \sim f_j}(\pi_\ell f_\ell(X) > \pi_j f_j(X))$ by drawing `n_mc` quasi-Monte Carlo points **from each component** (inverse-transform sampling of a Sobol sequence through the Cholesky factor) and comparing **log-densities**. The average pairwise overlap `BarOmega` is the unweighted mean of $\Omega_{j\ell} + \Omega_{\ell j}$ over $j < \ell$, matching `MixSim::overlap()` / **FSDA MixSim**.

Usage

```
overlap_gaussian_mc(true_theta, n_mc = 10000L, seed = NULL, J = NULL)
```

Arguments

<code>true_theta</code>	List with <code>p</code> , <code>mu</code> , <code>sigma</code> as in <code>check_true_theta()</code> .
<code>n_mc</code>	Draws per component (default 10000).
<code>seed</code>	Optional RNG seed (passed to Sobol randomisation and to the uniform fallback).
<code>J</code>	Optional number of components.

Details

For two equal-weight components this pairwise overlap equals the histogram similarity $\int \min(f_j, f_\ell)$ and therefore $1 - \text{TV}(f_j, f_\ell)$ (Nielsen and Sun 2018).

Value

A list with `BarOmega`, `MaxOmega`, and `OmegaMap`.

References

Nielsen F, Sun K (2018). “Guaranteed Deterministic Bounds on the Total Variation Distance between Univariate Mixtures.” arXiv:1806.11311. doi:10.48550/arxiv.1806.11311. <https://arxiv.org/abs/1806.11311>.

See Also

`compute_average_overlap()`, `compute_average_riemannian()`

Examples

```
set.seed(1)
theta <- list(
  p = c(0.5, 0.5),
  mu = cbind(c(0, 0), c(3, 0)),
  sigma = array(c(diag(2), diag(2)), dim = c(2, 2, 2))
)
overlap_gaussian_mc(theta, n_mc = 2000L)$BarOmega
```

pivot_mc_estimates	<i>Pivot Monte Carlo proportion estimates to a long table</i>
--------------------	---

Description

Turns the optimisation block of `run_simulation_benchmark()` into one row per replicate, algorithm, and cell type, with the matching true proportion from `theta_true`. The raincloud intervals drawn from this table are **empirical Monte Carlo quantiles** of $\hat{p}$, not confidence intervals for p (Allen et al. 2019).

Usage

```
pivot_mc_estimates(benchmark)
```

Arguments

`benchmark` List returned by `run_simulation_benchmark()`.

Value

A tibble with `algorithm`, `cell_type`, `estimate`, `p_true`, `error`, `sample_id`, and any scenario metadata columns.

References

Allen M, Poggiali D, Whitaker K, Marshall TR, van Langen J, Kievit RA (2019). “Raincloud Plots: A Multi-Platform Tool for Robust Data Visualization.” *Wellcome Open Research*, **4**, 63. ISSN 2398-502X. doi:10.12688/wellcomeopenres.15191.2. <https://wellcomeopenresearch.org/articles/4-63>.

See Also

[plot_mc_raincloud\(\)](#), [plot_mc_forest\(\)](#)

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
out <- run_simulation_benchmark(
  tibble::tibble(
    ID = "B1",
    true_theta = list(list(p = c(0.5, 0.5), mu = mu, sigma = Sigma))
  ),
  deconvolution_functions = list(
    "nnls" = list(FUN = deconvolute_ratios_nnls)
  ),
  n = 2,
  cores = 1
)
nrow(pivot_mc_estimates(out))
```

plot_algorithm_similarity

Tile heatmap of algorithm-similarity correlations

Description

`ggplot2::geom_tile()` display of [algorithm_similarity\(\)](#), with rows and columns ordered by Ward D2 clustering of $\sqrt{1-r}$. Optional dendrogram via `ggdendro` (Suggests), drawn to the **right** of the tiles with leaves flush against the heatmap and Ward merge heights on the axis. This is the default for a small correlation matrix; [plot_correlation_Heatmap\(\)](#) is reserved for linked multi-omics grids.

Usage

```
plot_algorithm_similarity(
  benchmark,
  facet_rows = NULL,
  facet_cols = NULL,
  dendrogram = FALSE
)
```

Arguments

benchmark	List from <code>run_simulation_benchmark()</code> .
facet_rows, facet_cols	Optional scenario columns. When supplied, correlations are computed inside each facet cell.
dendrogram	If TRUE, attach a <code>ggdendro</code> <code>ggplot</code> to the right of the tiles as attribute "dendrogram" (ignored when scenario facets are used).

Value

A `ggplot` object.

See Also

`algorithm_similarity()`, `plot_mc_metric_dots()`

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
out <- run_simulation_benchmark(
  tibble::tibble(
    true_theta = list(list(p = c(0.5, 0.5), mu = mu, sigma = Sigma))
  ),
  deconvolution_functions = list(
    "nnls" = list(FUN = deconvolute_ratios_nnls),
    "rlm" = list(FUN = deconvolute_ratios_rlm)
  ),
  n = 4,
  cores = 1
)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot_algorithm_similarity(out)
}
```

Description

For each algorithm, visualises a selected score over the design grid of simulated p (and related scenario factors). Requires optional Suggests packages ComplexHeatmap, circlize, and viridis (install ComplexHeatmap via Bioconductor). Use this helper only for **linked, annotated, hierarchical** grids. For algorithm-similarity correlation matrices prefer `plot_algorithm_similarity()` (`ggplot2::geom_tile()`).

Usage

```
plot_correlation_Heatmap(
  distribution_metrics,
  score_variable = "model_mse",
  n_break = 20,
  uni_scale = TRUE,
  file = NULL
)
```

Arguments

<code>distribution_metrics</code>	Tibble of metric scores from a benchmark.
<code>score_variable</code>	Column name of the metric to display ("model_mse", "model_rmse", "model_coef_determination", "model_coef_determination_adjusted", "model_mae", "model_cor", "model_ccc"). Matching is case-insensitive.
<code>n_break</code>	Number of colour breaks.
<code>uni_scale</code>	If FALSE, each panel uses its own colour scale.
<code>file</code>	Optional PDF path. When supplied, heatmaps are drawn with <code>grDevices::pdf()</code> ; a missing .pdf suffix is added (G4.0).

Value

A named list of ComplexHeatmap heatmap objects (one per algorithm).

See Also

`plot_algorithm_similarity()`, `plot_mc_metric_dots()`

Examples

```
metrics <- tibble::tibble(
  correlation_celltype1 = c(0, 0, 0.5, 0.5),
  correlation_celltype2 = c(0, 0.5, 0, 0.5),
  algorithm = "nnls",
  model_mse = c(0.01, 0.02, 0.015, 0.03)
)
if (
  requireNamespace("ComplexHeatmap", quietly = TRUE) &&
  requireNamespace("circlize", quietly = TRUE) &&
  requireNamespace("viridis", quietly = TRUE)
)
```

```

) {
  ht <- plot_correlation_Heatmap(metrics, score_variable = "model_mse")
  names(ht)
}

```

plot_mc_forest

Forest plot of ADEMP Monte Carlo summaries

Description

Dot-and-whisker display of bias, RMSE, MAE, coverage, mean interval width, SE/SD ratio, and optimiser failure rate by algorithm and cell type (Allen et al. 2019). Coverage whiskers are the Wilson interval already stored on monte_carlo (`coverage_mc_interval()`; (Wilson 1927)); they are intervals for the coverage *rate*, not for p_j . Bias is referenced at 0; coverage at 0.95; SE/SD (`se_sd_ratio`, mean model SE / empirical SD) at 1. Interior Wald SEs come from `vcov_ilm_delta()`. Pairwise algorithm contrasts (MAE differences versus a reference solver on the same Monte Carlo replicates) can be read from the raincloud of paired errors; they do not need a second bootstrap.

Usage

```

plot_mc_forest(
  benchmark,
  facet_rows = NULL,
  facet_cols = NULL,
  metrics = c("bias", "rmse", "mae", "coverage", "mean_interval_width", "se_sd_ratio",
    "failure_rate")
)

```

Arguments

benchmark	List from <code>run_simulation_benchmark()</code> , or a table from <code>pivot_mc_estimates()</code> .
facet_rows, facet_cols	Optional column names for <code>ggplot2::facet_grid()</code> rows and columns.
metrics	Character vector of summaries to display.

Value

A ggplot object.

References

Allen M, Poggiali D, Whitaker K, Marshall TR, van Langen J, Kievit RA (2019). “Raincloud Plots: A Multi-Platform Tool for Robust Data Visualization.” *Wellcome Open Research*, **4**, 63. ISSN 2398-502X. doi:10.12688/wellcomeopenres.15191.2. <https://wellcomeopenresearch.org/articles/4-63>.

Wilson EB (1927). “Probable Inference, the Law of Succession, and Statistical Inference.” *Journal of the American Statistical Association*, **22**(158), 209–212. ISSN 0162-1459. doi:10.2307/2276774.

See Also

[plot_mc_raincloud\(\)](#), [coverage_mc_interval\(\)](#), [vcov_ilr_delta\(\)](#), [theme_decovart_facets\(\)](#)

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
out <- run_simulation_benchmark(
  tibble::tibble(
    n_genes = 2L,
    cosine = 0,
    true_theta = list(list(p = c(0.5, 0.5), mu = mu, sigma = Sigma))
  ),
  deconvolution_functions = list(
    "nnls" = list(FUN = deconvolute_ratios_nnls)
  ),
  n = 4,
  cores = 1
)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot_mc_forest(out, facet_cols = "cosine")
}
```

plot_mc_metric_dots *Faceted dot plot of several ADEMP metrics*

Description

One panel per metric; colour is a **min-max normalised** score with $\backslash(1) = \text{best inside that metric}$ (and facet). Do not map a second primary score to point size. Coverage is scored as $|\hat{C} - 0.95|$, so over-wide intervals that inflate coverage are not rewarded. There is no default weighted composite.

Usage

```
plot_mc_metric_dots(
  benchmark,
  facet_rows = NULL,
  facet_cols = NULL,
  metrics = c("rmse", "mae", "coverage", "mean_interval_width"),
  weights = NULL
)
```

Arguments

benchmark	List from <code>run_simulation_benchmark()</code> , or a table from <code>pivot_mc_estimates()</code> .
facet_rows, facet_cols	Optional column names for <code>ggplot2::facet_grid()</code> rows and columns.
metrics	Character vector of summaries to display.
weights	Optional named numeric weights (sum to 1) used only to add a composite facet beside the components.

Value

A ggplot object.

See Also

`plot_mc_forest()`, `plot_algorithm_similarity()`

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
out <- run_simulation_benchmark(
  tibble::tibble(
    cosine = 0,
    true_theta = list(list(p = c(0.5, 0.5), mu = mu, sigma = Sigma))
  ),
  deconvolution_functions = list(
    "nnls" = list(FUN = deconvolute_ratios_nnls)
  ),
  n = 4,
  cores = 1
)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot_mc_metric_dots(out, facet_cols = "cosine")
}
```

plot_mc_raincloud	<i>Horizontal raincloud of Monte Carlo proportion estimates</i>
-------------------	---

Description

Half-eye densities, dots, and empirical 50% / 95% intervals (Allen et al. 2019) for the Monte Carlo sampling distribution of $\hat{p}_j$. Default aesthetics: numeric axis is the estimation error $\hat{p}_j - p_j^*$ (so a vertical line at 0 is bias); the y-axis is the cell type; colour and dodge group the deconvolution algorithm. Optional facet_rows / facet_cols split scenarios (for example number of genes versus pairwise cosine).

Usage

```
plot_mc_raincloud(
  benchmark,
  quantity = c("error", "estimate"),
  facet_rows = NULL,
  facet_cols = NULL,
  .width = c(0.5, 0.95),
  include_dots = TRUE,
  max_dots = 2000L,
  max_rows = NULL,
  dodge_width = 0.95,
  slab_scale = 1.4,
  slab_alpha = 1,
  category_spacing = 1,
  slab = c("halfeye", "dotsinterval")
)
```

Arguments

benchmark	List from <code>run_simulation_benchmark()</code> , or a table from <code>pivot_mc_estimates()</code> .
quantity	"error" (default) or "estimate".
facet_rows, facet_cols	Optional column names for <code>ggplot2::facet_grid()</code> rows and columns.
.width	Passed to <code>ggdist::stat_halfeye()</code> ; default <code>c(0.5, 0.95)</code> .
include_dots	If FALSE, omit <code>ggdist::stat_dots()</code> (use this on large Monte Carlo tables: the Fortran QP in <code>nudge_bins</code> cannot take long vectors).
max_dots	Maximum rows passed to <code>stat_dots</code> when <code>include_dots = TRUE</code> . Extra rows are subsampled.
max_rows	Optional cap on the plotting table (half-eye and dots). When the Monte Carlo stack is huge, subsample before drawing.
dodge_width	Width passed to <code>ggplot2::position_dodge()</code> .
slab_scale	Slab height for <code>ggdist::stat_halfeye()</code> .

slab_alpha	Transparency of the density slab (1 is opaque).
category_spacing	Vertical gap between discrete y categories (cell types). Values greater than 1 insert extra space so slabs from neighbouring types do not overlap; the axis still shows the original labels.
slab	"halfeye" (default KDE raincloud) or "dotsinterval" (quantile dots plus interval; avoids empty bounded slabs).

Details

When `quantity = "estimate"` and `slab = "halfeye"`, slabs use a bounded kernel on $[0, 1]$ (`ggdist::density_bounded()`) with a coloured outline (`slab_colour / slab_linewidth`). Vertex pile-up can leave that KDE empty; `slab = "dotsinterval"` (`ggdist::stat_dotsinterval()`) draws quantile dots instead. True proportions are labelled at the top of each panel (the most abundant type to the right of its reference line; the others to the left).

The inner interval is the central 50% of Monte Carlo replicates; the outer interval is the central 95%. These are **not** confidence intervals for p_j .

Value

A ggplot object.

References

Allen M, Poggiali D, Whitaker K, Marshall TR, van Langen J, Kievit RA (2019). "Raincloud Plots: A Multi-Platform Tool for Robust Data Visualization." *Wellcome Open Research*, **4**, 63. ISSN 2398-502X. doi:10.12688/wellcomeopenres.15191.2. <https://wellcomeopenresearch.org/articles/4-63>.

See Also

`plot_mc_forest()`, `pivot_mc_estimates()`

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
out <- run_simulation_benchmark(
  tibble::tibble(
    n_genes = 2L,
    cosine = 0,
    true_theta = list(list(p = c(0.5, 0.5), mu = mu, sigma = Sigma))
  ),
```

```

deconvolution_functions = list(
  "nnls" = list(FUN = deconvolute_ratios_nnls)
),
n = 4,
cores = 1
)
if (requireNamespace("ggdist", quietly = TRUE)) {
  plot_mc_raincloud(out, facet_cols = "cosine")
}

```

reference_bootstrap_decovart

Reference-based bootstrap for signature and composition uncertainty

Description

The frequentist DeCovarT likelihood treats μ and each Σ_j as known. In a labelled, reference-based model those moments come from purified profiles whose *cell-type names stay attached*. This bootstrap resamples the experimental units that generated the reference (donors, by default), or redraws compositions from a Dirichlet law, then re-estimates the plug-in moments and refits (Efron 1979).

Usage

```

reference_bootstrap_decovart(
  y = NULL,
  reference_profiles,
  bulk_expression = NULL,
  cell_type_labels = NULL,
  donor_ids = NULL,
  method = c("donors", "cells", "dirichlet"),
  n_boot = 199L,
  level = 0.95,
  dirichlet_alpha = 1,
  regenerate_bulk = FALSE,
  p = NULL,
  n_replicates = NULL,
  ridge = 1e-06,
  epsilon = 10^-8,
  itmax = 500L
)

```

Arguments

y Numeric vector (or one-column matrix) $\mathbf{y} \in \mathbb{R}^G$.

reference_profiles	Named list of $G \times n_j$ matrices, one purified sample matrix per cell type. Names must match the intended column order of the signature. Alternatively, a single $G \times n_{\text{ref}}$ matrix together with cell_type_labels.
bulk_expression	Numeric matrix $\mathbf{Y} \in \mathcal{M}_{G \times N}$ of replicate bulk profiles sharing one composition, used instead of $\mathbf{y}$ for the pooled (population) test.
cell_type_labels	Optional character or factor vector of length <code>ncol(reference_profiles)</code> when reference_profiles is a matrix.
donor_ids	Donor (or other clustering) labels aligned with the purified columns. A named list of vectors matching reference_profiles, or a vector of length <code>ncol(reference_profiles)</code> when the reference is a single matrix. Required for <code>method = "donors"</code> .
method	One of "donors" (default), "cells", or "dirichlet".
n_boot	Number of bootstrap replicates.
level	Confidence level for percentile intervals.
dirichlet_alpha	Positive Dirichlet concentration: a scalar recycled to J , or a length- J vector. Ignored unless <code>method = "dirichlet"</code> . The default 1 is uniform on the simplex.
regenerate_bulk	If TRUE (donors / cells only), simulate a new bulk from the bootstrapped moments instead of keeping the observed $\mathbf{Y}$ fixed.
p	Composition to simulate from. Defaults to the MLE computed from $\mathbf{y}$ / bulk_expression.
n_replicates	Number of bulk columns simulated per replicate; defaults to the number of observed columns.
ridge	Non-negative ridge multiple for the sample covariance of each cell type. The default $1\text{e-}6$ is relative to the mean diagonal.
epsilon, itmax	Relative convergence tolerance and iteration budget passed to <code>stats::optim()</code> .

Details

Permuting gene order, or permuting cell-type labels of an already averaged signature, is **not** a bootstrap: the first destroys the gene-wise pairing of bulk and reference, and the second is algebraic equivariance rather than a source of uncertainty (see `equivariance_check_decovart()` and `vignette("theory-DeCovarT-MLE-properties")`). The three method options instead resample units that actually vary.

donors Cluster bootstrap: resample donor identifiers *within each cell type*, take every purified column of the sampled donors (a donor drawn twice is included twice), rebuild $(\boldsymbol{\mu}, \boldsymbol{\Sigma}_j)$, and refit. This is the default: it targets biological / reference-population uncertainty rather than technical cell-level noise alone.

cells Resample purified columns independently within each cell type (with replacement) and rebuild the plug-in moments. Use this when donor labels are unavailable.

dirichlet Hold the original plug-in moments fixed, draw $\mathbf{p}^{(b)} \sim \text{Dirichlet}(\boldsymbol{\alpha})$, simulate bulk profiles from the Gaussian convolution, and refit. This is a composition-sweep diagnostic, not a confidence interval for one observed bulk.

For donors and cells the observed bulk is kept fixed unless regenerate_bulk = TRUE, in which case each replicate also draws $\mathbf{Y}^{(b)}$ from the bootstrapped moments at $\mathbf{p}$ (the observed MLE by default). That is the pipeline reference units $\rightarrow S^{(b)} \rightarrow Y^{(b)} \rightarrow \hat{\mathbf{p}}^{(b)}$.

When a cell type has fewer purified columns than genes, the sample covariance is singular. A ridge ridge * mean(diag(S)) is then added on the diagonal so that $\Sigma(\mathbf{p})$ remains factorisable. Set ridge = 0 to disable that guard (and accept possible Cholesky failures).

Value

A list with method, estimates ($J \times n_{\text{boot}}$), interval (percentile bounds), mean_signature_matrix / Sigma from the original reference, and p_hat from the original plug-in fit when a bulk is supplied. For method = "dirichlet", p_simulated stores the drawn compositions.

References

Efron B (1979). "Bootstrap Methods: Another Look at the Jackknife." *The Annals of Statistics*, 7(1), 1–26. doi:10.1214/aos/1176344552.

See Also

`bootstrap_decovart()`, `equivariance_check_decovart()`

Other decovart_inference: `bootstrap_decovart()`, `boundary_diagnostics()`, `chi_bar_square_pvalue()`, `confint_profile_decovart()`, `equivariance_check_decovart()`, `lrt_decovart()`, `multistart_decovart()`, `profile_loglik_decovart()`, `restricted_mle_decovart()`

Examples

```
set.seed(1)
mu <- matrix(c(20, 40, 15, 40, 20, 25), nrow = 3)
colnames(mu) <- paste0("ct", 1:2)
Sigma <- array(c(diag(3), diag(3)), dim = c(3, 3, 2))
refs <- lapply(seq_len(2), function(j) {
  t(MASS::mvrnorm(n = 8, mu = mu[, j], Sigma = Sigma[, , j]))
})
names(refs) <- colnames(mu)
donor_ids <- lapply(refs, function(x) {
  rep(paste0("d", 1:2), length.out = ncol(x))
})
y <- drop(mu %*% c(0.6, 0.4))
reference_bootstrap_decovart(
  y,
  refs,
  donor_ids = donor_ids,
  n_boot = 15
)$interval
```

 repair_simplex

Repair a numeric vector onto the unit simplex

Description

Clips numerical under/overflow and renormalises so that $\mathbf{1}^T \mathbf{p} = 1$ and $\mathbf{p} \geq \mathbf{0}$. This is a **repair / renormalisation** step for estimated proportions, not a Euclidean projection onto the simplex and not a statistical-identifiability constraint.

When open = TRUE, entries on the closed-simplex boundary are nudged into the relative interior so that `isometric_log_ratio()` is defined (ILR / Marquardt / Newton starts). That open-simplex path is the former role of `starting_simplex()`.

Usage

```
repair_simplex(
  p,
  tolerance = 100 * .Machine$double.eps,
  open = FALSE,
  nms = NULL
)
```

Arguments

p	Numeric vector $\mathbf{p} \in \mathbb{R}^J$.
tolerance	Non-negative tolerance for treating entries as zero (default $100 * .Machine$double.eps$).
open	Logical; if TRUE, push the repaired vector off the boundary into the open simplex.
nms	Optional names attached to the returned vector (e.g. cell-type colnames).

Value

Numeric vector on the simplex Δ^{J-1} (open when open = TRUE).

See Also

`compositions::clo()` for compositional closure.

Examples

```
repair_simplex(c(0.2, 0.3, 0.5 + 1e-12))
repair_simplex(c(1, 0, 0), open = TRUE)
```

run_simulation_benchmark

Simulate bulk mixtures and benchmark deconvolution algorithms

Description

Wrapper around [simulate_bulk_mixture\(\)](#), [deconvolute_ratios\(\)](#), and [compute_benchmark_metrics\(\)](#) (called inside [deconvolute_ratios\(\)](#)). Each row of `scenario_config` describes one generative model $(\mu, (\Sigma_j)_j, p)$ stored in a list column `true_theta`. Scenario rows are always evaluated **sequentially** to avoid nested parallelism; sample-level workers live only in [deconvolute_ratios\(\)](#). Scenario builders (factorial grids, overlap summaries, etc.) should live in analysis scripts; see `scripts/fig02_bivariate_toy.R` and the paper-scenario vignettes.

Usage

```
run_simulation_benchmark(
  scenario_config,
  deconvolution_functions,
  n = 200,
  standardise = FALSE,
  scaled = FALSE,
  cores = 1L,
  coverage_interval = "wilson",
  verbose = FALSE,
  progress_every = 10L
)
```

Arguments

<code>scenario_config</code>	Tibble or list of scenario rows. Each row must contain a <code>true_theta</code> list column (or list element) with at least <code>mu</code> and <code>sigma</code> . Optional per-row <code>n</code> overrides the default <code>n</code> .
<code>deconvolution_functions</code>	Named list passed to deconvolute_ratios() ; each element has <code>FUN</code> and optional <code>additional_parameters</code> for <code>do.call()</code> .
<code>n</code>	Default number of bulk replicates N when <code>scenario_config</code> has no <code>n</code> column.
<code>standardise</code> , <code>scaled</code>	Passed to deconvolute_ratios() .
<code>cores</code>	Workers for the per-sample loop inside deconvolute_ratios() . Defaults to 1L.
<code>coverage_interval</code>	Coverage interval for the Monte Carlo coverage <i>rate</i> ; see coverage_mc_interval() .
<code>verbose</code>	If TRUE, print each scenario row and (when the grid has at most 10 scenarios) every <code>progress_every</code> inferred samples. Large factorial grids log one line per scenario only, so logs stay readable.

`progress_every` Sample-progress interval passed to `deconvolute_ratios()` when verbose is TRUE. Defaults to 10L.

Value

A list with:

- `regression`: global (per-sample composition scores) and `cell_type` (across-sample Pearson / F1 / spillover);
- `monte_carlo`: ADEMP summaries per cell type;
- `optimisation`: per-sample elapsed time, memory, KKT residual, and $\hat{p}$;
- `config`: tibble of scenario metadata (one row per scenario);
- `theta_true`: list of convolution parameters (p, μ, Σ_j) actually used to draw the bulk;
- `descriptors`: kept scenario statistics (composition, mean, SPD, network, tangent Fisher, MixSim BarOmega, pairwise Hellinger);
- `supplementary`: Jeffreys overlap, recorded separately;
- `call`: the matched call (`match.call()`). Scripts should persist those pieces with `write_simulation_artefacts()` rather than saving the whole list.

See Also

`simulate_bulk_mixture()`, `deconvolute_ratios()`, `compute_benchmark_metrics()`, `describe_simulation_scenario()`, `coverage_mc_interval()`, `plot_mc_raincloud()`, `plot_mc_forest()`, `write_simulation_artefacts()`

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
scenario_config <- tibble::tibble(
  ID = "B1_Ho",
  true_theta = list(list(
    p = c(0.5, 0.5),
    mu = mu,
    sigma = Sigma
  ))
)
out <- run_simulation_benchmark(
  scenario_config = scenario_config,
  deconvolution_functions = list(
    "nnls" = list(FUN = deconvolute_ratios_nnls)
  ),
  n = 2,
```

```

    cores = 1
  )
  nrow(out$optimisation)

```

scale_covariance_array

Scale component covariances, keeping precision zeros

Description

Replaces each Σ_j by $s\Sigma_j$. Then $\Omega_j(s) = \Omega_j/s$, so the **support** of the precision (structural zeros from the graph) is unchanged.

Usage

```
scale_covariance_array(sigma, scale)
```

Arguments

sigma	$G \times G \times J$ covariance array.
scale	Positive scalar s .

Value

Scaled array.

See Also

[scale_covariances_to_overlap\(\)](#)

Examples

```

sigma <- array(c(diag(2), diag(2)), dim = c(2, 2, 2))
scaled <- scale_covariance_array(sigma, 4)
scaled[, , 1]

```

simulate_bulk_mixture *Simulate bulk mixtures from a multivariate Gaussian convolution*

Description

For each bootstrap sample $i = 1, \dots, N$, draws **latent** purified profiles $\mathbf{x}_{\cdot j}^{(i)} \sim \mathcal{N}_G(\boldsymbol{\mu}_{\cdot j}, \boldsymbol{\Sigma}_j)$ independently for each cell type $j = 1, \dots, J$, then forms the bulk by the linear convolution

$$\mathbf{y}_{\cdot i} = \mathbf{X}^{(i)} \mathbf{p} = \sum_{j=1}^J p_j \mathbf{x}_{\cdot j}^{(i)},$$

matching the article's conditional model $\mathbf{y} \mid (\boldsymbol{\zeta}, \mathbf{p}) \sim \mathcal{N}_G(\boldsymbol{\mu} \mathbf{p}, \boldsymbol{\Sigma}(\mathbf{p}))$ with $\boldsymbol{\Sigma}(\mathbf{p}) = \sum_j p_j^2 \boldsymbol{\Sigma}_j$.

The mean signature $\boldsymbol{\mu}$ is shared across samples; only the latent draws $\mathbf{x}_{\cdot j}^{(i)}$ vary with i . Stacking those draws into the three-way array $\mathcal{X} = (x_{gji}) \in \mathcal{M}_{G \times J \times N}$, the bulk matrix is the mode-2 tensor–vector contraction

$$\mathbf{Y} = \mathcal{X} \times_2 \mathbf{p}, \quad y_{gi} = \sum_{j=1}^J x_{gji} p_j \quad (g = 1, \dots, G; i = 1, \dots, N),$$

which for each sample recovers $\mathbf{y}_{\cdot i} = \mathbf{X}^{(i)} \mathbf{p}$ with $\mathbf{X}^{(i)} \in \mathcal{M}_{G \times J}$.

Usage

```
simulate_bulk_mixture(
  signature_matrix,
  Sigma,
  p = rep(1/ncol(signature_matrix), ncol(signature_matrix)),
  n = 500,
  truncate_negative = FALSE
)
```

Arguments

signature_matrix	Mean signature $\boldsymbol{\mu} = (\mu_{gj}) \in \mathcal{M}_{G \times J}$ (shared across samples; not the latent profiles).
Sigma	Array of covariances $(\boldsymbol{\Sigma}_j)_j \in \mathcal{M}_{G \times G \times J}$.
p	Proportion vector $\mathbf{p} \in \Delta^{J-1}$ (default: uniform).
n	Number of bulk / bootstrap samples N .
truncate_negative	If TRUE, replace negative bulk entries with 0 so $\mathbf{Y}$ is a valid RNA-seq-style non-negative matrix. The latent Gaussian draws are unchanged. deconvolute_ratios() allows negative bulk entries (with a warning): Gaussian convolutions can produce them. Flooring is optional, not required.

Value

A list with:

- latent_profiles: array $\mathcal{X} = (x_{gji}) \in \mathcal{M}_{G \times J \times N}$ of **unobserved** cell-type-specific draws (one $G \times J$ slice per sample i);
- Y: matrix $\mathbf{Y} \in \mathcal{M}_{G \times N}$ whose columns are bulk vectors $\mathbf{y}_{\cdot i}$, obtained as $\mathcal{X} \times_2 \mathbf{p}$.

See Also

[deconvolute_ratios\(\)](#), [run_simulation_benchmark\(\)](#)

Examples

```
set.seed(1)
genes <- paste0("g", 1:2)
cts <- paste0("ct", 1:2)
mu <- matrix(c(20, 22, 22, 20), nrow = 2, dimnames = list(genes, cts))
Sigma <- array(
  c(1, 0, 0, 1, 1, 0, 0, 1),
  dim = c(2, 2, 2),
  dimnames = list(genes, genes, cts)
)
sim <- simulate_bulk_mixture(mu, Sigma, p = c(0.5, 0.5), n = 5)
dim(sim$Y)
```

simulate_hierarchical_grn_moments

Simulate GRN first- and second-order moments

Description

Builds a mean matrix $\boldsymbol{\mu} \in \mathcal{M}_{G \times J}$ and **cell-type-specific** second-order moments $(\boldsymbol{\Omega}_j, \boldsymbol{\Sigma}_j = \boldsymbol{\Omega}_j^{-1})_{j=1}^J$ under a graph-constrained precision model. Means realise a target Gram matrix via the symmetric square root of `generate_mean_signature_matrix()` (equicorrelation ρ by default, or a full target_gram). For each cell type, an adjacency is drawn from a random-graph model (or supplied), i.i.d. signed weights with inhibitory fraction `prop_inhibitory` form $\mathbf{W}_j$, and the precision is completed by a spectral shift. Distinct cell types receive **independent** precision draws by default (biology rarely shares one network across types); pass a length- J `graph_model` / `graph_params` or a pre-built adjacency list / array for hybrid designs.

Usage

```
simulate_hierarchical_grn_moments(
  n_genes,
  n_celltypes = 2L,
  mean_scale = 10,
  target_cosine = 0,
  target_gram = NULL,
```

```

seed = NULL,
nonnegative = FALSE,
precision_shift,
precision_scale,
prop_inhibitory = 0.5,
graph_model = c("erdos_renyi", "hub", "star", "scale_free", "stochastic_block_model",
  "small_world"),
graph_params = list(),
adjacency = NULL
)

```

Arguments

n_genes	Integer G ; must be at least n_celltypes.
n_celltypes	Integer $J \geq 2$.
mean_scale	Positive scalar s (centroid norms). Default 10. Hold fixed when studying cosine / collinearity alone.
target_cosine	Numeric pairwise cosine ρ for the equicorrelation Gram. Ignored when target_gram is supplied. Must lie in $[-1/(J-1), 1]$.
target_gram	Optional symmetric $J \times J$ cosine matrix (unit diagonal). Overrides target_cosine.
seed	Optional integer. When supplied, the orthonormal frame $\mathbf{Q}$ is drawn from a Gaussian QR; otherwise it is deterministic. Ignored when nonnegative = TRUE.
nonnegative	If TRUE, use a disjoint-support nonnegative frame so that $\mu \geq 0$ whenever $R^{1/2} \geq 0$ (required by deconvolute_ratios()). The Gram is unchanged.
precision_shift	Diagonal cushion u for the spectral shift (scalar, or length J).
precision_scale	Positive magnitude v of signed off-diagonal precision weights (scalar, or length J).
prop_inhibitory	Numeric in $[0, 1]$; fraction of edges with positive precision weight (inhibitory partial correlation). Default 0.5 balances inhibitory and activatory edges (scalar, or length J).
graph_model	One of "erdos_renyi", "hub", "scale_free", "stochastic_block_model", "small_world", or a character vector of length J .
graph_params	Named list of generator parameters (shared), or a list of length J of such named lists (see generate_random_network_skeleton()).
adjacency	Optional pre-built undirected adjacencies: a list of $J \times G \times G$ matrices, or a $G \times G \times J$ array. When supplied, graph_model / graph_params are ignored for skeleton generation.

Value

Named list with:

- mean_profiles: matrix μ ;

- covariance_matrices: array $(\Sigma_j)_j \in \mathcal{M}_{G \times G \times J}$;
- precision_matrices: array $(\Omega_j)_j \in \mathcal{M}_{G \times G \times J}$;
- graph_structure: adjacency_matrices, weighted_adjacencies, and normalised_precision (all $G \times G \times J$);
- objectives: mean_abs_cosine and sum_euclidean_distance.

Examples

```
set.seed(42)
moments <- simulate_hierarchical_grn_moments(
  n_genes = 40L,
  n_celltypes = 3L,
  mean_scale = 10,
  target_cosine = 0.1,
  precision_shift = 0.1,
  precision_scale = 0.3,
  prop_inhibitory = 0.5,
  graph_model = "scale_free"
)
str(moments, max.level = 2)
```

spd_affine_invariant_distance

Affine-invariant Riemannian (AIRM) distance between two SPD matrices

Description

AIRM is the **affine-invariant Riemannian metric** on the cone of symmetric positive-definite matrices: $d_R(A, B) = \|\log(A^{-1/2}BA^{-1/2})\|_F$. Do **not** use the Frobenius $\|A - B\|_F$: that Euclidean chord leaves the cone, is not inversion-invariant, and suffers the swelling effect.

Usage

```
spd_affine_invariant_distance(a, b)
```

Arguments

a, b Symmetric positive-definite matrices of equal size.

Value

Non-negative scalar.

See Also

[compute_average_riemannian\(\)](#)

Examples

```
# Two bivariate Gaussians: identity vs a correlated SPD covariance.
a <- diag(2)
b <- matrix(c(2, 0.5, 0.5, 1), nrow = 2)
spd_affine_invariant_distance(a, a)
d <- spd_affine_invariant_distance(a, b)
d > 0
```

starting_simplex

Open-simplex start for ILR solvers

Description

The convolution log-likelihood is not globally concave, so the start can change which basin the solver enters. Draw several independent Dirichlet starts (different RNG streams or [multistart_decovart\(\)](#)) rather than a single draw when probing multimodality.

Usage

```
starting_simplex(
  n_celltypes,
  initial_p = NULL,
  nms = NULL,
  y = NULL,
  mean_signature_matrix = NULL,
  dirichlet_alpha = 1
)
```

Arguments

n_celltypes Integer J .

initial_p One of:

- NULL or "barycentre": equi-balanced ($1/J, \dots, 1/J$);
- a numeric vector of length J : used as-is after [repair_simplex\(\)](#);
- "dirichlet": one $\text{Dirichlet}(\alpha, \dots, \alpha)$ draw (see `dirichlet_alpha`);
- "qp" (aliases "deconrnaseq", "lse1"): mean-only simplex QP from [deconvolute_ratios_decon](#) (y and mean_signature_matrix required).

nms Optional names (cell-type colnames).

y, mean_signature_matrix

Bulk and signature, required for a QP start.

dirichlet_alpha

Positive concentration, recycled to length J . The default 1 is uniform on the simplex. $\alpha > 1$ concentrates mass near the barycentre; $\alpha < 1$ puts extra mass near faces (boundary-biased restarts).

Value

A length- J open-simplex vector.

See Also

[multistart_decovart\(\)](#), [deconvolute_ratios_deconrnaseq\(\)](#)

Examples

```
starting_simplex(3L)
set.seed(1)
starting_simplex(3L, "dirichlet")
```

theme_decovart_facets *Faceted ggplot2 theme (black strips, panel border)*

Description

theme_minimal() plus a panel border and white-on-black facet strips, after theme_features() in the [atlas-feature-selection-benchmark](#) plotting helpers.

Usage

```
theme_decovart_facets(base_size = 11, ...)
```

Arguments

base_size	Base font size for ggplot2::theme_minimal() .
...	Passed to ggplot2::theme() .

Value

A ggplot2 theme object.

See Also

[plot_mc_forest\(\)](#), [plot_mc_raincloud\(\)](#)

Examples

```
ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
  ggplot2::geom_point() +
  ggplot2::facet_wrap(~cyl) +
  theme_decovart_facets()
```

vcov_ilr_delta

Cramer–Rao / ILR delta-method covariance of $\hat{\mathbf{p}}$ **Description**

Maps the expected Fisher information of unconstrained proportions through the isometric log-ratio (ILR) chart and back to the simplex via the delta method.

Let $\mathbf{p} = \text{softmax}(\mathbf{V}\mathbf{z})$ with Jacobian $\mathbf{J}_\psi = \mathbf{S}(\mathbf{p})\mathbf{V}$ ([jacobian_isometric_logistic\(\)](#)). Fisher information transforms as

$$I_z = \mathbf{J}_\psi^T I(\mathbf{p}) \mathbf{J}_\psi = \mathbf{V}^T \mathbf{S}(\mathbf{p}) I(\mathbf{p}) \mathbf{S}(\mathbf{p}) \mathbf{V}.$$

The first-order delta method then yields

$$\text{Var}(\hat{\mathbf{p}}) \approx \mathbf{J}_\psi I_z^{-1} \mathbf{J}_\psi^T.$$

This simplex covariance is invariant to orthogonal rotations of $\mathbf{V}$. The construction is undefined on the simplex boundary (the log-ratio chart blows up); the function then returns NA with a warning.

Wald intervals in [vcov.decovart_fit\(\)](#), [confint.decovart_fit\(\)](#) and [deconvolute_ratios\(\)](#) all use this expected-Fisher ILR map. The observed Hessian [hessian_loglik_constrained\(\)](#) is for Newton steps only; it is not the Wald covariance.

Usage

```
vcov_ilr_delta(p, mean_signature_matrix, Sigma, warn = TRUE)
```

Arguments

<code>p</code>	Numeric proportions on the open simplex.
<code>mean_signature_matrix</code>	Mean signature $\boldsymbol{\mu}$ ($G \times J$).
<code>Sigma</code>	Cell-type covariances $G \times G \times J$.
<code>warn</code>	If FALSE, skip the boundary / singularity warnings (Monte Carlo loops). Default TRUE.

Value

Symmetric $J \times J$ asymptotic covariance of $\hat{\mathbf{p}}$, or a matrix of NA if the bound is undefined / singular.

See Also

[expected_fisher_unconstrained\(\)](#), [vcov.decovart_fit\(\)](#), [confint.decovart_fit\(\)](#), [jacobian_isometric_logistic\(\)](#)

Examples

```
# Two cell types: a single unconstrained ILR coordinate maps back to  
# a rank-1 covariance on the simplex (rows sum to zero).  
p <- c(ct1 = 0.6, ct2 = 0.4)  
mu <- cbind(ct1 = c(0, 0), ct2 = c(3, 0))  
Sigma <- array(c(diag(2), diag(2)), dim = c(2, 2, 2))  
v <- vcov_ilr_delta(p, mu, Sigma)  
round(v, 4)  
abs(rowSums(v)) < 1e-10
```

Index

- * **decovart_fit**
 - fit_decovart, 30
- * **decovart_inference**
 - bootstrap_decovart, 5
 - chi_bar_square_pvalue, 8
 - confint_profile_decovart, 18
 - lrt_decovart, 38
 - multistart_decovart, 40
 - reference_bootstrap_decovart, 51
- .inner_product(), 29
- additive_log_ratio(additive_logistic), 3
- additive_logistic, 3
- additive_logistic(), 36, 37
- algorithm_similarity, 4
- algorithm_similarity(), 43, 44
- bootstrap_decovart, 5, 9, 19, 39, 41, 53
- bootstrap_decovart(), 9, 39, 53
- boundary_diagnostics, 7, 9, 19, 39, 41, 53
- boundary_diagnostics(), 31, 41
- check_true_theta, 7
- check_true_theta(), 11–13, 27, 41
- chi_bar_square_pvalue, 7, 8, 19, 39, 41, 53
- chi_bar_square_pvalue(), 7, 39
- coef(), 30
- coef.decovart_fit(fit_decovart), 30
- coef.decovart_fit(), 32
- composition_from_entropy, 10
- composition_from_entropy(), 18, 28
- compositions::alr(), 3
- compositions::alrInv(), 4
- compositions::clo(), 54
- compute_average_jeffreys, 10
- compute_average_jeffreys(), 8, 16
- compute_average_overlap, 11
- compute_average_overlap(), 8, 11, 16, 27, 28, 42
- compute_average_riemannian, 12
- compute_average_riemannian(), 28, 42, 61
- compute_benchmark_metrics, 13
- compute_benchmark_metrics(), 21, 55, 56
- compute_glmnet_gene_scores, 15
- compute_glmnet_gene_scores(), 11
- compute_shannon_entropy, 17
- compute_shannon_entropy(), 10, 28
- confint(), 30
- confint.decovart_fit(fit_decovart), 30
- confint.decovart_fit(), 19, 22, 29, 32, 38, 64
- confint_profile_decovart, 7, 9, 18, 39, 41, 53
- confint_profile_decovart(), 31, 39
- coverage_mc_interval, 20
- coverage_mc_interval(), 46, 47, 55, 56
- deconvolute_ratios, 20
- deconvolute_ratios(), 25, 26, 32, 34, 55, 56, 58–60, 64
- deconvolute_ratios_cibersort, 23
- deconvolute_ratios_deconrnaseq(deconvolute_ratios_cibersort), 23
- deconvolute_ratios_deconrnaseq(), 27, 62, 63
- deconvolute_ratios_gls, 26
- deconvolute_ratios_gls(), 33
- deconvolute_ratios_gradient_descent(deconvolute_ratios_cibersort), 23
- deconvolute_ratios_L_BFGS_B(deconvolute_ratios_cibersort), 23
- deconvolute_ratios_Marquardt_Levenberg(deconvolute_ratios_cibersort), 23
- deconvolute_ratios_Marquardt_Levenberg(), 22, 32, 40

- deconvolute_ratios_Newton_Raphson
(deconvolute_ratios_cibersort),
[23](#)
- deconvolute_ratios_nnlsl
(deconvolute_ratios_cibersort),
[23](#)
- deconvolute_ratios_rlm
(deconvolute_ratios_cibersort),
[23](#)
- deconvolute_ratios_simulated_annealing
(deconvolute_ratios_cibersort),
[23](#)
- describe_simulation_scenario, [27](#)
- describe_simulation_scenario(), [56](#)
- equicorrelation_gram, [28](#)
- equicorrelation_gram(), [33](#)
- equivariance_check_decovart, [7](#), [9](#), [19](#), [39](#),
[41](#), [53](#)
- equivariance_check_decovart(), [52](#), [53](#)
- expected_fisher_unconstrained, [29](#)
- expected_fisher_unconstrained(), [28](#), [32](#),
[64](#)
- fit_decovart, [30](#)
- fit_decovart(), [32](#), [41](#)
- fitted(), [30](#)
- fitted.decovart_fit(fit_decovart), [30](#)
- fixed_gls_covariance, [33](#)
- fixed_gls_covariance(), [26](#), [27](#)
- generate_mean_signature_matrix, [33](#)
- ggdist::density_bounded(), [50](#)
- ggdist::stat_dots(), [49](#)
- ggdist::stat_dotsinterval(), [50](#)
- ggdist::stat_halfeye(), [49](#)
- ggplot2::facet_grid(), [46](#), [48](#), [49](#)
- ggplot2::position_dodge(), [49](#)
- ggplot2::theme(), [63](#)
- ggplot2::theme_minimal(), [63](#)
- glmnet::glmnet(), [15](#), [16](#)
- graphics::plot(), [31](#)
- grDevices::pdf(), [45](#)
- helmert_basis, [35](#)
- helmert_basis(), [28](#), [37](#)
- hessian_isometric_logistic, [36](#)
- hessian_loglik_constrained(), [64](#)
- isometric_log_ratio
(isometric_logistic), [36](#)
- isometric_log_ratio(), [35](#), [37](#), [54](#)
- isometric_logistic, [36](#)
- isometric_logistic(), [3](#), [26](#), [35–37](#)
- jacobian_isometric_logistic, [37](#)
- jacobian_isometric_logistic(), [32](#), [64](#)
- limSolve::lsei(), [25](#)
- lrt_decovart, [7](#), [9](#), [19](#), [38](#), [41](#), [53](#)
- lrt_decovart(), [5](#), [7](#), [9](#), [18](#), [19](#), [31](#)
- MASS::lm.gls(), [26](#)
- MASS::rlm(), [25](#)
- match.call(), [28](#), [56](#)
- MixSim::overlap(), [11](#), [41](#)
- multistart_decovart, [7](#), [9](#), [19](#), [39](#), [40](#), [53](#)
- multistart_decovart(), [25](#), [31](#), [62](#), [63](#)
- nnls::nnls(), [25](#)
- nobs(), [30](#)
- nobs.decovart_fit(fit_decovart), [30](#)
- overlap_gaussian_mc, [41](#)
- overlap_gaussian_mc(), [11](#), [12](#)
- pivot_mc_estimates, [42](#)
- pivot_mc_estimates(), [5](#), [46](#), [48–50](#)
- plot(), [30](#)
- plot.decovart_fit(fit_decovart), [30](#)
- plot_algorithm_similarity, [43](#)
- plot_algorithm_similarity(), [5](#), [45](#), [48](#)
- plot_correlation_Heatmap, [44](#)
- plot_correlation_Heatmap(), [43](#)
- plot_mc_forest, [46](#)
- plot_mc_forest(), [43](#), [48](#), [50](#), [56](#), [63](#)
- plot_mc_metric_dots, [47](#)
- plot_mc_metric_dots(), [44](#), [45](#)
- plot_mc_raincloud, [49](#)
- plot_mc_raincloud(), [43](#), [47](#), [56](#), [63](#)
- print(), [30](#)
- print.decovart_fit(fit_decovart), [30](#)
- print.summary.decovart_fit
(fit_decovart), [30](#)
- profile_loglik_decovart, [7](#), [9](#), [19](#), [39](#), [41](#),
[53](#)
- reference_bootstrap_decovart, [7](#), [9](#), [19](#),
[39](#), [41](#), [51](#)

reference_bootstrap_decovart(), [6](#), [7](#)
repair_simplex, [54](#)
repair_simplex(), [22](#), [25](#), [26](#), [31](#), [62](#)
residuals(), [30](#)
residuals.decovart_fit (fit_decovart),
 [30](#)
restricted_mle_decovart, [7](#), [9](#), [19](#), [39](#), [41](#),
 [53](#)
restricted_mle_decovart(), [38](#)
run_simulation_benchmark, [55](#)
run_simulation_benchmark(), [5](#), [28](#), [42](#), [44](#),
 [46](#), [48](#), [49](#), [59](#)

scale_covariance_array, [57](#)
scale_covariances_to_overlap(), [57](#)
simulate_bulk_mixture, [58](#)
simulate_bulk_mixture(), [55](#), [56](#)
simulate_hierarchical_grn_moments, [59](#)
spd_affine_invariant_distance, [61](#)
spd_affine_invariant_distance(), [13](#)
starting_simplex, [62](#)
starting_simplex(), [25](#), [26](#), [40](#), [54](#)
stats::confint(), [31](#)
stats::nlminb(), [25](#)
stats::optim(), [6](#), [19](#), [24](#), [25](#), [31](#), [38](#), [52](#)
stats::pchisq(), [9](#)
summary(), [30](#)
summary.decovart_fit (fit_decovart), [30](#)

theme_decovart_facets, [63](#)
theme_decovart_facets(), [47](#)

vcov(), [30](#)
vcov.decovart_fit (fit_decovart), [30](#)
vcov.decovart_fit(), [3](#), [29](#), [32](#), [64](#)
vcov_ilr_delta, [64](#)
vcov_ilr_delta(), [14](#), [22](#), [25](#), [29](#), [32](#), [46](#), [47](#)

write_simulation_artefacts(), [56](#)