Package: BatchVaria (via r-universe)

November 25, 2026

Title Variance-Aware Evaluation of Batch Correction

Version 1.0.0

Description Evaluates batch correction in omics data by quantifying how a correction redistributes variance, not only whether it removes it. Records correction and variance-decomposition provenance in the metadata of an ordinary SummarizedExperiment, and harmonises both the batch correction methods and the variance-profiling engines behind documented contracts, so that either set can be extended without modifying the package. Variance fractions are compositional, so removing variance associated with one term inflates the fraction attributed to every other. Each decomposition is therefore recorded alongside the total variance it is relative to and the share an unbalanced design cannot attribute to any single term, which is what distinguishes a correction that removed unwanted variation from one that merely removed variation.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.5.0), SummarizedExperiment

Roxygen list(markdown = TRUE)

biocViews GeneExpression, RNASeq, BatchEffect, StatisticalMethod, Software, Normalization, QualityControl, Transcriptomics, DataRepresentation, Visualization, Sequencing

Suggests testthat (>= 3.0.0), knitr, rmarkdown, BiocStyle, limma

VignetteBuilder knitr

Imports S4Vectors, rlang, methods, variancePartition, stats, ggplot2, FNN, reformulas, sva, tidyr, tidyselect, dplyr, grid

Config/testthat/edition 3

URL https://github.com/cmpmolonc/BatchVaria

BugReports https://github.com/cmpmolonc/BatchVaria/issues

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libpng-dev libxml2-dev libssl-dev zlib1g-dev

Repository Bioconductor 3.24

Date/Publication 2026-11-23 22:57:07 UTC

RemoteUrl https://github.com/bioc/BatchVaria

RemoteRef RELEASE_3_24

RemoteSha 544fb6b9ea8acba67937c017b75cbf0ca66991f2

Contents

assayVariance . . . 3
availableCorrectionMethods . . . 4
availableVarianceMethods . . . 4
basisProjection . . . 5
basisRetention . . . 6
BatchVaria-requirements . . . 7
compareCorrelations . . . 9
comparePCA . . . 10
evaluateCorrections . . . 11
exampleBatchVaria . . . 12
newVarianceSummary . . . 13
plotBasisProjection . . . 14
plotBatchEntropy . . . 15
plotCorrelationChange . . . 16
plotPCATrajectory . . . 17
plotSampleDistance . . . 18
plotVarianceComposition . . . 18
plotVarianceDelta . . . 20
plotVarianceRadar . . . 20
profileVariance . . . 21
provenance . . . 23
recordVariance . . . 24
registerCorrectionMethod . . . 25
registerVarianceEngine . . . 27
runCorrection . . . 29
unregisterCorrectionMethod . . . 31
unregisterVarianceEngine . . . 31
varianceChange . . . 32
varianceDelta . . . 33
varianceHistory . . . 34
varianceResults . . . 35
varianceTable . . . 35

Index **38**

`assayVariance` *Total variance of each assay*

Description

Reports the total variance carried by each assay, independent of any variance decomposition. Total variance is the sum of per-feature variances, i.e. the trace of the feature covariance matrix.

Usage

```
assayVariance(object, assays = NULL)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assays`	Character vector of assay names (default: all assays).

Details

Variance fractions reported by `profileVariance` are compositional: they sum to one within an assay. A correction that removes variance associated with one term therefore *increases* the fraction attributed to every remaining term, even when the absolute variance of those terms is unchanged or reduced.

Because of this, a rising variance fraction cannot on its own be read as preserved or enhanced signal. `assayVariance()` supplies the denominator, so that a fraction can be converted to an absolute quantity:

$$absolutevariance = variancefraction \times totalvariance$$

Comparing absolute variances across assays is immune to the compositional effect and is the appropriate basis for judging whether a correction preserved biological signal.

Assays with fewer than two samples yield `NA` per-feature variances, which are excluded from the total.

Value

A data.frame with columns `assay`, `total_variance`, `n_features` and `n_samples`, one row per assay.

See Also

`varianceResults`, `profileVariance`

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100, nSamples = 8)
bv <- runCorrection(bv, method = "combat", batch = "batch")
assayVariance(bv, assays = c("raw", "raw_combat"))
```

`availableCorrectionMethods` *Available correction methods*

Description

Returns the names of the registered batch correction methods.

Usage

```
availableCorrectionMethods()
```

Value

Character vector of method identifiers

See Also

`registerCorrectionMethod`

Examples

```
availableCorrectionMethods()
```

`availableVarianceMethods` *Available variance profiling methods*

Description

Returns the names of supported variance decomposition engines.

Usage

```
availableVarianceMethods()
```

Details

The list is the contents of the engine registry, so it grows when `registerVarianceEngine` is called. Every engine listed here answers the same question – how much of the variance is attributable to each term of a model – so their results share a schema and can be compared directly.

PCA is deliberately not among them. It is unsupervised, so its output is not attributable to any covariate, and its terms are latent axes rather than model terms: putting both in one `variance_fraction` column under one `term` key would assert a commensurability that does not hold. PCA remains available as an embedding through `comparePCA`, `basisRetention` and the plotting layer, where a shared reference basis makes cross-assay comparison well defined.

Value

Character vector of method identifiers

See Also

`registerVarianceEngine` to add one; `comparePCA` for the PCA-based view.

Examples

```
availableVarianceMethods()
```

`basisProjection` *Sample coordinates on a reference assay's principal axes*

Description

Projects each assay onto a basis fitted once on a reference assay and returns the sample coordinates.

Usage

```
basisProjection(object, assays = NULL, nPCs = 2, reference = NULL)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assays`	Character vector of assays to project (default: all).
`nPCs`	Number of components to return (default: 2).
`reference`	Assay supplying the basis. Inferred from the correction lineage when `NULL`, falling back to the first assay.

Details

Every assay is projected onto one basis, using the reference's own centring and scaling, so coordinates are directly comparable across assays: a sample that has not moved has the same position in every assay. Fitting a PCA separately per assay would make `PC1` a different direction in each, so the same sample would land in different positions depending only on which assay it was drawn from. Fitting over a chosen pair of assays would make the position depend on what the assay was compared against.

Because the axes are the reference's, they are principal components of the reference and not of the assay being projected. Coordinates for a corrected assay describe where its samples sit in the original structure, which is the comparison of interest; they are not that assay's own principal components and the components are not ordered by its variance. `comparePCA` reports how much of each assay's variance each axis carries, and `basisRetention` sums that over the whole basis.

This is the function `plotBasisProjection` draws, exported so the coordinates are available without going through a plot.

Value

A data.frame with one row per assay/sample combination and columns `assay`, `sample`, `reference` and one column per requested component, named `PC1`, `PC2` and so on.

See Also

`basisRetention` for the variance summary, `comparePCA` for the per-component breakdown, `plotBasisProjection` to plot the coordinates.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- runCorrection(bv, method = "combat", batch = "batch")
head(basisProjection(bv, assays = c("raw", "raw_combat")))
```

---

basisRetention *Variance retained on a reference assay's principal axes*

Description

Reports how much of each assay's variance lies along the principal directions of a reference assay, and how much has moved off them.

Usage

```
basisRetention(object, assays = NULL, reference = NULL)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assays`	Character vector of assays to measure (default: all).
`reference`	Assay supplying the basis. Inferred from the correction lineage when `NULL`, falling back to the first assay.

Details

`retention` is the fraction of an assay's total variance that the reference's principal axes account for; `off_basis` is the remainder. For the reference itself retention is one by construction.

The interpretation is a single distinction that no independently fitted PCA can draw: a correction that redistributes variance *within* the structure already present keeps retention near one, while a correction that introduces new structure drives it down. On the example data ComBat retains about 0.99 while an assay distorted with added noise retains about 0.41 - both look like changed variance in a per-assay decomposition, but only the second has moved variance into directions the original data did not occupy.

Features with no variance in the reference are kept in the basis on their original scale rather than dropped, so variance a correction introduces in a previously flat feature is counted as off-basis instead of disappearing from the measurement.

Value

A data.frame with one row per assay and columns `assay`, `reference`, `retention`, `off_basis` and `n_features`.

See Also

`comparePCA` for the per-component breakdown.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- runCorrection(bv, method = "combat", batch = "batch")
basisRetention(bv, assays = c("raw", "raw_combat", "raw_noise"))
```

`BatchVaria-requirements` *Requirements BatchVaria places on a SummarizedExperiment*

Description

Every public function that takes an object checks it with the internal `.check_se()` before using it, and that check is the whole of what BatchVaria requires of an object. An object is acceptable if it is a `SummarizedExperiment` whose ledgers, if it has any, are well formed and refer to things it still has.

Details

The requirements are:

- The object is a `SummarizedExperiment`. Expression matrices are assays; sample annotation is `colData`.
- `metadata(object)$correction_history`, if present, is a list of entries each naming a `method`, an `assay_in`, an `assay_out` and a `batch`. The assays it names must exist, and `batch` and `preserve` must name `colData` columns.
- `metadata(object)$variance_history`, if present, is a list of entries each naming an `assay` and a `method` and carrying a `result` data.frame with `term` and `variance_fraction` columns. The assay it names must exist.

Both ledgers are optional. A plain `SummarizedExperiment` with neither is a valid input to `runCorrection` and `profileVariance`, which is how one is normally built: the ledgers are what those functions write.

Value

No return value. This page documents the requirements BatchVaria places on a `SummarizedExperiment` rather than a function; `exampleBatchVaria` returns an object that meets them.

Why not a class

These rules were previously a `setValidity()` method on a `BatchVariaData` class. S4 validity runs at construction and on an explicit `validObject()`, and at no other point – notably not on `[`, `assay<-`, `colData<-` or `metadata<-`. Checking at the point of use sees the object as it is when a function is asked to work with it, which is the only moment the answer matters.

See Also

`BatchVaria` for an overview of the package, `provenance` for reading the ledgers, and `exampleBatchVaria` for an object that satisfies them.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
class(bv)
SummarizedExperiment::assayNames(bv)

## an ordinary SummarizedExperiment is accepted; the ledgers are
## written by the functions that need them
se <- SummarizedExperiment::SummarizedExperiment(
    assays = list(raw = SummarizedExperiment::assay(bv, "raw")),
    colData = SummarizedExperiment::colData(bv)
)
assayVariance(se)
```

`compareCorrelations` *Compare sample correlation structure across assays*

Description

Computes the mean absolute difference in sample–sample correlation matrices between a baseline assay and one or more comparison assays.

Usage

```
compareCorrelations(object, baseline = "raw", assays = NULL)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`baseline`	Baseline assay name. When `NULL` (the default) it is inferred from the correction ledger, falling back to `"raw"`.
`assays`	Character vector of assays to compare (default: all assays)

Details

Samples are the columns of each assay, so the correlation matrices are computed over columns and are `ncol(assay)` by `ncol(assay)`. The reported change is averaged over the unique sample pairs (the strict upper triangle): the diagonal is identical in every assay by construction, and each off-diagonal pair appears twice in a symmetric matrix.

Value

data.frame with columns: assay, correlation_change

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- runCorrection(bv, method = "combat", batch = "batch")
compareCorrelations(bv)
```

`comparePCA` *Compare PCA variance structure across assays*

Description

Computes the proportion of variance explained by principal components for one or more assays.

Usage

```
comparePCA(object, assays = NULL, nPCs = 3, reference = NULL)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assays`	Character vector of assays (default: all assays)
`nPCs`	Number of principal components to return (default: 3)
`reference`	Assay whose PCA basis the others are projected onto. Inferred from the correction lineage when `NULL`, falling back to the first assay.

Details

Every assay is projected onto a single basis fitted on `reference`, rather than each being decomposed independently. A per-assay fit makes `PC1` a different linear combination of features in every row, with an arbitrary sign and a rank order that reshuffles when correction removes a dominant axis: the label stays put while the direction it names does not, so subtracting one row from another compares two unrelated quantities.

The trade-off is that the reference is privileged, which is stated rather than incidental. A second, related choice: the embedding standardises features to unit variance before decomposing, so every feature contributes equally to sample geometry regardless of how much variance it carries. The variance engines do not standardise, so PCA and the variance ledger describe variance on different footings and their numbers should not be read against each other. Standardising is the usual choice for an embedding whose purpose is showing sample relationships; it would be the wrong choice for attributing variance to covariates, which is why PCA is not one of the variance engines.

Fractions are expressed relative to each assay's own total variance, so they sum to one for the reference and to less than one for any assay whose variance has moved into directions the reference basis does not span. That shortfall is informative: a correction that reshapes variance within the existing structure keeps the sum near one, while one that introduces new structure drives it down.

Value

data.frame with columns: assay, component, variance, reference

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- runCorrection(bv, method = "combat", batch = "batch")
comparePCA(bv)
```

`evaluateCorrections` *Evaluate batch correction performance across assays*

Description

Runs a suite of diagnostic comparisons between a baseline assay and one or more corrected assays, including variance decomposition, PCA structure, and sample correlation analysis.

Usage

```
evaluateCorrections(
  object,
  baseline = "raw",
  diagnostics = c("variance", "pca", "correlation"),
  assays = NULL,
  varianceMethod = "anova"
)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`baseline`	Character string specifying the reference assay (default `"raw"`).
`diagnostics`	Character vector of diagnostics to compute. Supported options include `"variance"`, `"pca"`, and `"correlation"`.
`assays`	Optional character vector of assay names to evaluate. If `NULL`, all assays except the baseline are used.
`varianceMethod`	Character string specifying the variance decomposition method to use (default `"anova"`).

Details

This function acts as a high-level orchestration layer, combining multiple evaluation strategies into a unified interface. By default, all supported diagnostics are run across all non-baseline assays.

Value

A named list containing diagnostic results for each assay comparison.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch, methods = "anova")
res <- evaluateCorrections(bv, assays = c("raw", "raw_center"))
names(res)
```

`exampleBatchVaria` *Example BatchVaria dataset with batch and biological effects*

Description

Generates a synthetic `SummarizedExperiment` meeting BatchVaria's requirements (see BatchVaria-requirements), with several pre-generated corrected assays for vignette demonstrations.

Usage

```
exampleBatchVaria(nGenes = 800, nSamples = 20, confounding = 0)
```

Arguments

`nGenes`	Number of genes. Must be at least 10, so that each effect block (group, batch, both) contains at least one gene.
`nSamples`	Number of samples. Must be a multiple of 4 and at least 4, giving a 2 x 2 design of two batches by two groups. The design is balanced unless `confounding` is raised above zero.
`confounding`	Degree of batch/group confounding, from 0 (an orthogonal 2 x 2 design) to 1 (batch and group fully aliased). Intermediate values place a corresponding majority of each group in one batch.

Details

The `raw_center` and `raw_scale` assays are retained as no-op controls rather than as correction methods. Per-feature centring leaves every variance component untouched, and uniform scaling multiplies them all equally, so both leave the variance composition identical to `raw` and both retain the raw principal axes exactly (`basisRetention` returns 1). They are useful precisely because they should register as doing nothing: a diagnostic that reports a change on either of them is reporting an artefact.

The expression matrix and the pre-generated `raw_noise` assay are drawn with `rnorm`, so repeated calls return different data. Seeding is deliberately left to the caller rather than done inside the function: call `set.seed()` beforehand whenever you need reproducible output.

Value

A `SummarizedExperiment`.

See Also

set.seed

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100, nSamples = 8)
bv
SummarizedExperiment::assayNames(bv)

## confounding slides the design from orthogonal to aliased
balanced <- exampleBatchVaria(nGenes = 100, confounding = 0)
table(balanced$batch, balanced$group)

confounded <- exampleBatchVaria(nGenes = 100, confounding = 0.6)
table(confounded$batch, confounded$group)
```

---

newVarianceSummary *Build a conforming variance summary*

Description

Constructs the data.frame that profileVariance expects from a variance engine, and validates it. Use this rather than assembling the frame by hand when writing an engine.

Usage

```
newVarianceSummary(source, term, varianceFraction, nFeatures, level = NULL)
```

Arguments

source	Character. Name of the engine producing the result.
term	Character vector of model terms.
varianceFraction	Numeric vector of variance shares, one per term.
nFeatures	Number of features the decomposition was computed over.
level	Optional character vector of factor levels per term.

Value

A validated data.frame with columns source, term, variance_fraction and n_features.

See Also

registerVarianceEngine for the full contract.

Examples

```
newVarianceSummary(
    source = "example",
    term = c("batch", "residual"),
    varianceFraction = c(0.3, 0.7),
    nFeatures = 100
)
```

`plotBasisProjection` *Plot samples projected onto a reference assay's basis*

Description

Draws the coordinates returned by `basisProjection`: every assay projected onto a single basis fitted on a reference assay, so panels placed side by side are directly comparable.

Usage

```
plotBasisProjection(
  object,
  assays = "raw",
  colourBy = "batch",
  reference = NULL
)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assays`	Character vector of assays to plot (default `"raw"`).
`colourBy`	Column name from `colData(object)` used to colour samples (default `"batch"`).
`reference`	Assay supplying the basis every panel is projected onto. Inferred from the correction lineage when `NULL`.

Details

All panels share one basis, fitted on `reference` and used to project every assay. Fitting a PCA separately per panel would make "PC1" a different direction in each, so panels carrying the same axis labels could not be compared: a sample would appear to move between panels because the axes moved, not because the sample did.

The axes are therefore the reference's principal components, not those of the assay in the panel, which is why this is named for the basis rather than for PCA. What a panel shows is where an assay's samples sit in the reference's structure.

This is a convenience wrapper. `basisProjection` returns the same coordinates as a data.frame for plotting by other means.

Value

A list of ggplot2 objects, one per assay.

See Also

basisProjection for the coordinates, basisRetention for how much variance stays on the basis.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)

plotBasisProjection(
    bv,
    assays = "raw",
    colourBy = "batch"
)
```

---

plotBatchEntropy *Plot batch mixing entropy*

Description

Computes local neighbourhood entropy based on k-nearest neighbours in PCA space to assess batch mixing.

Usage

```
plotBatchEntropy(object, assayName = "raw", batchVar = "batch", k = 10)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
assayName	Assay name
batchVar	Column in colData representing batch
k	Number of neighbours

Value

ggplot object

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
plotBatchEntropy(bv, assayName = "raw", batchVar = "batch")
bv <- runCorrection(bv, method = "combat", batch = "batch")
plotBatchEntropy(bv, assayName = "raw_combat", batchVar = "batch")
```

---

plotCorrelationChange *Plot correlation preservation after correction*

Description

Visualises the mean absolute change in sample–sample correlation structure across assays following batch correction.

Usage

```
plotCorrelationChange(evalResult)
```

Arguments

evalResult	Output from evaluateCorrections()

Value

ggplot object

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch, methods = "anova")
bv <- runCorrection(bv, method = "combat", batch = "batch")
bv <- profileVariance(bv, ~batch, assayName = "raw_combat", methods = "anova")
res <- evaluateCorrections(bv, baseline = "raw")
plotCorrelationChange(res)
```

`plotPCATrajectory` *Plot PCA correction trajectories*

Description

Plot PCA correction trajectories

Usage

```
plotPCATrajectory(
  object,
  assayBefore = "raw",
  assayAfter = "raw_combat",
  colourBy = "batch",
  reference = NULL
)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assayBefore`	baseline assay
`assayAfter`	corrected assay
`colourBy`	variable for colouring
`reference`	Assay whose PCA basis both assays are projected onto. Defaults to `assayBefore`.

Details

Both assays are projected onto a basis fitted once on `reference`, so the arrows measure movement relative to fixed directions and the 'before' positions are identical in every trajectory drawn from the same object. Fitting a PCA jointly over the pair being plotted would instead make those positions depend on which correction they were compared with, so two trajectories from one object could not be overlaid.

Value

ggplot object

See Also

`comparePCA` for the shared-basis convention and its trade-off.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch, assayName = "raw")
bv <- runCorrection(bv, method = "combat", batch = "batch")
bv <- profileVariance(bv, ~batch, assayName = "raw_combat")
plotPCATrajectory(bv, assayBefore = "raw", assayAfter = "raw_combat")
```

---

plotSampleDistance *Plot sample distance heatmap*

Description

Computes pairwise sample distances and visualises them as a heatmap.

Usage

```
plotSampleDistance(object, assayName = "raw")
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
assayName	Character string specifying assay name (default "raw")

Value

A ggplot object

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
plotSampleDistance(bv, assayName = "raw")
```

---

plotVarianceComposition *Plot variance composition across model terms*

Description

Visualise the proportion of expression variance attributed to each model term for a given assay. Variance fractions are obtained from the variance_history ledger recorded by profileVariance().

Usage

```
plotVarianceComposition(
  object,
  assays = NULL,
  method = "anova",
  formula = NULL
)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
assays	Character vector of assays to plot. Defaults to every assay profiled with method.
method	Variance engine used to compute the results (default "anova").
formula	Model formula to plot. Required when the chosen method has results for more than one formula, since a single chart can only show one decomposition per assay.

Details

The result is displayed as a stacked bar chart showing the relative contribution of batch effects, biological covariates, and residual variance.

This function retrieves variance attribution results from metadata(object)$variance_history. One bar is drawn per assay and the segments within it are the model terms, so each bar sums to 100 per cent. No baseline is required: this is an absolute view of where variance sits, whereas plotVarianceDelta shows the signed change in those fractions relative to a baseline assay.

Where an assay has been profiled more than once with the same method, the most recent record is used.

Stacked bars assume non-negative segments. The anova engine's shared term is negative under suppression, where model terms explain more jointly than separately; the bar then extends below zero and the positive segments overshoot 100 per cent. This is warned about, and the numbers should be read from varianceTable.

Value

A ggplot2 stacked bar chart representing the variance fractions attributed to each term in the variance model.

See Also

plotVarianceDelta for the change-vs-baseline view.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch, methods = "anova")
```

```
plotVarianceComposition(bv)
```

`plotVarianceDelta` *Plot variance change relative to baseline*

Description

Plot variance change relative to baseline

Usage

```
plotVarianceDelta(evalResult)
```

Arguments

`evalResult`	output from evaluateCorrections()

Value

ggplot object

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch, methods = "anova")
bv <- runCorrection(bv, method = "combat", batch = "batch")
bv <- profileVariance(bv, ~batch, assayName = "raw_combat", methods = "anova")
evalResult <- evaluateCorrections(bv)
plotVarianceDelta(evalResult)
```

`plotVarianceRadar` *Radar plot of variance redistribution*

Description

Visualises change in variance contribution (relative to baseline) across model terms using a polar (radar) representation.

Usage

```
plotVarianceRadar(object, assays = NULL, method = "anova", baseline = "raw")
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
assays	Character vector of assays (default: all)
method	Character vector of variance methods
baseline	Baseline assay

Value

ggplot object

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch, methods = "anova")
bv <- runCorrection(bv, method = "combat", batch = "batch")
bv <- profileVariance(bv, ~batch, assayName = "raw_combat", methods = "anova")
plotVarianceRadar(bv)
```

---

profileVariance *Profile sources of variance in expression data*

Description

Profile sources of variance in expression data

Usage

```
profileVariance(
  object,
  formula,
  methods = "anova",
  assayName = NULL,
  assays = NULL,
  ...,
  verbose = FALSE
)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
formula	Model formula specifying covariates

methods	Character vector of variance engines to apply. Defaults to "anova" alone; name several to profile with each.
assayName	Name of a single assay to profile. A convenience alternative to assays when only one assay is of interest; assays takes precedence if both are supplied.
assays	Character vector of assays to profile (default: all assays)
...	Additional arguments passed to variance engines
verbose	Show progress output from the variance engines (default FALSE). This governs only what the engines report about their progress. Diagnostics BatchVaria derives itself are warnings and are always raised, whatever verbose is set to.

Details

Features with zero or non-finite variance are excluded from every engine before profiling, and the number excluded is reported once per assay. Such features are common in unfiltered count data and carry no information for a variance decomposition; retaining them causes engine-specific failures rather than engine-specific answers.

Each assay/method combination is attempted independently. If one engine fails, a warning names the assay, the method and the reason, and profiling continues with the remaining combinations. An error is raised only when every attempt fails, so a partial run always returns the results that succeeded rather than discarding them.

Engines differ in the formulas they accept, and each builds its own design matrix. variancePartition accepts random-effects notation such as ~ (1 | batch); anova models fixed effects only and declines such formulas by name. Supplying a random-effects formula to both engines is therefore expected to profile with variancePartition and report why anova did not participate.

Only anova runs by default. Profiling with every registered engine would leave the ledger holding several decompositions after a single call, and varianceTable and its relatives summarise exactly one - so the shortest path through the package would end in an error asking which was meant. It would also make registerVarianceEngine change the behaviour of calls that never named the new engine. Comparing engines is worth doing, but it is a deliberate act: name them.

The anova engine decomposes variance per model term using Type II sums of squares, emitting one row per term plus residual and shared. Type II sums of squares are order independent but do not add up to the model sum of squares under an unbalanced design: variance that the design cannot attribute to any single term is reported as shared rather than left out, so the fractions remain a partition. Under an orthogonal design shared is zero; under complete confounding of two terms it absorbs everything they explain, since neither is identifiable given the other. It can be negative where terms explain more jointly than separately, which is reported with a warning.

Fractions are averaged across features by default, matching how variancePartition summarises, so the two engines describe the typical feature and stay comparable. Pass weighting = "pooled" to weight each feature by the variance it carries instead. The choice and the sums-of-squares type are recorded alongside each result.

Value

Updated SummarizedExperiment

See Also

assayVariance, varianceHistory

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch)
length(varianceHistory(bv))

## Comparing engines is opt-in
bv <- profileVariance(bv, ~batch, methods = c("anova", "variancePartition"))
unique(vapply(varianceHistory(bv), function(x) x$method, character(1)))
```

---

provenance *Report the analysis history recorded on an object*

Description

Prints what has been done to an object: its dimensions and assays, the corrections that produced each assay, and the variance decompositions recorded against them. Returns the two tables invisibly, as data.frames.

Usage

```
provenance(object)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.

Details

This is the readable view of the two ledgers BatchVaria writes into metadata(). varianceHistory returns the variance ledger unprocessed; the correction ledger is metadata(object)$correction_history.

The correction table reports preserve and no_op for each step. preserve is what distinguishes two runs of one method on one assay, and no_op marks a correction that returned an assay indistinguishable from its input - a condition that is easy to miss on the console and would otherwise be invisible once the session that produced it has ended.

Value

Invisibly, a list with elements assays (character), corrections and variance_profiles (data.frames, or NULL where the ledger is empty).

See Also

varianceHistory, varianceResults

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100, nSamples = 8)
bv <- runCorrection(bv, method = "combat", batch = "batch")
bv <- profileVariance(bv, ~batch, assays = c("raw", "raw_combat"))
provenance(bv)
```

---

recordVariance *Record variance decomposition results*

Description

Stores the output of a variance decomposition method in the internal variance ledger held in the object's `metadata()`. Each entry captures the assay, model formula, method used, and resulting variance summary.

Usage

```
recordVariance(
  object,
  assayName,
  formula,
  result,
  method = "variancePartition"
)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assayName`	Character. Name of the assay associated with the result
`formula`	Model formula used for variance decomposition
`result`	Data frame containing variance decomposition results. Must include at least `source`, `term`, and `variance_fraction` columns. Results built by `newVarianceSummary`, and so every result from a registered engine, also carry `n_features`; a summary assembled by hand need not, and that column is then `NA` for the entry in `varianceResults` output. The `NA` records absent metadata rather than a failure, and nothing downstream reads it.
`method`	Character. Name of the variance decomposition method

Details

Results are appended to `metadata(object)$variance_history` as a list of entries. Each entry contains:

- `assay` – assay name
- `formula` – model formula
- `method` – variance decomposition method
- `result` – variance summary data frame

Value

Updated `SummarizedExperiment` with appended variance record

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
res <- data.frame(
    source = "anova",
    term = c("model", "residual"),
    variance_fraction = c(0.3, 0.7)
)
bv <- recordVariance(
    bv,
    assayName = "raw",
    formula = ~batch,
    result = res,
    method = "anova"
)
length(varianceHistory(bv))
```

`registerCorrectionMethod` *Register a batch correction method*

Description

Adds a correction method to the set that `runCorrection` can dispatch to, so that the methods are extensible rather than fixed.

Usage

```
registerCorrectionMethod(name, method, overwrite = FALSE)
```

Arguments

name	Character. Name callers will pass as method.
method	A function implementing the correction contract below.
overwrite	Logical. Replace a method of the same name.

Details

A correction method is a function with the signature

```
function(assayMatrix, batch, sampleData, preserve = NULL, ...)
```

where assayMatrix is features by samples, batch is the name of a single sampleData column holding the batch structure, sampleData is the colData of the object, and preserve is an optional character vector of sampleData column names. The ... is required: runCorrection() forwards method-specific arguments through it, so each method must tolerate arguments intended for another.

preserve is required in the signature rather than optional, which is where this contract diverges from registerVarianceEngine. A method declaring only ... would absorb it silently and return a plausible matrix from a correction that protected nothing. The variance engines have no argument whose silent absorption changes the answer in that way.

Methods build their own design matrices from preserve rather than receiving one, so a design assembled for one method is never forced through another.

Direction of preserve. preserve names variables whose variation should be *kept*. A method protects them, typically by estimating the batch effect adjusted for them. It maps to mod in sva::ComBat() and to design in limma::removeBatchEffect(), which its own documentation describes as the conditions "to be preserved".

Note that removeBatchEffect() *also* has an argument spelled covariates, and it means the opposite: numeric covariates to be adjusted for, that is, removed. An adapter that wires preserve to covariates will strip exactly the signal it was asked to protect, and the result will still look like a successful correction - batch variance falls and the table reads as expected. Check the direction.

Batch. batch is a single column name, validated by runCorrection. Methods accepting more than one batch variable must expose the others through ....

Progress output. A method need not gate its own progress reporting. runCorrection suppresses messages raised while a method runs unless called with verbose = TRUE, and it does so at the point it invokes the method, so registered methods inherit the policy. Report progress with message() rather than cat() to participate in it. Anything a method has found out about the data, as opposed to reporting about its own progress, should be a warning: those are never suppressed.

Return value. A numeric matrix whose dim and dimnames are identical() to those of assayMatrix. This makes sample order part of the contract rather than a convention: delta tables, the PCA basis and correlation structure are all keyed on sample identity, so a reordered return would misalign every downstream comparison without raising an error. The check is applied to whatever a method returns, because a conforming signature does not guarantee a conforming result.

Methods take a matrix and return a matrix. The container is never passed in, and recording the correction in the ledger is runCorrection's responsibility, not the method's.

Value

The registered name, invisibly.

See Also

availableCorrectionMethods, unregisterCorrectionMethod, runCorrection

Examples

```
## a method that centres each batch on the overall feature mean
centreByBatch <- function(assayMatrix, batch, sampleData,
                          preserve = NULL, ...) {
    b <- factor(sampleData[[batch]])
    out <- assayMatrix
    for (lv in levels(b)) {
        idx <- which(b == lv)
        out[, idx] <- out[, idx] - rowMeans(out[, idx, drop = FALSE])
    }
    out + rowMeans(assayMatrix)
}

registerCorrectionMethod("batchCentre", centreByBatch)
availableCorrectionMethods()

set.seed(1)
bv <- exampleBatchVaria(nGenes = 50)
bv <- runCorrection(bv, method = "batchCentre", batch = "batch")
SummarizedExperiment::assayNames(bv)

unregisterCorrectionMethod("batchCentre")
```

---

registerVarianceEngine

Register a variance profiling engine

Description

Adds a variance decomposition engine to the set that profileVariance can dispatch to, so that the engines are extensible rather than fixed.

Usage

```
registerVarianceEngine(name, engine, overwrite = FALSE)
```

Arguments

name	Character. Name callers will pass as methods.
engine	A function implementing the engine contract below.
overwrite	Logical. Replace an engine of the same name.

Details

An engine is a function with the signature

```
function(assayMatrix, formula, sampleData, ...)
```

where `assayMatrix` is features by samples with zero-variance features already excluded, `formula` is the model, and `sampleData` is the `colData` of the object. The `...` is required: `profileVariance()` forwards engine-specific arguments through it, and an engine must tolerate arguments meant for another.

The return value must be a data.frame with one row per model term and the columns

- `source` - the engine's own name
- `term` - the model term the variance is attributed to
- `variance_fraction` - that term's share of total variance
- `n_features` - features the decomposition was computed over

Extra columns are allowed and are carried through to `varianceResults`, where they are `NA` for engines that do not supply them; the built-in `anova` engine uses this to record its sums-of-squares type and weighting. `variance_fraction` must be non-negative except for the reserved term `"shared"`, which holds variance an unbalanced design cannot attribute to any single term and may be negative under suppression.

Two term names are reserved, because they mean the same thing whatever the engine: `"residual"` for variance no model term accounts for, and `"shared"` as above. An engine whose underlying implementation spells these differently must normalise before returning - the built-in `variancePartition` engine renames `"Residuals"` on the way out. Harmonising the columns but not the vocabulary would produce results that look comparable and cannot be joined. Every other term an engine may name as its model does.

Use `newVarianceSummary` to build a conforming result rather than assembling the data.frame by hand.

Progress output. An engine need not gate its own progress reporting. `profileVariance` suppresses messages raised while an engine runs unless called with `verbose = TRUE`, and it does so at the point it invokes the engine, so registered engines inherit the policy. Report progress with `message()` rather than `cat()` to participate in it. Anything an engine has found out about the data, as opposed to reporting about its own progress, should be a warning: those are never suppressed.

Value

The registered name, invisibly.

See Also

`availableVarianceMethods`, `unregisterVarianceEngine`, `newVarianceSummary`

Examples

```
## an engine that attributes everything to the first model term
flatEngine <- function(assayMatrix, formula, sampleData, ...) {
    newVarianceSummary(
        source = "flat",
        term = c("everything", "residual"),
        varianceFraction = c(0.4, 0.6),
        nFeatures = nrow(assayMatrix)
    )
}

registerVarianceEngine("flat", flatEngine)
availableVarianceMethods()

set.seed(1)
bv <- exampleBatchVaria(nGenes = 50)
bv <- profileVariance(bv, ~batch, assays = "raw", methods = "flat")
varianceResults(bv, method = "flat")

unregisterVarianceEngine("flat")
```

---

runCorrection *Apply batch correction to expression data*

Description

Runs a specified batch correction method on an assay and stores the corrected matrix as a new assay on the object.

Usage

```
runCorrection(
  object,
  method = "combat",
  batch,
  preserve = NULL,
  assayName = "raw",
  newAssayName = NULL,
  ...,
  verbose = FALSE
)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
method	Character string specifying the correction method

batch	Column name in colData(object) defining batch structure.
preserve	Optional character vector of colData column names whose variation should be *kept*. The batch effect is estimated adjusted for these, so biological signal they carry is protected rather than removed.
assayName	Name of the input assay (default "raw").
newAssayName	Optional corrected assay name.
...	Additional arguments passed to the correction engine.
verbose	Show progress output from the correction engine (default FALSE). This governs only what the engine reports about its progress. Diagnostics BatchVaria derives itself are warnings and are always raised, whatever verbose is set to.

Details

Two methods are registered on load. "combat" calls sva::ComBat(), which applies empirical Bayes shrinkage to each batch's location *and* scale. "limma" calls limma::removeBatchEffect(), which removes a per-feature location shift only. The two therefore differ in magnitude even where they agree in direction, and neither should be expected to reproduce the other.

Both expect log-scale data. Neither is appropriate on raw counts, and removeBatchEffect() in particular is intended for exploratory work

- visualisation, clustering, distance calculations - rather than as input to differential expression testing, where the batch term belongs in the model instead. Correcting first and testing afterwards understates the residual variance and overstates significance.

removeBatchEffect() fits batch and the preserve design simultaneously, so a design in which the two are close to aliased is rank deficient and the batch effect is not separately estimable. That is a property of the design rather than of the method.

Further methods can be added with registerCorrectionMethod.

Value

Updated SummarizedExperiment with corrected assay added.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- runCorrection(bv, method = "combat", batch = "batch")
SummarizedExperiment::assayNames(bv)
```

unregisterCorrectionMethod

Remove a registered batch correction method

Description

Remove a registered batch correction method

Usage

```
unregisterCorrectionMethod(name)
```

Arguments

name	Character. Name of the method to remove.

Value

The removed name, invisibly.

See Also

registerCorrectionMethod

Examples

```
dummy <- function(assayMatrix, batch, sampleData, preserve = NULL, ...) {
    assayMatrix
}
registerCorrectionMethod("dummy", dummy)
unregisterCorrectionMethod("dummy")
availableCorrectionMethods()
```

unregisterVarianceEngine

Remove a registered variance profiling engine

Description

Remove a registered variance profiling engine

Usage

```
unregisterVarianceEngine(name)
```

Arguments

name	Character. Name of the engine to remove.

Value

The removed name, invisibly.

See Also

registerVarianceEngine

Examples

```
dummy <- function(assayMatrix, formula, sampleData, ...) {
    newVarianceSummary(
        source = "dummy", term = "all",
        varianceFraction = 1, nFeatures = nrow(assayMatrix)
    )
}
registerVarianceEngine("dummy", dummy)
unregisterVarianceEngine("dummy")
availableVarianceMethods()
```

---

varianceChange *Summarise variance change vs raw for a model term*

Description

Summarise variance change vs raw for a model term

Usage

```
varianceChange(
  object,
  term,
  assays = NULL,
  method = NULL,
  formula = NULL,
  baseline = NULL
)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
term	Model term to evaluate (e.g. "batch")

assays	Assays to include
method	Variance method to summarise. Required when the ledger holds results for more than one method.
formula	Model formula to summarise. Required when the chosen method has results for more than one formula.
baseline	Assay that changes are computed against. Inferred from the correction ledger when NULL.

Value

data.frame of variance change for a single model term versus the baseline assay

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch)
varianceChange(bv, term = "residual")
```

---

varianceDelta *Compute variance redistribution relative to baseline*

Description

Calculates the change in variance explained by each model term for one or more assays relative to a baseline assay.

Usage

```
varianceDelta(
  object,
  assays = NULL,
  method = "anova",
  formula = NULL,
  baseline = NULL
)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
assays	Character vector of assay names (default: all assays)
method	Variance estimation method
formula	Model formula to compare under. Required when the chosen method has results for more than one formula, since a delta between decompositions of different models is not interpretable.

baseline — Baseline assay name. When NULL (the default) it is inferred from the correction ledger, falling back to "raw".

Value

data.frame with columns: assay, method, term, delta

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch, methods = "anova")
bv <- runCorrection(bv, method = "combat", batch = "batch")
bv <- profileVariance(bv, ~batch, assayName = "raw_combat", methods = "anova")
varianceDelta(bv)
```

---

varianceHistory *Access variance profiling history*

Description

Access variance profiling history

Usage

```
varianceHistory(object)
```

Arguments

object — A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.

Value

The variance ledger, metadata(object)$variance_history, unprocessed: a list with one entry per recorded decomposition, or NULL if nothing has been profiled. Entries the package writes carry assay, method, formula, formula_key, result, fingerprint and timestamp; only assay, method and result are required of an entry (see BatchVaria-requirements). Use varianceResults for the results as one data.frame.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch)
length(varianceHistory(bv))
```

varianceResults *Retrieve variance decomposition results*

Description

Extracts variance decomposition results stored in the object's `metadata()`.

Usage

```
varianceResults(object, assayName = NULL, method = NULL)
```

Arguments

`object`	A `SummarizedExperiment` meeting BatchVaria's requirements. See BatchVaria-requirements.
`assayName`	Optional assay name to filter results.
`method`	Optional method name to filter results.

Value

A data.frame containing variance decomposition results in long format.

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch)
varianceResults(bv, assayName = "raw")
```

varianceTable *Create variance summary tables*

Description

Generates a variance percentage table and a delta-versus-baseline table from the variance ledger, for a single method and model formula.

Usage

```
varianceTable(
  object,
  assays = NULL,
  method = NULL,
  formula = NULL,
  baseline = NULL,
  formatDelta = FALSE,
  digits = 2
)
```

Arguments

object	A SummarizedExperiment meeting BatchVaria's requirements. See BatchVaria-requirements.
assays	Character vector of assay names to include
method	Variance method to summarise. Required when the ledger holds results for more than one method.
formula	Model formula to summarise. Required when the chosen method has results for more than one formula.
baseline	Assay that deltas are computed against. When NULL (the default) it is inferred from the correction ledger – the assay the corrected assays were derived from – falling back to "raw" if present. If no baseline can be determined the percent table is still returned and delta is NULL.
formatDelta	Logical; if TRUE, format delta with signs
digits	Number of digits for rounding

Details

A variance table describes one decomposition. Results from different engines are not commensurable - pca reports variance along principal axes while anova and variancePartition report variance attributable to model terms - so stacking them in a single table produces columns that sum to well over 100\ assay profiled under two formulas yields two different decompositions.

The anova engine contributes a shared row holding variance that an unbalanced design cannot attribute to any single term. It is zero under an orthogonal design and positive under confounding. It is negative where terms explain more jointly than separately (a suppression effect), which shows as a negative percentage; the column still sums to 100 because the other terms are correspondingly larger.

method and formula therefore select exactly one decomposition. Either may be omitted when the ledger is unambiguous; when it is not, the error lists what is available rather than choosing silently. Where an assay has been profiled repeatedly under the same key, the most recent result is used.

Value

list with percent, delta and the baseline the delta was computed against. delta is NULL when no baseline could be determined.

See Also

assayVariance for the total variance that these percentages are expressed relative to

Examples

```
set.seed(1)
bv <- exampleBatchVaria(nGenes = 100)
bv <- profileVariance(bv, ~batch)
varianceTable(bv)
```

Index

assayVariance, 3, *23*, *37*
availableCorrectionMethods, 4, *27*
availableVarianceMethods, 4, *28*

basisProjection, 5, *14, 15*
basisRetention, *5*, 6, *6*, *12*, *15*
BatchVaria, *8*
BatchVaria-requirements, *3*, *5*, 7, *7*, *9–12*, *14, 15*, *17–19*, *21*, *23, 24*, *29*, *32–36*

compareCorrelations, 9
comparePCA, *5–7*, 10, *17*

evaluateCorrections, 11
exampleBatchVaria, *8*, 12

newVarianceSummary, 13, *24*, *28*

plotBasisProjection, *6*, 14
plotBatchEntropy, 15
plotCorrelationChange, 16
plotPCATrajectory, 17
plotSampleDistance, 18
plotVarianceComposition, 18
plotVarianceDelta, *19*, 20
plotVarianceRadar, 20
profileVariance, *3*, *8*, *13*, 21, *27, 28*
provenance, *8*, 23

recordVariance, 24
registerCorrectionMethod, *4*, 25, *30, 31*
registerVarianceEngine, *5*, *13*, *22*, *26*, 27, *32*
rnorm, *12*
runCorrection, *8*, *25–27*, 29

set.seed, *13*

unregisterCorrectionMethod, *27*, 31
unregisterVarianceEngine, *28*, 31

varianceChange, 32
varianceDelta, 33
varianceHistory, *23, 24*, 34
varianceResults, *3*, *24*, *28*, *34*, 35
varianceTable, *19*, *22*, 35