Package 'bakR'

October 12, 2022

Title Analyze and Compare Nucleotide Recoding RNA Sequencing Datasets

Version 0.2.4

Description Several implementations of a novel Bayesian hierarchical statistical model of nucleotide recoding RNA-seq experiments (NR-seq; TimeLapse-seq, SLAM-seq, TUC-seq, etc.) for analyzing and comparing NR-seq datasets (see 'Vock and Simon' (2022) <doi:10.1101/2022.09.02.505697>). NR-seq is a powerful extension of RNA-seq that provides information about the kinetics of RNA metabolism (e.g., RNA degradation rate constants), which is notably lacking in standard RNA-seq data. The statistical model makes maximal use of these high-throughput datasets by sharing information across transcripts to significantly improve uncertainty quantification and increase statistical power. 'bakR' includes a maximally efficient implementation of this model for conservative initial investigations of datasets. 'bakR' also provides more highly powered implementations using the probabilistic programming language 'Stan' to sample from the full posterior distribution. 'bakR' performs multiple-test adjusted statistical inference with the output of these model implementations to help biologists separate signal from background. Methods to automatically visualize key results and detect batch effects are also provided.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Biarch true

Depends R (>= 3.5.0)

Imports purrr, methods, Rcpp (>= 0.12.0), RcppParallel (>= 5.0.1), rstan, rstantools (>= 2.1.1), dplyr, stats, magrittr, Hmisc, ggplot2, data.table

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), RcppParallel (>= 5.0.1), rstan (>= 2.18.1), StanHeaders (>= 2.18.0)

SystemRequirements GNU make

Suggests rmarkdown, knitr, DESeq2, pheatmap

VignetteBuilder knitr

URL https://simonlabcode.github.io/bakR/

BugReports https://github.com/simonlabcode/bakR/issues/

NeedsCompilation yes

Author Isaac Vock [aut, cre] (<https://orcid.org/0000-0002-7178-6886>)

Maintainer Isaac Vock <isaac.vock@gmail.com>

Repository CRAN

Date/Publication 2022-10-10 16:10:02 UTC

R topics documented:

---

bakR-package *The 'bakR' package.*

Description

A DESCRIPTION OF THE PACKAGE

References

Stan Development Team (2020). RStan: the R interface to Stan. R package version 2.21.2. https://mc-stan.org

bakRData *bakR Data object helper function for users*

Description

This function creates an object of class bakRData

Usage

```
bakRData(cB, metadf)
```

Arguments

cB	Dataframe with columns corresponding to feature ID, number of Ts, number of mutations, sample ID, and number of identical observations
metadf	Dataframe detailing s4U label time and experimental ID of each sample

Value

A bakRData object. This has two components: a data frame describing experimental details (metadf) and a data frame containing the NR-seq data (cB).

Examples

```
# Load cB
data("cB_small")

# Load metadf
data("metadf")

# Create bakRData object
bakRData <- bakRData(cB_small, metadf)
```

bakRFit *Estimating kinetic parameters from nucleotide recoding RNA-seq data*

Description

bakRFit analyzes nucleotide recoding RNA-seq data to estimate kinetic parameters relating to RNA stability and changes in RNA stability induced by experimental perturbations. Several statistical models of varying efficiency and accuracy can be used to fit data.

Usage

```
bakRFit(
  obj,
  StanFit = FALSE,
  HybridFit = FALSE,
  high_p = 0.2,
  totcut = 50,
  Ucut = 0.25,
  AvgU = 4,
  FastRerun = FALSE,
  FOI = c(),
  concat = TRUE,
  StanRateEst = FALSE,
  RateEst_size = 25,
  low_reads = 1000,
  high_reads = 5000,
  chains = 1,
  NSS = FALSE,
  Chase = FALSE,
  BDA_model = FALSE,
  ...
)
```

Arguments

obj	bakRData object produced by bakRData or a bakRFit object produced by bakRFit
StanFit	Logical; if TRUE, then the MCMC implementation is run. Will only be used if obj is a bakRFit object
HybridFit	Logical; if TRUE, then the Hybrid implementation is run. Will only be used if obj is a bakRFit object
high_p	Numeric; Any transcripts with a mutation rate (number of mutations / number of Ts in reads) higher than this in any -s4U control samples (i.e., samples that were not treated with s4U) are filtered out
totcut	Numeric; Any transcripts with less than this number of sequencing reads in any sample are filtered out
Ucut	Numeric; All transcripts must have a fraction of reads with 2 or less Us less than this cutoff in all samples
AvgU	Numeric; All transcripts must have an average number of Us greater than this cutoff in all samples
FastRerun	Logical; only matters if a bakRFit object is passed to bakRFit. If TRUE, then the Stan-free model implemented in fast_analysis is rerun on data, foregoing fitting of either of the 'Stan' models.
FOI	Features of interest; character vector containing names of features to analyze
concat	Logical; If TRUE, FOI is concatenated with output of reliableFeatures
StanRateEst	Logical; if TRUE, a simple 'Stan' model is used to estimate mutation rates for fast_analysis; this may add a couple minutes to the runtime of the analysis.

RateEst_size	Numeric; if StanRateEst is TRUE, then data from RateEst_size genes are used for mutation rate estimation. This can be as low as 1 and should be kept low to ensure maximum efficiency
low_reads	Numeric; if StanRateEst is TRUE, then only features with more than low_reads reads in all samples will be used for mutation rate estimation
high_reads	Numeric; if StanRateEst is TRUE, then only features with less than high_read reads in all samples will be used for mutation rate estimation. A high read count cutoff is as important as a low read count cutoff in this case because you don't want the fraction labeled of chosen features to be extreme (e.g., close to 0 or 1), and high read count features are likely low fraction new features.
chains	Number of Markov chains to sample from. 1 should suffice since these are validated models. Running more chains is generally preferable, but memory constraints can make this unfeasible.
NSS	Logical; if TRUE, logit(fn)s are directly compared to avoid assuming steady-state
Chase	Logical; Set to TRUE if analyzing a pulse-chase experiment. If TRUE, kdeg = -ln(fn)/tl where fn is the fraction of reads that are s4U (more properly referred to as the fraction old in the context of a pulse-chase experiment).
BDA_model	Logical; if TRUE, variance is regularized with scaled inverse chi-squared model. Otherwise a log-normal model is used.
...	Arguments passed to either fast_analysis (if a bakRData object) or TL_Stan and Hybrid_fit (if a bakRFit object)

Details

If bakRFit is run on a bakRData object, cBprocess and then fast_analysis will always be called. The former will generate the processed data that can be passed to the model fitting functions (fast_analysis and TL_Stan). The call to fast_analysis will generate a list of dataframes containing information regarding the fast_analysis fit. fast_analysis is always called because its output is required for both Hybrid_fit and TL_Stan.

If bakRFit is run on a bakRFit object, cBprocess will not be called again, as the output of cBprocess will already be contained in the bakRFit object. Similarly, fast_analysis will not be called again unless bakRFit is rerun on the bakRData object. or if FastRerun is set to TRUE. If you want to generate model fits using different parameters for cBprocess, you will have to rerun bakRFit on the bakRData object.

See the documentation for the individual fitting functions for details regarding how they analyze nucleotide recoding data. What follows is a brief overview of how each works

fast_analysis (referred to as the MLE implementation in the bakR paper) either estimates mutation rates from + and (if available) - s4U samples or uses mutation rate estimates provided by the user to perform maximum likelihood estimation (MLE) of the fraction of RNA that is labeled for each replicate of nucleotide recoding data provided. Uncertainties for each replicate's estimate are approximated using asymptotic results involving the Fisher Information and assuming known mutation rates. Replicate data is pooled using an approximation to hierarchical modeling that relies on analytic solutions to simple Bayesian models. Linear regression is used to estimate the relationship between read depths and replicate variability for uncertainty estimation regularization, again performed using analytic solutions to Bayesian models.

TL_Stan with Hybrid_Fit set to TRUE (referred to as the Hybrid implementation in the bakR paper) takes as input estimates of the logit(fraction new) and uncertainty provided by fast_analysis. It then uses 'Stan' on the backend to implement a hierarchical model that pools data across replicates and the dataset to estimate effect sizes (L2FC(kdeg)) and uncertainties. Replicate variability information is pooled across each experimental condition to regularize variance estimates using a hierarchical linear regression model.

The default behavior of TL_Stan (referred to as the MCMC implementation in the bakR paper) is to use 'Stan' on the back end to implement a U-content exposure adjusted Poisson mixture model to estimate fraction news from the mutational data. Partial pooling of replicate variability estimates is performed as with the Hybrid implementation.

Value

bakRFit object with results from statistical modeling and data processing. Objects possibly included are:

- Fast_Fit; Always will be present. Output of fast_analysis
- Hybrid_Fit; Only present if HybridFit = TRUE. Output of TL_stan
- Stan_Fit; Only present if StanFit = TRUE. Output of TL_stan
- Data_lists; Always will be present. Output of cBprocess with Fast and Stan == TRUE

Examples

```
# Simulate data for 1000 genes, 2 replicates, 2 conditions
simdata <- Simulate_bakRData(1000, nreps = 2)

# You always must fit fast implementation before any others
Fit <- bakRFit(simdata$bakRData)
```

---

cBprocess *Curate data in bakRData object for statistical modeling*

Description

cBprocess creates the data structures necessary to analyze nucleotide recoding RNA-seq data with any of the statistical model implementations in bakRFit. The input to cBprocess must be an object of class bakRData. The output can contain data passable to fast_analysis (if Fast == TRUE), TL_stan with StanFit = TRUE if Stan == TRUE, or both.

Usage

```
cBprocess(
  obj,
  high_p = 0.2,
  totcut = 50,
  Ucut = 0.25,
  AvgU = 4,
  Stan = TRUE,
  Fast = TRUE,
  FOI = c(),
  concat = TRUE
)
```

Arguments

obj	An object of class bakRData
high_p	Numeric; Any transcripts with a mutation rate (number of mutations / number of Ts in reads) higher than this in any no s4U control samples are filtered out
totcut	Numeric; Any transcripts with less than this number of sequencing reads in any sample are filtered out
Ucut	Numeric; All transcripts must have a fraction of reads with 2 or less Us less than this cutoff in all samples
AvgU	Numeric; All transcripts must have an average number of Us greater than this cutoff in all samples
Stan	Boolean; if TRUE, then data_list that can be passed to 'Stan' is curated
Fast	Boolean; if TRUE, then dataframe that can be passed to fast_analysis() is curated
FOI	Features of interest; character vector containing names of features to analyze
concat	Boolean; If TRUE, FOI is concatenated with output of reliableFeatures

Details

The 1st step executed by `cBprocess` is to find the names of features which are deemed "reliable". A reliable feature is one with sufficient read coverage in every single sample (i.e., > totcut reads in all samples) and limited mutation content in all -s4U control samples (i.e., < high_p mutation rate in all samples lacking s4U feeds). This is done with a call to `reliableFeatures`. The 2nd step is to extract only reliableFeatures from the cB dataframe in the `bakRData` object. During this process, a numerical ID is given to each reliableFeature, with the numerical ID corresponding to the order in which each feature is found in the original cB (this might typically be alphabetical order).

The 3rd step is to prepare a dataframe where each row corresponds to a set of n identical reads (that is they come from the same sample and have the same number of mutations and Us). Part of this process involves assigning an arbitrary numerical ID to each replicate in each experimental condition. The numerical ID will correspond to the order the sample appears in metadf. The outcome of this step is multiple dataframes with variable information content. These include a dataframe with information about read counts in each sample, one which logs the U-contents of each feature, one which is compatible with `fast_analysis` and thus groups reads by their number of mutations as well as their number of Us, and one which is compatible with `TL_stan` with StanFit

== TRUE and thus groups ready by only their number of mutations. At the end of this step, two other smaller data structures are created, one which is an average count matrix (a count matrix where the ith row and jth column corresponds to the average number of reads mappin to feature i in experimental condition j, averaged over all replicates) and the other which is a sample lookup table that relates the numerical experimental and replicate IDs to the original sample name.

If FOI is non-null and concat == TRUE, the features listed in FOI will be included in the list of reliable features that make it past filtering. If FOI is non-null and concat == FALSE, the features listed in FOI will be the only reliable features that make it past filtering. If FOI is null and concat == FALSE, an error will be thrown.

Value

returns list of objects that can be passed to `TL_stan` and/or `fast_analysis`. Those objects are:

- Stan_data; list that can be passed to `TL_stan` with Hybrid_Fit = FALSE. Consists of metadata as well as data that 'Stan' will analyze. Data to be analyzed consists of equal length vectors. The contents of Stan_data are:
  - NE; Number of datapoints for 'Stan' to analyze (NE = Number of Elements)
  - NF; Number of features in dataset
  - TP; Numerical indicator of s4U feed (0 = no s4U feed, 1 = s4U fed)
  - FE; Numerical indicator of feature
  - num_mut; Number of U-to-C mutations observed in a particular set of reads
  - MT; Numerical indicator of experimental condition (Exp_ID from metadf)
  - nMT; Number of experimental conditions
  - R; Numerical indicator of replicate
  - nrep; Number of replicates (analysis requires same number of replicates of all conditions)
  - num_obs; Number of reads with identical data (number of mutations, feature of origin, and sample of origin)
  - tl; Vector of label times for each experimental condition
  - U_cont; Log2-fold-difference in U-content for a feature in a sample relative to average U-content for that sample
  - Avg_Reads; Standardized log10(average read counts) for a particular feature in a particular condition, averaged over replicates
  - Avg_Reads_natural; Unstandardized average read counts for a particular feature in a particular condition, averaged over replicates. Used for `plotMA`
  - sdf; Dataframe that maps numerical feature ID to original feature name. Also has read depth information
  - sample_lookup; Lookup table relating MT and R to the original sample name
- Fast_df; A data frame that can be passed to `fast_analysis`. The contents of Fast_df are:
  - sample; Original sample name
  - XF; Original feature name
  - TC; Number of T to C mutations
  - nT; Number of Ts in read
  - n; Number of identical observations
  - fnum; Numerical indicator of feature

  - type; Numerical indicator of s4U feed (0 = no s4U feed, 1 = s4U fed)
  - mut; Numerical indicator of experimental condition (Exp_ID from metadf)
  - reps; Numerical indicator of replicate
- Count_Matrix; A matrix with read count information. Each column represents a sample and each row represents a feature. Each entry is the raw number of read counts mapping to a particular feature in a particular sample. Column names are the corresponding sample names and row names are the corresponding feature names.

Examples

```
# Load cB
data("cB_small")

# Load metadf
data("metadf")

# Create bakRData
bakRData <- bakRData(cB_small, metadf)

# Preprocess data
data_for_bakR <- cBprocess(obj = bakRData)
```

`cB_small` *Example cB data frame*

Description

Subset of a cB file from the DCP2 dataset published in Luo et al. 2020. The original file is large (69 MB), so the example cB file has been downsampled and contains only 10 genes (rather than 25012). The columns are described in the Getting_Started vignette.

Usage

```
data(cB_small)
```

Format

A dataframe with 5788 rows and 5 variables; each row corresponds to a group of sequencing reads

sample Sample name

TC Number of T-to-C mutations

nT Number of Ts

XF Name of feature to which the group of reads map; usually a gene name

n Number of identical sequencing reads

References

Luo et al. (2020) Biochemistry. 59(42), 4121-4142

Examples

```
data(cB_small)
data(metadf)
bakRdat <- bakRData(cB_small, metadf)
```

`fast_analysis` *Efficiently analyze nucleotide recoding data*

Description

`fast_analysis` analyzes nucleotide recoding data maximum likelihood estimation with the L-BFGS-B algorithm implemented by `stats::optim` combined with analytic solutations to simple Bayesian models to perform approximate partial pooling. Output includes kinetic parameter estimates in each replicate, kinetic parameter estimates averaged across replicates, and log-2 fold changes in the degradation rate constant (L2FC(kdeg)). Averaging takes into account uncertainties estimated using the Fisher Information and estimates are regularized using analytic solutions of fully Bayesian models. The result is that kdegs are shrunk towards population means and that uncertainties are shrunk towards a mean-variance trend estimated as part of the analysis.

Usage

```
fast_analysis(
  df,
  pnew = NULL,
  pold = NULL,
  no_ctl = FALSE,
  read_cut = 50,
  features_cut = 50,
  nbin = NULL,
  prior_weight = 2,
  MLE = TRUE,
  lower = -7,
  upper = 7,
  se_max = 2.5,
  mut_reg = 0.1,
  p_mean = 0,
  p_sd = 1,
  StanRate = FALSE,
  Stan_data = NULL,
  null_cutoff = 0,
  NSS = FALSE,
  Chase = FALSE,
  BDA_model = FALSE
)
```

Arguments

df	Dataframe in form provided by cB_to_Fast
pnew	Labeled read mutation rate; default of 0 means that model estimates rate from s4U fed data. If pnew is provided by user, must be a vector of length == number of s4U fed samples. The 1st element corresponds to the s4U induced mutation rate estimate for the 1st replicate of the 1st experimental condition; the 2nd element corresponds to the s4U induced mutation rate estimate for the 2nd replicate of the 1st experimental condition, etc.
pold	Unlabeled read mutation rate; default of 0 means that model estimates rate from no-s4U fed data
no_ctl	Logical; if TRUE, then -s4U control is not used for background mutation rate estimation
read_cut	Minimum number of reads for a given feature-sample combo to be used for mut rate estimates
features_cut	Number of features to estimate sample specific mutation rate with
nbin	Number of bins for mean-variance relationship estimation. If NULL, max of 10 or (number of logit(fn) estimates)/100 is used
prior_weight	Determines extent to which logit(fn) variance is regularized to the mean-variance regression line
MLE	Logical; if TRUE then replicate logit(fn) is estimated using maximum likelihood; if FALSE more conservative Bayesian hypothesis testing is used
lower	Lower bound for MLE with L-BFGS-B algorithm
upper	Upper bound for MLE with L-BFGS-B algorithm
se_max	Uncertainty given to those transcripts with estimates at the upper or lower bound sets. This prevents downstream errors due to abnormally high standard errors due to transcripts with extreme kinetics
mut_reg	If MLE has instabilities, empirical mut rate will be used to estimate fn, multiplying pnew by 1+mut_reg and pold by 1-mut_reg to regularize fn
p_mean	Mean of normal distribution used as prior penalty in MLE of logit(fn)
p_sd	Standard deviation of normal distribution used as prior penalty in MLE of logit(fn)
StanRate	Logical; if TRUE, a simple 'Stan' model is used to estimate mutation rates for fast_analysis; this may add a couple minutes to the runtime of the analysis.
Stan_data	List; if StanRate is TRUE, then this is the data passed to the 'Stan' model to estimate mutation rates. If using the bakRFit wrapper of fast_analysis, then this is created automatically.
null_cutoff	bakR will test the null hypothesis of |effect size| < |null_cutoff|
NSS	Logical; if TRUE, logit(fn)s are compared rather than log(kdeg) so as to avoid steady-state assumption.
Chase	Logical; Set to TRUE if analyzing a pulse-chase experiment. If TRUE, kdeg = -ln(fn)/tl where fn is the fraction of reads that are s4U (more properly referred to as the fraction old in the context of a pulse-chase experiment)
BDA_model	Logical; if TRUE, variance is regularized with scaled inverse chi-squared model. Otherwise a log-normal model is used.

Details

Unless the user supplies estimates for pnew and pold, the first step of `fast_analysis` is to estimate the background and metabolic label (will refer to as s4U for simplicity, though bakR is compatible with other metabolic labels such as s6G) induced mutation rates. The former is best performed with a -s4U control sample, that is, a normal RNA-seq sample that lacks a -s4U feed or TimeLapse chemistry conversion of s4U to a C analog. If this sample is missing, both background and s4U induced mutation rates are estimated from the s4U fed samples. For the s4U mutation rate, features with sufficient read depth, as defined by the `read_cut` parameter, and the highest mutation rates are assumed to be completely labeled. Thus, the average mutation rates in these features is taken as the estimate of the s4U induced mutation rate in that sample. s4U induced mutation rates are estimated on a per-sample basis as there is often much more variability in these mutation rates than in the background mutation rates.

If a -s4U control is included, the background mutation rate is estimated using all features in the control sample(s) with read depths greater than `read_cut`. The average mutation rate among these features is taken as the estimated background mutation rate, and that background is assumed to be constant for all samples. If a -s4U control is missing, then a strategy similar to that used to estimate s4U induced mutation rates is used. In this case, the lowest mutation rate features with sufficient read depths are used, and there average mutation rate is the background mutation rate estimate, as these features are assumed to be almost entirely unlabeled. Another slightly more computationally intensive but more accurate strategy to estimate mutation rates is to set `StanRate = TRUE`. This will fit a non-hierarchical mixture model to a small subset of transcripts using 'Stan'. The default in `bakRFit` is to use 25 transcripts. If `StanRate` is TRUE, then a data list must be passed to `Stan_data` of the form that appears in the bakRFit object's Data_list$Stan_data entry.

Once mutation rates are estimated, fraction news for each feature in each sample are estimated. The approach utilized is MLE using the L-BFGS-B algorithm implemented in `stats::optim`. The assumed likelihood function is derived from a Poisson mixture model with rates adjusted according to each feature's empirical U-content (the average number of Us present in sequencing reads mapping to that feature in a particular sample). Fraction new estimates are then converted to degradation rate constant estimates using a solution to a simple ordinary differential equation model of RNA metabolism.

Once fraction new and kdegs are estimated, the uncertainty in these parameters is estimated using the Fisher Information. In the limit of large datasets, the variance of the MLE is inversely proportional to the Fisher Information evaluated at the MLE. Mixture models are typically singular, meaning that the Fisher information matrix is not positive definite and asymptotic results for the variance do not necessarily hold. As the mutation rates are estimated a priori and fixed to be > 0, these problems are eliminated. In addition, when assessing the uncertainty of replicate fraction new estimates, the size of the dataset is the raw number of sequencing reads that map to a particular feature. This number is often large (>100) which increases the validity of invoking asymptotics.

With kdegs and their uncertainties estimated, replicate estimates are pooled and regularized. There are two key steps in this downstream analysis. 1st, the uncertainty for each feature is used to fit a linear ln(uncertainty) vs. log10(read depth) trend, and uncertainties for individual features are shrunk towards the regression line. The uncertainty for each feature is a combination of the Fisher Information asymptotic uncertainty as well as the amount of variability seen between estimates. Regularization of uncertainty estimates is performed using the analytic results of a Normal distribution likelihood with known mean and unknown variance and conjugate priors. The prior parameters are estimated from the regression and amount of variability about the regression line. The strength of regularization can be tuned by adjusting the `prior_weight` parameter, with larger

numbers yielding stronger shrinkage towards the regression line. The 2nd step is to regularize the average kdeg estimates. This is done using the analytic results of a Normal distribution likelihood model with unknown mean and known variance and conjugate priors. The prior parameters are estimated from the population wide kdeg distribution (using its mean and standard deviation as the mean and standard deviation of the normal prior). In the 1st step, the known mean is assumed to be the average kdeg, averaged across replicates and weighted by the number of reads mapping to the feature in each replicate. In the 2nd step, the known variance is assumed to be that obtained following regularization of the uncertainty estimates.

Effect sizes (changes in kdeg) are obtained as the difference in log(kdeg) means between the reference and experimental sample(s), and the log(kdeg)s are assumed to be independent so that the variance of the effect size is the sum of the log(kdeg) variances. P-values assessing the significance of the effect size are obtained using a moderated t-test with number of degrees of freedom determined from the uncertainty regression hyperparameters and are adjusted for multiple testing using the Benjamini- Hochberg procedure to control false discovery rates (FDRs).

In some cases, the assumed ODE model of RNA metabolism will not accurately model the dynamics of a biological system being analyzed. In these cases, it is best to compare logit(fraction new)s directly rather than converting fraction new to log(kdeg). This analysis strategy is implemented when `NSS` is set to TRUE. Comparing logit(fraction new) is only valid If a single metabolic label time has been used for all samples. For example, if a label time of 1 hour was used for NR-seq data from WT cells and a 2 hour label time was used in KO cells, this comparison is no longer valid as differences in logit(fraction new) could stem from differences in kinetics or label times.

Value

List with dataframes providing information about replicate-specific and pooled analysis results. The output includes:

- Fn_Estimates; dataframe with estimates for the fraction new and fraction new uncertainty for each feature in each replicate. The columns of this dataframe are:
  - Feature_ID; Numerical ID of feature
  - Exp_ID; Numerical ID for experimental condition (Exp_ID from metadf)
  - Replicate; Numerical ID for replicate
  - logit_fn; logit(fraction new) estimate, unregularized
  - logit_fn_se; logit(fraction new) uncertainty, unregularized and obtained from Fisher Information
  - nreads; Number of reads mapping to the feature in the sample for which the estimates were obtained
  - log_kdeg; log of degradation rate constant (kdeg) estimate, unregularized
  - kdeg; degradation rate constant (kdeg) estimate
  - log_kd_se; log(kdeg) uncertainty, unregularized and obtained from Fisher Information
  - sample; Sample name
  - XF; Original feature name
- Regularized_ests; dataframe with average fraction new and kdeg estimates, averaged across the replicates and regularized using priors informed by the entire dataset. The columns of this dataframe are:
  - Feature_ID; Numerical ID of feature

  - Exp_ID; Numerical ID for experimental condition (Exp_ID from metadf)
  - avg_log_kdeg; Weighted average of log(kdeg) from each replicate, weighted by sample and feature-specific read depth
  - sd_log_kdeg; Standard deviation of the log(kdeg) estimates
  - nreads; Total number of reads mapping to the feature in that condition
  - sdp; Prior standard deviation for fraction new estimate regularization
  - theta_o; Prior mean for fraction new estimate regularization
  - sd_post; Posterior uncertainty
  - log_kdeg_post; Posterior mean for log(kdeg) estimate
  - kdeg; exp(log_kdeg_post)
  - kdeg_sd; kdeg uncertainty
  - XF; Original feature name
- Effects_df; dataframe with estimates of the effect size (change in logit(fn)) comparing each experimental condition to the reference sample for each feature. This dataframe also includes p-values obtained from a moderated t-test. The columns of this dataframe are:
  - Feature_ID; Numerical ID of feature
  - Exp_ID; Numerical ID for experimental condition (Exp_ID from metadf)
  - L2FC(kdeg); Log2 fold change (L2FC) kdeg estimate or change in logit(fn) if NSS TRUE
  - effect; LFC(kdeg)
  - se; Uncertainty in L2FC_kdeg
  - pval; P-value obtained using effect_size, se, and a z-test
  - padj; pval adjusted for multiple testing using Benjamini-Hochberg procedure
  - XF; Original feature name
- Mut_rates; list of two elements. The 1st element is a dataframe of s4U induced mutation rate estimates, where the mut column represents the experimental ID and the rep column represents the replicate ID. The 2nd element is the single background mutation rate estimate used
- Hyper_Parameters; vector of two elements, named a and b. These are the hyperparameters estimated from the uncertainties for each feature, and represent the two parameters of a Scaled Inverse Chi-Square distribution. Importantly, a is the number of additional degrees of freedom provided by the sharing of uncertainty information across the dataset, to be used in the moderated t-test.
- Mean_Variance_lms; linear model objects obtained from the uncertainty vs. read count regression model. One model is run for each Exp_ID

Examples

```
# Simulate small dataset
sim <- Simulate_bakRData(300, nreps = 2)

# Fit fast model to get fast_df
Fit <- bakRFit(sim$bakRData)

# Fit fast model with fast_analysis
Fast_Fit <- fast_analysis(Fit$Data_lists$Fast_df)
```

FnPCA *Creating PCA plots with logit(fn) estimates*

Description

This function creates a 2-component PCA plot using logit(fn) estimates.

Usage

```
FnPCA(obj, log_kdeg = FALSE)
```

Arguments

obj	Object contained within output of bakRFit. So, either Fast_Fit (MLE implementation fit), Stan_Fit (MCMC implementation fit), or Hybrid_Fit (Hybrid implementation fit)
log_kdeg	Boolean; if TRUE, then log(kdeg) estimates used for PCA rather than logit(fn). Currently only compatible with Fast_Fit

Value

A ggplot2 object.

Examples

```
# Simulate data for 500 genes and 2 replicates
sim <- Simulate_bakRData(500, nreps = 2)

# Fit data with fast implementation
Fit <- bakRFit(sim$bakRData)

# Fn PCA
FnPCA(Fit$Fast_Fit)

# log(kdeg) PCA
FnPCA(Fit$Fast_Fit, log_kdeg = TRUE)
```

Heatmap_kdeg *Creating a L2FC(kdeg) matrix that can be passed to heatmap functions*

Description

Heatmap_kdeg creates a matrix where each column represents a pair of samples (reference and experimental) and each row represents a feature. The entry in the ith row and jth column is the L2FC(kdeg) for feature i when comparing sample with experimental ID j+1 to the reference sample

Usage

```
Heatmap_kdeg(obj, zscore = FALSE, filter_sig = FALSE, FDR = 0.05)
```

Arguments

obj	Object outputted by bakRFit
zscore	Logical; if TRUE, then each matrix entry is log-odds fold change in the fraction new (a.k.a the effect size) divided by the uncertainty in the effect size
filter_sig	Logical; if TRUE, then only features which have a statistically significant L2FC(kdeg) in at least one comparison are kept
FDR	Numeric; False discovery to control at if filter_sig is TRUE.

Value

A matrix. Rows represent transcripts which were differentially expressed and columns represent (from left to right) differential kinetics z-score, differential expression z-score, and a mechanism score where positive represents synthesis driven and negative degradation driven changes in expression.

Examples

```
# Simulate data
sim <- Simulate_bakRData(1000)

# Fit data with fast implementation
Fit <- bakRFit(sim$bakRData)

# L2FC(kdeg) heatmap matrix
L2FC_kdeg_heat <- Heatmap_kdeg(Fit$Fast_Fit)
```

metadf *Example meatdf data frame*

Description

metadf dataframe describing the data present in the cB file that can be loaded with `data(cB_small)`. The contents are discussed in great detail in the Getting_started vignette.

Usage

```
data(metadf)
```

Format

A dataframe with 6 rows and 2 variables: row names are samples in the corresponding cB

tl time of s4U labeling, in hours

Exp_ID numerical ID of reference and experimental conditions; 1 is reference and 2 is the single experimental condition

Examples

```
data(cB_small)
data(metadf)
bakRdat <- bakRData(cB_small, metadf)
```

new_bakRData *bakRData object constructor for internal use*

Description

This function efficiently creates an object of class bakRData without performing rigorous checks

Usage

```
new_bakRData(cB, metadf)
```

Arguments

`cB`	Dataframe with columns corresponding to feature ID, number of Ts, number of mutations, sample ID, and number of identical observations
`metadf`	Dataframe detailing s4U label time and experimental ID of each sample

NSSHeat *Construct heatmap for non-steady state (NSS) analysis*

Description

This uses the output of bakR and a differential expression analysis software to construct a dataframe that can be passed to pheatmap::pheatmap(). This heatmap will display the result of a steady-state quasi-independent analysis of NR-seq data.

Usage

```
NSSHeat(
  bakRFit,
  DE_df,
  bakRModel = c("MLE", "Hybrid", "MCMC"),
  DE_cutoff = 0.05,
  bakR_cutoff = 0.05,
  Exp_ID = 2,
  lid = 4
)
```

Arguments

bakRFit	bakRFit object
DE_df	dataframe of required format with differential expression analysis results. See Further-Analyses vignette for details on what this dataframe should look like
bakRModel	Model fit from which bakR implementation should be used? Options are MLE, Hybrid, or MCMC
DE_cutoff	padj cutoff for calling a gene differentially expressed
bakR_cutoff	padj cutoff for calling a fraction new significantly changed
Exp_ID	Exp_ID of experimental sample whose comparison to the reference sample you want to use. Only one reference vs. experimental sample comparison can be used at a time
lid	Maximum absolute value for standardized score present in output. This is for improving aesthetics of any heatmap generated with the output.

Value

returns data frame that can be passed to pheatmap::pheatmap()

Examples

```
# Simulate small dataset
sim <- Simulate_bakRData(100, nreps = 2)
```

```
# Analyze data with bakRFit
Fit <- bakRFit(sim$bakRData)

# Number of features that made it past filtering
NF <- nrow(Fit$Fast_Fit$Effects_df)

# Simulate mock differential expression data frame
DE_df <- data.frame(XF = as.character(1:NF),
                        log2FoldChange = stats::rnorm(NF, 0, 2))

DE_df$stat <- DE_df$log2FoldChange/0.5

DE_df$padj <- 2*stats::dnorm(-abs(DE_df$stat))

# make heatmap matrix
Heatmap <- NSSHeat(bakRFit = Fit,
               DE_df = DE_df,
               bakRModel = "MLE")
```

---

plotMA *Creating L2FC(kdeg) MA plot from fit objects*

Description

This function outputs a L2FC(kdeg) MA plot. Plots are colored according to statistical significance and the sign of L2FC(kdeg)

Usage

```
plotMA(
  obj,
  Model = c("MLE", "Hybrid", "MCMC"),
  FDR = 0.05,
  Exps = NULL,
  Exp_shape = FALSE
)
```

Arguments

obj	Object of class bakRFit outputted by bakRFit function
Model	String identifying implementation for which you want to generate an MA plot
FDR	False discovery rate to control at for significance assessment
Exps	Vector of Experimental IDs to include in plot; must only contain elements within 2:(# of experimental IDs)
Exp_shape	Logical indicating whether to use Experimental ID as factor determining point shape in volcano plot

Value

A ggplot object. Each point represents a transcript. The x-axis is log-10 transformed replicate average read counts, y-axis is the log-2 fold-change in the degradation rate constant.

Examples

```
# Simulate data for 500 genes and 2 replicates
sim <- Simulate_bakRData(500, nreps = 2)

# Fit data with fast implementation
Fit <- bakRFit(sim$bakRData)

# Volcano plot
plotMA(Fit, Model = "MLE")
```

---

plotVolcano *Creating L2FC(kdeg) volcano plot from fit objects*

Description

This function creates a L2FC(kdeg) volcano plot. Plots are colored according to statistical significance and sign of L2FC(kdeg).

Usage

```
plotVolcano(obj, FDR = 0.05, Exps = NULL, Exp_shape = FALSE)
```

Arguments

obj	Object contained within output of bakRFit. So, either Fast_Fit (MLE implementation fit), Stan_Fit (MCMC implementation fit), or Hybrid_Fit (Hybrid implementation fit)
FDR	False discovery rate to control at for significance assessment
Exps	Vector of Experimental IDs to include in plot; must only contain elements within 2:(# of experimental IDs)
Exp_shape	Logical indicating whether to use Experimental ID as factor determining point shape in volcano plot

Value

A ggplot object. Each point represents a transcript. The x-axis is the log-2 fold change in the degradation rate constant and the y-axis is the log-10 transformed multiple test adjusted p value.

Examples

```
# Simulate data for 500 genes and 2 replicates
sim <- Simulate_bakRData(500, nreps = 2)

# Fit data with fast implementation
Fit <- bakRFit(sim$bakRData)

# Volcano plot
plotVolcano(Fit$Fast_Fit)
```

`reliableFeatures` *Identify features (e.g., transcripts) with high quality data*

Description

This function identifies all features (e.g., transcripts, exons, etc.) for which the mutation rate is below a set threshold in the control (no s4U) sample and which have more reads than a set threshold in all samples. If there is no -s4U sample, then only the read count cutoff is considered. Additional filtering options are only relevant if working with short RNA-seq read data. This includes filtering out features with extremely low empirical U-content (i.e., the average number of Us in sequencing reads from that feature) and those with very few reads having at least 3 Us in them.

Usage

```
reliableFeatures(obj, high_p = 0.2, totcut = 50, Ucut = 0.25, AvgU = 4)
```

Arguments

`obj`	Object of class bakRData
`high_p`	highest mutation rate accepted in control samples
`totcut`	readcount cutoff
`Ucut`	Must have a fraction of reads with 2 or less Us less than this cutoff in all samples
`AvgU`	Must have an average number of Us greater than this

Value

vector of gene names that passed reliability filter

Examples

```
# Load cB
data("cB_small")

# Load metadf
data("metadf")

# Create bakRData
bakRData <- bakRData(cB_small, metadf)

# Find reliable features
features_to_keep <- reliableFeatures(obj = bakRData)
```

`Simulate_bakRData` *Simulating nucleotide recoding data*

Description

`Simulate_bakRData` simulates a `bakRData` object. It's output also includes the simulated values of all kinetic parameters of interest. Only the number of genes (`ngene`) has to be set by the user, but an extensive list of additional parameters can be adjusted.

Usage

```
Simulate_bakRData(
  ngene,
  num_conds = 2L,
  nreps = 3L,
  eff_sd = 0.75,
  eff_mean = 0,
  fn_mean = 0,
  fn_sd = 1,
  kslog_c = 0.8,
  kslog_sd = 0.95,
  tl = 60,
  p_new = 0.05,
  p_old = 0.001,
  read_lengths = 200L,
  p_do = 0,
  noise_deg_a = -0.3,
  noise_deg_b = -1.5,
  noise_synth = 0.1,
  sd_rep = 0.05,
  low_L2FC_ks = -1,
  high_L2FC_ks = 1,
```

```
  num_kd_DE = c(0L, as.integer(rep(round(as.integer(ngene)/2), times =
    as.integer(num_conds) - 1))),
  num_ks_DE = rep(0L, times = as.integer(num_conds)),
  scale_factor = 150,
  sim_read_counts = TRUE,
  a1 = 5,
  a0 = 0.01,
  nreads = 50L,
  alpha = 25,
  beta = 75,
  STL = FALSE,
  STL_len = 40
)
```

Arguments

ngene	Number of genes to simulate data for
num_conds	Number of experimental conditions (including the reference condition) to simulate
nreps	Number of replicates to simulate
eff_sd	Effect size; more specifically, the standard deviation of the normal distribution from which non-zero changes in logit(fraction new) are pulled from.
eff_mean	Effect size mean; mean of normal distribution from which non-zero changes in logit(fraction new) are pulled from. Note, setting this to 0 does not mean that some of the significant effect sizes will be 0, as any exact integer is impossible to draw from a continuous random number generator. Setting this to 0 just means that there is symmetric stabilization and destabilization
fn_mean	Mean of fraction news of simulated transcripts in reference condition. The logit(fraction) of RNA from each transcript that is metabolically labeled (new) is drawn from a normal distribution with this mean
fn_sd	Standard deviation of fraction news of simulated transcripts in reference condition. The logit(fraction) of RNA from each transcript that is metabolically labeled (new) is drawn from a normal distribution with this sd
kslog_c	Synthesis rate constants will be drawn from a lognormal distribution with meanlog = kslog_c - mean(log(kd_mean)) where kd_mean is determined from the fraction new simulated for each gene as well as the label time (tl).
kslog_sd	Synthesis rate lognormal standard deviation; see kslog_c documentation for details
tl	metabolic label feed time
p_new	metabolic label (e.g., s4U) induced mutation rate. Can be a vector of length num_conds
p_old	background mutation rate
read_lengths	Total read length for each sequencing read (e.g., PE100 reads correspond to read_lengths = 200)

p_do	Rate at which metabolic label containing reads are lost due to dropout; must be between 0 and 1
noise_deg_a	Slope of trend relating log10(standardized read counts) to log(replicate variability)
noise_deg_b	Intercept of trend relating log10(standardized read counts) to log(replicate variability)
noise_synth	Homoskedastic variability of L2FC(ksyn)
sd_rep	Variance of lognormal distribution from which replicate variability is drawn
low_L2FC_ks	Most negative L2FC(ksyn) that can be simulated
high_L2FC_ks	Most positive L2FC(ksyn) that can be simulated
num_kd_DE	Vector where each element represents the number of genes that show a significant change in stability relative to the reference. 1st entry must be 0 by definition (since relative to the reference the reference sample is unchanged)
num_ks_DE	Same as num_kd_DE but for significant changes in synthesis rates.
scale_factor	Factor relating RNA concentration (in arbitrary units) to average number of read counts
sim_read_counts	Logical; if TRUE, read counts are simulated as coming from a heterodisperse negative binomial distribution
a1	Heterodispersion 1/reads dependence parameter
a0	High read depth limit of negative binomial dispersion parameter
nreads	Number of reads simulated if sim_read_counts is FALSE
alpha	shape1 parameter of the beta distribution from which U-contents (probability that a nucleotide in a read from a transcript is a U) are drawn for each gene.
beta	shape2 parameter of the beta distribution from which U-contents (probability that a nucleotide in a read from a transcript is a U) are drawn for each gene.
STL	logical; if TRUE, simulation is of STL-seq rather than a standard TL-seq experiment. The two big changes are that a short read length is required (< 60 nt) and that every read for a particular feature will have the same number of Us. Only one read length is simulated for simplicity.
STL_len	Average length of simulated STL-seq length. Since Pol II typically pauses about 20-60 bases from the promoter, this should be around 40

Details

Simulate_bakRData simulates a bakRData object using a realistic generative model with many adjustable parameters. Average RNA kinetic parameters are drawn from biologically inspired distributions. Replicate variability is simulated by drawing a feature's fraction new in a given replicate from a logit-Normal distribution with a heteroskedastic variance term with average magnitude given by the chosen read count vs. variance relationship. For each replicate, a feature's ksyn is drawn from a homoskedastic lognormal distribution. Read counts can either be set to the same value for all simulated features or can be simulated according to a heterodisperse negative binomial distribution. The latter is the default

The number of Us in each sequencing read is drawn from a binomial distribution with number of trials equal to the read length and probability of each nucleotide being a U drawn from a beta distribution. Each read is assigned to the new or old population according to a Bernoulli distribution with p = fraction new. The number of mutations in each read are then drawn from one of two binomial distributions; if the read is assigned to the population of new RNA, the number of mutations are drawn from a binomial distribution with number of trials equal to the number of Us and probability of mutation = p_new; if the read is assigned to the population of old RNA, the number of mutations is instead drawn from a binomial distribution with the same number of trials but with the probability of mutation = p_old. p_new must be greater than p_old because mutations in new RNA arise from both background mutations that occur with probability p_old as well as metabolic label induced mutations

Simulated read counts should be treated as if they are spike-in and RPKM normalized, so the same scale factor can be applied to each sample when comparing the sequencing reads (e.g., if you are performing differential expression analysis).

Function to simulate a bakRData object according to a realistic generative model

Value

A list containing a simulated bakRData object as well as a list of simulated kinetic parameters of interest. The contents of the latter list are:

- Effect_sim; Dataframe meant to mimic formatting of Effect_df that are part of bakRFit(StanFit = TRUE), bakRFit(HybridFit = TRUE) and bakRFit(bakRData object) output.
- Fn_mean_sim; Dataframe meant to mimic formatting of Regularized_ests that is part of bakRFit(bakRData object) output. Contains information about the true fraction new simulated in each condition (the mean of the normal distribution from which replicate fraction news are simulated)
- Fn_rep_sim; Dataframe meant to mimic formatting of Fn_Estimates that is part of \codebakRFit(bakRData object) output. Contains information about the fraction new simulated for each feature in each replicate of each condition.
- L2FC_ks_mean; The true L2FC(ksyn) for each feature in each experimental condition. The i-th column corresponds to the L2FC(ksyn) when comparing the i-th condition to the reference condition (defined as the 1st condition) so the 1st column is always all 0s
- RNA_conc; The average number of normalized read counts expected for each feature in each sample.

Examples

```
# 2 replicate, 2 experimental condition, 1000 gene simulation
sim_2reps <- Simulate_bakRData(ngene = 1000, nreps = 2)

# 3 replicate, 2 experimental condition, 1000 gene simulation
# with 100 instances of differential degradation kinetics
sim_3reps <- Simulate_bakRData(ngene = 1000, num_kd_DE = c(0, 100))

# 2 replicates, 3 experimental condition, 1000 gene simulation
# with 100 instances of differential degradation kinetics in the 1st
# condition and no instances of differential degradation kinetics in the
```

```
# 2nd condition
sim_3es <- Simulate_bakRData(ngene = 1000,
                             nreps = 2,
                             num_conds = 3,
                             num_kd_DE = c(0, 100, 0))
```

TL_stan *Fit 'Stan' models to nucleotide recoding RNA-seq data analysis*

Description

TL_stan is an internal function to analyze nucleotide recoding RNA-seq data with a fully Bayesian hierarchical model implemented in the PPL Stan. TL_stan estimates kinetic parameters and differences in kinetic parameters between experimental conditions. When assessing differences, a single reference sample is compared to each collection of experimental samples provided.

Usage

```
TL_stan(
  data_list,
  Hybrid_Fit = FALSE,
  keep_fit = FALSE,
  NSS = FALSE,
  chains = 1,
  ...
)
```

Arguments

data_list	List to pass to 'Stan' of form given by cBprocess
Hybrid_Fit	Logical; if TRUE, Hybrid 'Stan' model that takes as data output of fast_analysis is run.
keep_fit	Logical; if TRUE, 'Stan' fit object is included in output; typically large file so default FALSE.
NSS	Logical; if TRUE, models that directly compare logit(fn)s are used to avoid steady-state assumption
chains	Number of Markov chains to sample from. The default is to only run a single chain. Typical NR-seq datasets yield very memory intensive analyses, but running a single chain should decrease this burden. For reference, running the MCMC implementation (Hybrid_Fit = FALSE) with 3 chains on an NR-seq dataset with 3 replicates of 2 experimental conditions with around 20 million raw (unmapped) reads per sample requires over 100 GB of RAM. With a single chain, this burden drops to around 20 GB. Due to memory demands and time constraints (runtimes for the MCMC implementation border will likely be around 1-2 days) means that these models should usually be run in a specialized High Performance Computing (HPC) system.

... Arguments passed to `rstan::sampling` (e.g. iter, warmup, etc.).

Details

Two implementations of a similar model can be fit with TL_stan: a complete nucleotide recoding RNA-seq analysis and a hybrid analysis that takes as input results from `fast_analysis`. In the complete analysis (referred to in the bakR publication as the MCMC implementation), U-to-C mutations are modeled as coming from a Poisson distribution with rate parameter adjusted by the empirical U-content of each feature analyzed. Features represent whatever the user defined them to be when constructing the bakR data object. Typical feature categories are genes, exons, etc. Hierarchical modeling is used to pool data across replicates, across features, and across replicates. More specifically, replicate data for the same feature are partially pooled to estimate feature-specific mean fraction news and uncertainties. Feature means are partially pooled to estimate dataset-wide mean fraction news and standard deviations. The replicate variability for each feature is also partially pooled to determine a condition-wide heteroskedastic relationship between read depths and replicate variability. Partial pooling reduces subjectivity when determining priors by allowing the model to determine what priors make sense given the data. Partial pooling also regularizes estimates, reducing estimate variability and thus increasing estimate accuracy. This is particularly important for replicate variability estimates, which often rely on only a few replicates of data per feature and thus are typically highly unstable.

The hybrid analysis (referred to in the bakR publication as the Hybrid implementation) inherits the hierarchical modeling structure of the complete analysis, but reduces computational burden by foregoing per-replicate-and-feature fraction new estimation and uncertainty quantification. Instead, the hybrid analysis takes as data fraction new estimates and approximate uncertainties from `fast_analysis`. Runtimes of the hybrid analysis are thus often an order or magnitude or more shorter than with the complete analysis, but loses some accuracy by relying on point estimates and uncertainty quantification that is only valid in the limit of large dataset sizes (where the dataset size for the per-replicate-and-feature fraction new estimate is the raw number of sequencing reads mapping to the feature in that replicate).

Users also have the option to save or discard the 'Stan' fit object. Fit objects can be exceedingly large (> 10 GB) for most nucleotide recoding RNA-seq datasets. Therefore, if you don't want to store such a large object, a summary object will be saved instead, which greatly reduces the size of the output (~ 10-50 MB) while still retaining much of the important information. In addition, the output of `TL_stan` provides the estimates and uncertainties for key parameters (L2FC(kdeg), kdeg, and fraction new) that will likely be of most interest. That being said, there are some analyses that are only possible if the original fit object is saved. For example, the fit object will contain all of the samples from the posterior collected during model fitting. Thus, new parameters (e.g., L2FC(kdeg)'s comparing two experimental samples) not naturally generated by the model can be estimated post-hoc. Still, there are often approximate estimates that can be obtained for such parameters that don't rely on the full fit object. One analysis that is impossible without the original fit object is generating model diagnostic plots. These include trace plots (to show mixing and efficient parameter space exploration of the Markov chains), pairs plots (to show correlations between parameters and where any divergences occurred), and other visualizations that can help users assess how well a model ran. Because the models implemented by `TL_stan` are extensively validated, it is less likely that such diagnostics will be helpful, but often anomalies on your data can lead to poor model convergence, in which case assessing model diagnostics can help you identify the source of problems in your data. Summary statistics describing how well the model was able to estimate each parameter (n_eff and rhat) are provided in the fit summaries, but can often obscure some of the

nuanced details of model fitting.

Value

A list of objects:

- Effects_df; dataframe with estimates of the effect size (change in logit(fn)) comparing each experimental condition to the reference sample for each feature. This dataframe also includes p-values obtained from a moderated t-test. The columns of this dataframe are:
  - Feature_ID; Numerical ID of feature
  - Exp_ID; Numerical ID for experimental condition (Exp_ID from metadf)
  - L2FC_kdeg; L2FC(kdeg) posterior mean
  - L2FC_kd_sd; L2FC(kdeg) posterior sd
  - effect; identical to L2FC_kdeg (kept for symmetry with MLE fit output)
  - se; identical to L2FC_kd_sd (kept for symmetry with MLE fit output)
  - XF; Feature name
  - pval; p value obtained from effect and se + z-test
  - padj; p value adjusted for multiple testing using Benjamini-Hochberg procedure
- Kdeg_df; dataframe with estimates of the kdeg (RNA degradation rate constant) for each feature, averaged across replicate data. The columns of this dataframe are:
  - Feature_ID; Numerical ID of feature
  - Exp_ID; Numerical ID for experimental condition
  - kdeg; Degradation rate constant posterior mean
  - kdeg_sd; Degradation rate constant posterior standard deviation
  - XF; Original feature name
- Fn_Estimates; dataframe with estimates of the logit(fraction new) for each feature in each replicate. The columns of this dataframe are:
  - Feature_ID; Numerical ID for feature
  - Exp_ID; Numerical ID for experimental condition (Exp_ID from metadf)
  - Replicate; Numerical ID for replicate
  - logit_fn; Logit(fraction new) posterior mean
  - logit_fn_se; Logit(fraction new) posterior standard deviation
  - sample; Sample name
  - XF; Original feature name
- Fit_Summary; only outputted if keep_fit == FALSE. Summary of 'Stan' fit object with each row corresponding to a particular parameter. All posterior point descriptions are descriptions of the marginal posterior distribution for the parameter in that row. For example, the posterior mean is the average value for the parameter when averaging over all other parameter values. The columns of this dataframe are:
  - mean; Posterior mean for the parameter given by the row name
  - se_mean; Standard error of the posterior mean; essentially how confident the model is that what it estimates to be the posterior mean is what the posterior mean actually is. This will depend on the number of chains run on the number of iterations each chain is run for.
  - sd; Posterior standard deviation

  - 2.5%; 2.5th percentile of the posterior distribution. 2.5% of the posterior mass is below this point
  - 25%; 25th percentile of the posterior distribution
  - 50%; 50th percentile of the posterior distribution
  - 75%; 75th percentile of the posterior distribution
  - 97.5%; 97.5th percentile of the posterior distribution
  - n_eff; Effective sample size. The larger this is the better, though it should preferably be around the total number of iterations (iter x chains). Small values of this could represent poor model convergence
  - Rhat; Describes how well separate Markov chains mixed. This is preferably as close to 1 as possible, and values higher than 1 could represent poor model convergence
- Stan_Fit; only outputted if keep_fit == TRUE. This is the full 'Stan' fit object, an R6 object of class `stanfit`
- Mutation_Rates; data frame with information about mutation rate estimates. Has the same columns as Fit_Summary. Each row corresponds to either a background mutation rate (log_lambda_o) or an s4U induced mutation rate (log_lambda_n), denoted in the parameter column. The bracketed portion of the parameter name will contain two numbers. The first corresponds to the Exp_ID and the second corresponds to the Replicate_ID. For example, if the parameter name is log_lambda_o[1,2] then that row corresponds to the background mutation rate in the second replicate of experimental condition one. A final point to mention is that the estimates are on a log(avg. # of mutations) scale. So a log_lambda_n of 1 means that on average, there are an estimated 2.72 (exp(1)) mutations in reads from new RNA (i.e., RNA synthesized during s4U labeling).

`validate_bakRData` *bakR Data object validator*

Description

This functions ensures that input for bakRData object construction is valid

Usage

```
validate_bakRData(obj)
```

Arguments

`obj`	An object of class bakRData

Index