Package 'trace'

January 16, 2025

Title Tandem Repeat Analysis by Capillary Electrophoresis

Version 0.5.0

Description A pipeline for short tandem repeat instability analysis from fragment analysis data. Inputs of fsa files or peak tables, and a user supplied metadata data-frame. The package identifies ladders, calls peaks, identifies the modal peaks, calls repeats, then calculates repeat instability metrics (e.g. expansion index from Lee et al. (2010) <doi:10.1186/1752-0509-4-29>).

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Suggests dplyr, ggplot2, knitr, rmarkdown, shinytest2, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports graphics, grDevices, lme4, methods, mgcv, plotly, pracma, seqinr, shiny, stats, utils

VignetteBuilder knitr

Depends R (>= 2.10)

LazyData true

URL https://zachariahmclean.github.io/trace/

NeedsCompilation no

Author Zachariah McLean [aut, cre, cph] (<https://orcid.org/0000-0002-0968-0538>), Kevin Correia [aut], Andrew Jiang [ctb]

Maintainer Zachariah McLean <zachariah.louis.mclean@gmail.com>

Repository CRAN

Date/Publication 2025-01-16 11:00:02 UTC

Contents

`add_metadata` *Add Metadata to Fragments List*

Description

This function adds metadata information to a list of fragments.

Usage

```
add_metadata(
  fragments_list,
  metadata_data.frame,
  unique_id = "unique_id",
  metrics_group_id = "metrics_group_id",
  metrics_baseline_control = "metrics_baseline_control",
  batch_run_id = "batch_run_id",
  batch_sample_id = "batch_sample_id",
  batch_sample_modal_repeat = "batch_sample_modal_repeat"
)
```

Arguments

`fragments_list`	A list of fragment objects to which metadata will be added.
`metadata_data.frame`	A data frame containing the metadata information.
`unique_id`	(required) A character string indicating the column name for unique sample identifiers in the metadata.
`metrics_group_id`	(optional) A character string indicating the column name for sample group identifiers in the metadata. This is for the logical grouping of samples for metrics calculations (see `assign_index_peaks()`). To skip, provide NA.
`metrics_baseline_control`	(optional) A character string indicating the column name for baseline control indicators in the metadata. This is to identify the baseline control samples for grouping of samples for metrics calculations (see `assign_index_peaks()`). To skip, provide NA.
`batch_run_id`	(optional) A character string indicating the column name for the batch run identifiers in the metadata. This is for either batch correction or repeat correction in `call_repeats()`. To skip, provide NA.
`batch_sample_id`	(optional) A character string indicating the column name for an id of the size standard. For example, a sample code. This is for either batch correction or repeat correction in `call_repeats()`. To skip, provide NA.
`batch_sample_modal_repeat`	(optional) A character string indicating column name for the validated modal repeat length of size standard sample. This is for either repeat correction in `call_repeats()`. To skip, provide NA.

Details

This function adds specified metadata attributes to each fragment in the list. It matches the unique sample identifiers from the fragments list with those in the metadata data frame. To skip any of the optional columns, make parameter NA.

There are two key things metadata are required for. First is the grouping of samples (metrics_group_id & metrics_baseline_control) for the calculation of metrics and is used in `assign_index_peaks()`.

For example, specifying a sample where the modal allele is the inherited repeat length (eg a mouse tail sample) or sample(s) at the start of a time-course experiment. This is indicated with a TRUE in the metrics_baseline_control column of the metadata. Samples are then grouped together with the metrics_group_id column of the metadata. Multiple samples can be metrics_baseline_control, which can be helpful for the average repeat gain metric to have a more accurate representation of the average repeat at the start of the experiment.

The second key thing metadata can be used for is corrections in `call_repeats()`. There are two main correction approaches in `call_repeats()` that are somewhat related: either 'batch' or 'repeat'. Batch correction is relatively simple and just requires you to link samples across batches to correct batch-batch variation in repeat sizes. However, even though the repeat size that is return will be precise, it will not be accurate and underestimates the real repeat length. By contrast, repeat correction can be used to accurately call repeat lengths (which also corrects the batch effects). However, the repeat correction will only be as good as your sample used to call the repeat length so this is a challenging and advanced feature. You need to use a sample that reliably returns the same peak as the modal peak, or you need to be willing to understand the shape of the distribution and manually validate the repeat length of each batch_sample_id for each run.

Batch correction uses common sample(s) across fragment analysis runs to correct systematic batch effects that occur with repeat-containing amplicons in capillary electrophoresis. There are slight fluctuations of size across runs for amplicons containing repeats that result in systematic differences around 1-3 base pairs. So, if samples are to be analyzed for different runs, the absolute bp size is not comparable unless this batch effect is corrected. This is only relevant when the absolute size of a amplicons are compared for grouping metrics as described above (otherwise instability metrics are all relative and it doesn't matter that there's systematic batch effects across runs) or when plotting traces from different runs. This correction can be achieved by running a couple of samples in every fragment analysis run, or having a single run that takes a couple of samples from every run together, thereby linking them. These samples are then indicated in the metadata with batch_run_id (to group samples by fragment analysis run) and batch_sample_id (to enable linking samples across batches).

Finally, samples with known and validated repeat size can be used to accurately call the repeat length (and therefore also correct batch effects) in `call_repeats()`. Similar to batch correction, batch_run_id (to group samples by fragment analysis run) and batch_sample_id (to enable linking samples across batches) are used, but importantly batch_sample_modal_repeat is also set. The batch_sample_modal_repeat is the validated repeat length of the modal repeat of the sample. This validated repeat length is then used to call the repeat length of the modal repeat for each sample (by each batch_run_id). Importantly, this correction requires you to know with confidence the repeat length of the modal peak of the sample. Therefore it's important that the sample used for repeat correction has a clear and prominent modal peak. If the repeat length is very long, it's common for the modal peak of a sample to change so if you use this feature you're going to have to understand the shape of the distribution of your sample and double check that the correct peak has been called as the modal peak after you have used `find_alleles()`. If a different peak is selected as the modal peak than usual, you need to go back to the metadata and adjust the repeat size of the size standard (For example, your size standard sample has been validated to have 120 repeats. You run `find_alleles()` and look at the distribution of peaks and notice that the peak one repeat unit higher is the modal peak this time. Therefore, you're going to need to set the batch_sample_modal_repeat as 121 in the metadata just for that batch_run_id. In the other runs you would keep the batch_sample_modal_repeat as 120.).

Value

This function modifies list of fragments objects in place with metadata added.

Examples

```
gm_raw <- trace::example_data
metadata <- trace::metadata

test_fragments <- peak_table_to_fragments(gm_raw,
  data_format = "genemapper5",
  dye_channel = "B",
  min_size_bp = 300
)

add_metadata(
  fragments_list = test_fragments,
  metadata_data.frame = metadata,
  unique_id = "unique_id",
  metrics_group_id = "metrics_group_id",
  metrics_baseline_control = "metrics_baseline_control",
  batch_run_id = "batch_run_id",
  batch_sample_id = "batch_sample_id",
  batch_sample_modal_repeat = "batch_sample_modal_repeat"
)

# skip unwanted metadata by using NA

add_metadata(
  fragments_list = test_fragments,
  metadata_data.frame = metadata,
  unique_id = "unique_id",
  metrics_group_id = "metrics_group_id",
  metrics_baseline_control = "metrics_baseline_control",
  batch_run_id = NA,
  batch_sample_id = NA,
  batch_sample_modal_repeat = NA
)
```

---

assign_index_peaks *Assign index peaks*

Description

Assign index peaks in preparation for calculation of instability metrics

Usage

```
assign_index_peaks(
  fragments_list,
  grouped = FALSE,
  index_override_dataframe = NULL
)
```

Arguments

fragments_list	A list of "fragments_repeats" class objects representing fragment data.
grouped	Logical value indicating whether samples should be grouped to share a common index peak. FALSE will assign the sample's own modal allele as the index peak. TRUE will use metadata to assign the index peak based on the modal peak of another sample (see below for more details).
index_override_dataframe	A data.frame to manually set index peaks. Column 1: unique sample IDs, Column 2: desired index peaks (the order of the columns is important since the information is pulled by column position rather than column name). Closest peak in each sample is selected so the number needs to just be approximate.

Details

A key part of instability metrics is the index peak. This is the repeat length used as the reference peak for relative instability metrics calculations, like expansion index. This is usually the the inherited repeat length of a mouse, or the modal repeat length for the cell line at a starting time point.

If grouped is set to TRUE, this function groups the samples by their metrics_group_id and uses the samples set as metrics_baseline_control to set the index peak. Use add_metadata() to set these variables. This is useful for cases like inferring repeat size of inherited alleles from mouse tail data. If the samples that are going to be used to assign index peak are from different fragment analysis runs, use correction = "batch" in call_repeats() to make sure the systematic differences between runs are corrected and the correct index peak is assigned. If there are multiple samples used as baseline control, the median value will be used to assign index peak to corresponding samples.

For mice, if just a few samples have the inherited repeat signal shorter than the expanded population, you could not worry about this and instead use the index_override_dataframe. This can be used to manually override these assigned index repeat values (irrespective of whether grouped is TRUE or FALSE).

As a final option, the index peak could be manually assigned directly to a fragments_repeats class using the internal setter function fragments_repeats$set_index_peak().

Value

This function modifies list of fragments_repeats objects in place with index_repeat and index_signal added.

Examples

```
fsa_list <- lapply(cell_line_fsa_list, function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

fragments_list <- find_fragments(fsa_list,
  min_bp_size = 300
)

find_alleles(
  fragments_list
)
call_repeats(
  fragments_list
)

add_metadata(
  fragments_list,
  metadata_data.frame = trace::metadata
)

assign_index_peaks(
  fragments_list,
  grouped = TRUE
)

plot_traces(fragments_list[1], xlim = c(100,150))
```

`calculate_instability_metrics`
Calculate Repeat Instability Metrics

Description

This function computes instability metrics from a list of fragments_repeats data objects.

Usage

```
calculate_instability_metrics(
  fragments_list,
  peak_threshold = 0.05,
  window_around_index_peak = c(NA, NA),
  percentile_range = c(0.5, 0.75, 0.9, 0.95),
  repeat_range = c(2, 5, 10, 20)
)
```

Arguments

`fragments_list`	A list of "fragments_repeats" objects representing fragment data.
`peak_threshold`	The threshold for peak signals to be considered in the calculations, relative to the modal peak signal of the expanded allele.
`window_around_index_peak`	A numeric vector (length = 2) defining the range around the index peak. First number specifies repeats before the index peak, second after. For example, `c(-5, 40)` around an index peak of 100 would analyze repeats 95 to 140. The sign of the numbers does not matter (The absolute value is found).
`percentile_range`	A numeric vector of percentiles to compute (e.g., c(0.5, 0.75, 0.9, 0.95)).
`repeat_range`	A numeric vector specifying ranges of repeats for the inverse quantile computation.

Details

Each of the columns in the supplied dataframe are explained below:

General Information:

- `unique_id`: A unique identifier for the sample (usually the fsa file name).

Quality Control:

- `QC_comments`: Quality control comments.
- `QC_modal_peak_signal`: Quality control status based on the modal peak signal (Low < 500, very low < 100).
- `QC_peak_number`: Quality control status based on the number of peaks (Low < 20, very low < 10).
- `QC_off_scale`: Quality control comments for off-scale peaks. Potential peaks that are off-scale are given. However, a caveat is that this could be from any of the channels (ie it could be from the ladder channel but is the same scan as the given repeat).

General sample metrics:

- `modal_peak_repeat`: The repeat size of the modal peak.
- `modal_peak_signal`: The signal of the modal peak.
- `index_peak_repeat`: The repeat size of the index peak (the repeat value closest to the modal peak of the index sample).
- `index_peak_signal`: The signal of the index peak.
- `index_weighted_mean_repeat`: The weighted mean repeat size (weighted on the signal of the peaks) of the index sample.
- `n_peaks_total`: The total number of peaks in the repeat table.
- `n_peaks_analysis_subset`: The number of peaks in the analysis subset.
- `n_peaks_analysis_subset_expansions`: The number of expansion peaks in the analysis subset.
- `min_repeat`: The minimum repeat size in the analysis subset.
- `max_repeat`: The maximum repeat size in the analysis subset.

- `mean_repeat`: The mean repeat size in the analysis subset.
- `weighted_mean_repeat`: The weighted mean repeat size (weight on peak signal) in the analysis subset.
- `median_repeat`: The median repeat size in the analysis subset.
- `max_signal`: The maximum peak signal in the analysis subset.
- `max_delta_neg`: The maximum negative delta to the index peak.
- `max_delta_pos`: The maximum positive delta to the index peak.
- `skewness`: The skewness of the repeat size distribution.
- `kurtosis`: The kurtosis of the repeat size distribution.

Repeat instability metrics:

- `modal_repeat_change`: The difference between the modal repeat and the index repeat.
- `average_repeat_change`: The weighted mean of the sample (weighted by peak signal) subtracted by the weighted mean repeat of the index sample(s).
- `instability_index_change`: The instability index of the sample subtracted by the instability index of the index sample(s). This will be very similar to the average_repeat_change, with the key difference of instability_index_change being that it is an internally calculated metric for each sample, and therefore the random slight fluctuations of bp size (or systematic if across plates for example) will be removed. However, it requires the index peak to be correctly set for each sample, and if set incorrectly, can produce large arbitrary differences.
- `instability_index`: The instability index based on peak signal and distance to the index peak. (See Lee et al., 2010, doi:10.1186/17520509429).
- `instability_index_abs`: The absolute instability index. The absolute value is taken for the "Change from the main allele".
- `expansion_index`: The instability index for expansion peaks only.
- `contraction_index`: The instability index for contraction peaks only.
- `expansion_ratio`: The ratio of expansion peaks' signals to the main peak signal. Also known as "peak proportional sum" (See Genetic Modifiers of Huntington's Disease (GeM-HD) Consortium, 2019, doi:10.1016/j.cell.2019.06.036).
- `contraction_ratio`: The ratio of contraction peaks' signals to the main peak signal.
- `expansion_percentile_*`: The repeat size at specified percentiles of the cumulative distribution of expansion peaks.
- `expansion_percentile_for_repeat_*`: The percentile rank of specified repeat sizes in the distribution of expansion peaks.

Value

A data.frame with calculated instability metrics for each sample.

Examples

```
gm_raw <- trace::example_data
metadata <- trace::metadata

test_fragments <- peak_table_to_fragments(gm_raw,
  data_format = "genemapper5",
  dye_channel = "B",
```

```
  min_size_bp = 400
)

add_metadata(
  fragments_list = test_fragments,
  metadata_data.frame = metadata
)

find_alleles(
  fragments_list = test_fragments,
  peak_region_size_gap_threshold = 6,
  peak_region_signal_threshold_multiplier = 1
)


call_repeats(
  fragments_list = test_fragments,
  assay_size_without_repeat = 87,
  repeat_size = 3
)

assign_index_peaks(
  fragments_list = test_fragments,
  grouped = TRUE
)


# grouped metrics
# uses t=0 samples as indicated in metadata
test_metrics_grouped <- calculate_instability_metrics(
  fragments_list = test_fragments,
  peak_threshold = 0.05,
  window_around_index_peak = c(-40, 40),
  percentile_range = c(0.5, 0.75, 0.9, 0.95),
  repeat_range = c(2, 5, 10, 20)
)
```

`call_repeats` *Call Repeats for Fragments*

Description

This function calls the repeat lengths for a list of fragments.

Usage

```
call_repeats(
  fragments_list,
  assay_size_without_repeat = 87,
  repeat_size = 3,
```

```
  correction = "none",
  force_whole_repeat_units = FALSE,
  force_repeat_pattern = FALSE,
  force_repeat_pattern_size_period = repeat_size * 0.93,
  force_repeat_pattern_size_window = 0.5
)
```

Arguments

`fragments_list`	A list of fragments_repeats objects containing fragment data.
`assay_size_without_repeat`	An integer specifying the assay size without repeat for repeat calling. This is the length of the sequence flanking the repeat in the PCR product.
`repeat_size`	An integer specifying the repeat size for repeat calling. Default is 3.
`correction`	A character vector of either "batch" to carry out a batch correction from common samples across runs (known repeat length not required), or "repeat" to use samples with validated modal repeat lengths to correct the repeat length. Requires metadata to be added (see `add_metadata()`) with both "batch" and "repeat" requiring `"batch_run_id"`, "batch" requiring (`"batch_sample_id"`) and "repeat" requiring `"batch_sample_modal_repeat"` (but also benefits from having `"batch_sample_id"`).
`force_whole_repeat_units`	A logical value specifying if the peaks should be forced to be whole repeat units apart. Usually the peaks are slightly under the whole repeat unit if left unchanged.
`force_repeat_pattern`	A logical value specifying if the peaks should be re called to fit the specific repeat unit pattern. This requires trace information so you must have started with fsa files.
`force_repeat_pattern_size_period`	A numeric value to set the peak periodicity bp size. In fragment analysis, the peaks are usually slightly below the actual repeat unit size, so you can use this value to fine tune what the periodicity should be.
`force_repeat_pattern_size_window`	A numeric value for the size window when assigning the peak. The algorithm jumps to the predicted scan for the next peak. This value opens a window of the given base pair size neighboring scans to pick the tallest in.

Details

This function has a lot of different options features for determining the repeat length of your samples. This includes i) an option to force the peaks to be whole repeat units apart, ii) corrections to correct batch effects or accurately call repeat length by comparing to samples of known length, and iii) algorithms or re-calling the peaks to remove any contaminating peaks or shoulder-peaks.

———— correction ————

There are two main correction approaches that are somewhat related: either 'batch' or 'repeat'. Batch correction is relatively simple and just requires you to link samples across batches to correct

batch-batch variation in repeat sizes. However, even though the repeat size that is return will be precise, it will not be accurate and underestimates the real repeat length. By contrast, repeat correction can be used to accurately call repeat lengths (which also corrects the batch effects). However, the repeat correction will only be as good as your sample used to call the repeat length so this is a challenging and advanced feature. You need to use a sample that reliably returns the same peak as the modal peak, or you need to be willing to understand the shape of the distribution and manually validate the repeat length of each batch_sample_id for each run.

- Batch correction uses common sample(s) across fragment analysis runs to correct systematic batch effects that occur with repeat-containing amplicons in capillary electrophoresis. There are slight fluctuations of size across runs for amplicons containing repeats that result in systematic differences around 1-3 base pairs. So, if samples are to be analyzed for different runs, the absolute bp size is not comparable unless this batch effect is corrected. This is only relevant when the absolute size of a amplicons are compared for grouping metrics as described above (otherwise instability metrics are all relative and it doesn't matter that there's systematic batch effects across runs) or when plotting traces from different runs. This correction can be achieved by running a couple of samples in every fragment analysis run, or having a single run that takes a couple of samples from every run together, thereby linking them. These samples are then indicated in the metadata with batch_run_id (to group samples by fragment analysis run) and batch_sample_id (to enable linking samples across batches) (see `add_metadata()`). Use `plot_batch_correction_samples()` to plot the samples before and after correction to make sure that is has worked as expected.
- Samples with known and validated repeat size can be used to accurately call the repeat length (and therefore also correct batch effects). Similar to batch correction, batch_run_id (to group samples by fragment analysis run) and batch_sample_id (to enable linking samples across batches) are used, but importantly batch_sample_modal_repeat is also set (see `add_metadata()`). The batch_sample_modal_repeat is the validated repeat length of the modal repeat of the sample. This validated repeat length is then used to call the repeat length of the modal repeat for each sample (by each batch_run_id). Importantly, this correction requires you to know with confidence the repeat length of the modal peak of the sample. Therefore it's important that the sample used for repeat correction has a clear and prominent modal peak. If the repeat length is very long, it's common for the modal peak of a sample to change so if you use this feature you're going to have to understand the shape of the distribution of your sample and double check that the correct peak has been called as the modal peak after you have used `find_alleles()`. If a different peak is selected as the modal peak than usual, you need to go back to the metadata and adjust the repeat size of the size standard (For example, your size standard sample has been validated to have 120 repeats. You run `find_alleles()` and look at the distribution of peaks and notice that the peak one repeat unit higher is the modal peak this time. Therefore, you're going to need to set the batch_sample_modal_repeat as 121 in the metadata just for that batch_run_id. In the other runs you would keep the batch_sample_modal_repeat as 120.). For repeat correction, there are several functions to help visualize and summarize the correction:
  - Use `plot_batch_correction_samples()` to visualize the same sample across different batches. This can be helpful to make sure that the correction has worked the same across different runs.
  - Use `plot_repeat_correction_model()` to visualize the linear model use to correct repeat length for each `batch_run_id`. This can be helpful to make sure the supplied repeat length of different samples are lining up within each run.

- Generate a summary table of the predicted repeat length for each sample and the average residuals using `extract_repeat_correction_summary()`. This can be helpful to pinpoint the sample(s) that need adjusting.

———— force_whole_repeat_units ————

The `force_whole_repeat_units` option aims to correct for the systematic underestimation in fragment sizes that occurs in capillary electrophoresis. It is independent to the algorithms described above and can be used in conjunction. It modifies repeat lengths in a way that helps align peaks with the underlying repeat pattern, making the repeat lengths whole units (rather than ~0.9 repeats). The calculated repeat lengths start from the main peak's repeat length and increases in increments of the specified `repeat_size` in either direction. This option basically enables you to get exactly the same result as expansion_index values calculated from data from Genemapper.

———— force_repeat_pattern ————

This parameter re-calls the peaks based on specified (`force_repeat_pattern_size_period`) periodicity of the peaks. The main application of this algorithm is to solve the issue of contaminating peaks in the expected regular pattern of peaks. We can use the periodicity to jump between peaks and crack open a window (`force_repeat_pattern_size_window`) to then pick out the tallest scan in the window.

Value

This function modifies list of fragments objects in place with repeats added.

See Also

`find_alleles()`, `add_metadata()`, `plot_batch_correction_samples()`, `plot_repeat_correction_model()`, `extract_repeat_correction_summary()`

Examples

```
fsa_list <- lapply(cell_line_fsa_list[c(16:19)], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

fragments_list <- find_fragments(
  fsa_list,
  min_bp_size = 300
)

find_alleles(fragments_list)

add_metadata(fragments_list,
   metadata[c(16:19), ]
)

# Simple conversion from bp size to repeat size
call_repeats(
  fragments_list,
  assay_size_without_repeat = 87,
  repeat_size = 3
```

```
)

plot_traces(fragments_list[1], xlim = c(120, 170))

# Use force_whole_repeat_units algorithm to make sure called
# repeats are the exact number of bp apart

call_repeats(
  fragments_list,
  force_whole_repeat_units = TRUE,
  assay_size_without_repeat = 87,
  repeat_size = 3
)

plot_traces(fragments_list[1], xlim = c(120, 170))


# apply batch correction
call_repeats(
  fragments_list,
  correction = "batch",
  assay_size_without_repeat = 87,
  repeat_size = 3
)

plot_traces(fragments_list[1], xlim = c(120, 170))

# apply repeat correction
call_repeats(
  fragments_list,
  correction = "repeat",
  assay_size_without_repeat = 87,
  repeat_size = 3
)

plot_traces(fragments_list[1], xlim = c(120, 170))

#ensure only periodic peaks are called
call_repeats(
  fragments_list,
  force_repeat_pattern = TRUE,
  force_repeat_pattern_size_period = 2.75,
  assay_size_without_repeat = 87,
  repeat_size = 3
)

plot_traces(fragments_list[1], xlim = c(120, 170))
```

---

cell_line_fsa_list *A list of fsa files*

Description

A list of fsa files read into R using trace::read_fsa() that for example data

Usage

```
cell_line_fsa_list
```

Format

`cell_line_fsa_list`:
A list with 92 elements, each one being the contents of an fsa file:

Source

doi:10.1038/s41467024474850

`example_data` *example_data*

Description

example_data is genemapper output peak table

Usage

```
example_data
```

Format

`example_data`:
A genemapper output dataframe

Source

doi:10.1038/s41467024474850

example_data_repeat_table *example_data_repeat_table*

Description

example_data_repeat_table is data with repeats called

Usage

```
example_data_repeat_table
```

Format

example_data_repeat_table:
A genemapper output dataframe

Source

doi:10.1038/s41467024474850

extract_alleles *Extract Modal Peaks*

Description

Extracts modal peak information from each sample in a list of fragments.

Usage

```
extract_alleles(fragments_list)
```

Arguments

fragments_list	A list of fragments_repeats objects containing fragment data.

Value

A dataframe containing modal peak information for each sample

Examples

```
gm_raw <- trace::example_data

test_fragments <- peak_table_to_fragments(gm_raw,
  data_format = "genemapper5",
  dye_channel = "B",
  min_size_bp = 400
)

find_alleles(
  fragments_list = test_fragments,
  peak_region_size_gap_threshold = 6,
  peak_region_signal_threshold_multiplier = 1
)

extract_alleles(test_fragments)
```

`extract_fragments` *Extract All Fragments*

Description

Extracts peak data from each sample in a list of fragments.

Usage

```
extract_fragments(fragments_list)
```

Arguments

`fragments_list` A list of fragments_repeats objects containing fragment data.

Value

A dataframe containing peak data for each sample

Examples

```
gm_raw <- trace::example_data
metadata <- trace::metadata

test_fragments <- peak_table_to_fragments(gm_raw,
  data_format = "genemapper5",
  dye_channel = "B",
  min_size_bp = 400
)

add_metadata(
```

```
  fragments_list = test_fragments,
  metadata_data.frame = metadata
)

find_alleles(
  fragments_list = test_fragments
)

call_repeats(
  fragments_list = test_fragments,
  assay_size_without_repeat = 87,
  repeat_size = 3
)

extract_alleles(test_fragments)
```

`extract_ladder_summary` *Extract ladder summary*

Description

Extract a table summarizing the ladder models

Usage

```
extract_ladder_summary(fragments_trace_list, sort = FALSE)
```

Arguments

`fragments_trace_list`	a list of fragments trace objects
`sort`	A logical statement for if the samples should be ordered by average ladder R-squared.

Details

The ladder peaks are assigned using a custom algorithm that maximizes the fit of detected ladder peaks and given base-pair sizes. This function summarizes the R-squared values of these individual correlations.

Value

a dataframe of ladder quality information

Examples

```
fsa_list <- lapply(cell_line_fsa_list, function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

extract_ladder_summary(fsa_list, sort = TRUE)
```

`extract_repeat_correction_summary` *Extract repeat correction summary*

Description

Extracts a table summarizing the model used to correct repeat length

Usage

```
extract_repeat_correction_summary(fragments_list)
```

Arguments

`fragments_list`	A list of fragments_repeats class objects obtained from the `call_repeats()` function when the `correction = "repeat"` parameter is used.

Details

For each of the samples used for repeat correction, this table pulls out the modal repeat length called by the model (`allele_repeat`), how far that sample is on average from the linear model in repeat units by finding the average residuals (`avg_residual`), and the absolute value of the `avg_residual` (`abs_avg_residual`)

Value

A data.frame

Examples

```
fsa_list <- lapply(cell_line_fsa_list[16:19], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

fragments_list <- find_fragments(fsa_list, min_bp_size = 300)

test_alleles <- find_alleles(
  fragments_list
)

add_metadata(
```

```
  fragments_list,
  metadata
)


call_repeats(
  fragments_list = fragments_list,
  correction = "repeat"
)

# finally extract repeat correction summary
extract_repeat_correction_summary(fragments_list)
```

`extract_trace_table` *Extract traces*

Description

Extract the raw trace from a list of fragments objects

Usage

```
extract_trace_table(fragments_trace_list)
```

Arguments

`fragments_trace_list`	a list of fragments objects

Value

A dataframe of the raw trace data. Each row representing a single scan.

Examples

```
fsa_list <- lapply(cell_line_fsa_list[1], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

extracted_traces <- extract_trace_table(fsa_list)
```

`find_alleles` *Find Alleles*

Description

This function identifies main allele within each fragment object.

Usage

```
find_alleles(
  fragments_list,
  number_of_alleles = 1,
  peak_region_size_gap_threshold = 6,
  peak_region_signal_threshold_multiplier = 1
)
```

Arguments

`fragments_list`	A list of fragment objects containing peak data.
`number_of_alleles`	Number of alleles to be returned for each fragment. Must either be 1 or 2. Being able to identify two alleles is for cases when you are analyze different human samples with a normal and expanded alleles and you can't do the preferred option of simply ignoring the normal allele in `find_fragments()` (eg setting the min_bp_size above the normal allele bp size).
`peak_region_size_gap_threshold`	Gap threshold for identifying peak regions. The peak_region_size_gap_threshold is a parameter used to determine the maximum allowed gap between peak sizes within a peak region. Adjusting this parameter affects the size range of peaks that can be grouped together in a region. A smaller value makes it more stringent, while a larger value groups peaks with greater size differences, leading to broader peak regions that may encompass wider size ranges.
`peak_region_signal_threshold_multiplier`	Multiplier for the peak signal threshold. The peak_region_signal_threshold_multiplier parameter allows adjusting the threshold for identifying peak regions based on peak signals. Increasing this multiplier value will result in higher thresholds, making it more stringent to consider peaks as part of a peak region. Conversely, reducing the multiplier value will make the criteria less strict, potentially leading to more peaks being grouped into peak regions. It's important to note that this parameter's optimal value depends on the characteristics of the data and the specific analysis goals. Choosing an appropriate value for this parameter can help in accurately identifying meaningful peak regions in the data.

Details

This function finds the main alleles for each fragment in the list by identifying clusters of peaks ("peak regions") with the highest signal intensities. This is based on the idea that PCR amplicons

of repeats have clusters of peaks (from somatic mosaicism and PCR artifacts) that help differentiate the main allele of interest from capillary electrophoresis noise/contamination.

If number_of_alleles = 1, the tallest of peaks will be selected as the allele. This means that if your sample has multiple alleles, you have two options i) make sure that your data is subsetted to only include the allele of interest (using `min_bp_size` in `find_fragments()` to make sure that the smaller allele is excluded), or ii) setting number_of_alleles = 2, which will pick the two tallest peaks in their respective peak regions and set the main allele as the larger repeat size, and allele_2 as the shorter repeat size. We recommend the subsetting approach since that is far simpler and less likely to fail, and the second option only if you're doing an experiment analysis a large number of human samples where both the normal and expanded allele repeat lengths vary, which makes it very difficult to find a common bp size that excludes the normal allele.

The parameters `peak_region_signal_threshold_multiplier` and `peak_region_size_gap_threshold` will only need to be adjusted in rare cases if peaks are not being found for some reason. They influence the criteria for identifying peak regions. peak_region_signal_threshold_multiplier is multiplied to the mean height of all the peaks to create a hight threshold for inclusion into the peak region, so most of the time it's already a very low value and probably only needs to be changed if you have very few peaks. peak_region_size_gap_threshold is the distance between the peaks, either bp size, or repeats if repeats have already been called.

Value

This function modifies list of fragments_repeats objects in place with alleles added.

Examples

```
fsa_list <- lapply(cell_line_fsa_list[1], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

fragments_list <- find_fragments(fsa_list,
  min_bp_size = 300
)


find_alleles(
  fragments_list,
  peak_region_size_gap_threshold = 6,
  peak_region_signal_threshold_multiplier = 1
)
```

`find_fragments` *Find fragment peaks*

Description

Find fragment peaks in continuous trace data and convert to fragments_repeats class.

Usage

```
find_fragments(
  fragments_trace_list,
  smoothing_window = 21,
  minimum_peak_signal = 20,
  min_bp_size = 100,
  max_bp_size = 1000,
  ...
)
```

Arguments

fragments_trace_list	A list of fragments_trace objects containing fragment data.
smoothing_window	numeric: signal smoothing window size passed to pracma::savgol()
minimum_peak_signal	numeric: minimum signal of the raw trace. To have no minimum signal set as "-Inf".
min_bp_size	numeric: minimum bp size of peaks to consider
max_bp_size	numeric: maximum bp size of peaks to consider
...	pass additional arguments to pracma::findpeaks(), or change the default arguments we set. minimum_peak_signal above is passed to pracma::findpeaks() as minpeakheight, and peakpat has been set to '[+]{6,}[0]*[-]{6,}' so that peaks with flat tops are still called, see https://stackoverflow.com/questions/47914035/identify-sustained-peaks-using-pracmafindpeaks

Details

`find_fragments()` takes in a list of fragments_trace objects and returns a list of new fragments_repeats objects.

This function is basically a wrapper around pracma::findpeaks. As mentioned above, the default arguments arguments of pracma::findpeaks can be changed by passing them to find_fragments with ... .

If too many and inappropriate peaks are being called, this may also be solved with the different repeat calling algorithms in `call_repeats()`.

Value

a list of fragments_repeats objects.

Examples

```
fsa_list <- lapply(cell_line_fsa_list[1], function(x) x$clone())

find_ladders(fsa_list)
```

```
fragments_list <- find_fragments(fsa_list,
  min_bp_size = 300
)


# Manually inspect the ladders
plot_traces(fragments_list,
  show_peaks = TRUE, n_facet_col = 1,
  xlim = c(400, 550), ylim = c(0, 1200)
)
```

`find_ladders` *Ladder and bp sizing*

Description

Find the ladder peaks in and use that to call bp size

Usage

```
find_ladders(
  fragments_trace,
  ladder_channel = "DATA.105",
  signal_channel = "DATA.1",
 ladder_sizes = c(50, 75, 100, 139, 150, 160, 200, 250, 300, 340, 350, 400, 450, 490,
    500),
  ladder_start_scan = NULL,
  minimum_peak_signal = NULL,
  scan_subset = NULL,
  ladder_selection_window = 5,
  max_combinations = 2500000,
  warning_rsq_threshold = 0.998,
  show_progress_bar = TRUE
)
```

Arguments

`fragments_trace`	list from 'read_fsa' function
`ladder_channel`	string: which channel in the fsa file contains the ladder signal
`signal_channel`	string: which channel in the fsa file contains the data signal
`ladder_sizes`	numeric vector: bp sizes of ladder used in fragment analysis. defaults to GeneScan™ 500 LIZ™
`ladder_start_scan`	numeric: indicate the scan number to start looking for ladder peaks. Usually this can be automatically found (when set to NULL) since there's a big spike right at the start. However, if your ladder peaks are taller than the big spike, you will need to set this starting scan number manually.

`minimum_peak_signal`
: numeric: minimum signal of peak from smoothed signal.

`scan_subset`
: numeric vector (length 2): filter the ladder and data signal between the selected scans (eg scan_subset = c(3000, 5000)). to pracma::savgol().

`ladder_selection_window`
: numeric: in the ladder assigning algorithm, the we iterate through the scans in blocks and test their linear fit (We can assume that the ladder is linear over a short distance) This value defines how large that block of peaks should be.

`max_combinations`
: numeric: what is the maximum number of ladder combinations that should be tested

`warning_rsq_threshold`
: The value for which this function will warn you when parts of the ladder have R-squared values below the specified threshold.

`show_progress_bar`
: show progress bar

Details

This function takes a list of fragments_trace files (the output from read_fsa) and identifies the ladders in the ladder channel which is used to call the bp size. The output is a list of fragments_traces.

In this package, base pair (bp) sizes are assigned using a generalized additive model (GAM) with cubic regression splines. The model is fit to known ladder fragment sizes and their corresponding scan positions, capturing the relationship between scan number and bp size. Once trained, the model predicts bp sizes for all scans by interpolating between the known ladder points. This approach provides a flexible and accurate assignment of bp sizes, accommodating the slightly non-linear relationship.

Use `plot_data_channels()` to plot the raw data on the fsa file to identify which channel the ladder and data are in.

The ladder peaks are assigned from largest to smallest. I would recommend excluding size standard peaks less than 50 bp (eg size standard 35 bp).

Each ladder should be manually inspected to make sure that is has been correctly assigned.

Value

This function modifies list of fragments_trace objects in place with the ladder assigned and base pair calculated.

See Also

`plot_data_channels()` to plot the raw data in all channels. `plot_ladders()` to plot the assigned ladder peaks onto the raw ladder signal. `fix_ladders_interactive()` to fix ladders with incorrectly assigned peaks.

Examples

```
fsa_list <- lapply(cell_line_fsa_list[1], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

# Manually inspect the ladders
plot_ladders(fsa_list[1])
```

`fix_ladders_interactive` *Fix ladders interactively*

Description

An app for fixing ladders

Usage

```
fix_ladders_interactive(fragment_trace_list)
```

Arguments

`fragment_trace_list`
: A list of fragments_trace objects containing fragment data

Details

This function helps you fix ladders that are incorrectly assigned. Run `fix_ladders_interactive()` and provide output from `find_ladders`. In the app, for each sample, click on line for the incorrect ladder size and drag it to the correct peak.

Once you are satisfied with the ladders for all the broken samples, click the download button to generate a file that has the ladder correction data. Read this file back into R using readRDS, then use `fix_ladders_manual()` and supply the ladder correction data as `ladder_df_list`. This allows the manually corrected data to be saved and used within a script so that the correct does not need to be done every time. An example of what you would need to do:

ladder_df_list <- readRDS('path/to/exported/data.rds') test_ladders_fixed <- fix_ladders_manual(test_ladders_broken, ladder_df_list)

Value

interactive shiny app for fixing ladders

See Also

`fix_ladders_manual()`, `find_ladders()`

Examples

```
fsa_list <- lapply(cell_line_fsa_list["20230413_A08.fsa"], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

# to create an example, lets brake one of the ladders
brake_ladder_list <-  list(
   "20230413_A08.fsa" = data.frame(
    size = c(35, 50, 75, 100, 139, 150, 160, 200, 250, 300, 340, 350, 400, 450, 490, 500),
    scan = c(1544, 1621, 1850, 1912, 2143, 2201, 2261, 2506, 2805, 3135, 3380, 3442, 3760,
              4050, 4284, 4332)
   )
 )

fix_ladders_manual(
  fsa_list,
  brake_ladder_list
)

plot_ladders(fsa_list)


if (interactive()) {
  fix_ladders_interactive(fsa_list)
}

# once you have corrected your ladders in the app,
# export the data for incorporation into the script.
# You can then re-import the data and fix ladders as described in the help details.
```

`fix_ladders_manual` *Fix ladders manually*

Description

Manually assign the ladder peaks for samples in a fragments_trace_list

Usage

```
fix_ladders_manual(
  fragments_trace_list,
  ladder_df_list,
  warning_rsq_threshold = 0.998
)
```

Arguments

`fragments_trace_list`
: list of fragments_trace objects

`ladder_df_list`
: a list of dataframes, with the names being the unique id and the value being a dataframe. The dataframe has two columns, size (indicating the bp of the standard) and scan (the scan value of the ladder peak). It's critical that the element name in the list is the unique id of the sample.

`warning_rsq_threshold`
: The value for which this function will warn you when parts of the ladder have R-squared values below the specified threshold.

Details

This function returns a fragments_trace list the same length as was supplied. It goes through each sample and either just returns the same fragments_trace if the unique id doesn't match the samples that need the ladder fixed, or if it is one to fix, it will use the supplied dataframe in the ladder_df_list as the ladder. It then reruns the bp sizing methods on those samples.

This is best used with `fix_ladders_interactive()` that can generate a `ladder_df_list`.

Value

This function modifies list of fragments_trace objects in place with the selected ladders fixed.

Examples

```
fsa_list <- lapply(cell_line_fsa_list[1], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

# first manually determine the real ladder peaks using your judgment
# the raw ladder signal can be extracted
raw_ladder <- fsa_list[1]$raw_ladder

# or we can look at the "trace_bp_df" to see a dataframe that includes the scan and ladder signal
raw_ladder_df <- fsa_list[[1]]$trace_bp_df[, c("unique_id", "scan", "ladder_signal")]
plot(raw_ladder_df$scan, raw_ladder_df$ladder_signal)

# once you have figured what sizes align with which peak, make a dataframe. The
# fix_ladders_manual() function takes a list as an input so that multiple ladders
# can be fixed. Each sample would have the the list element name as it's unique id.

example_list <- list(
 "20230413_A07.fsa" = data.frame(
   size = c(100, 139, 150, 160, 200, 250, 300, 340, 350, 400, 450, 490, 500),
   scan = c(1909, 2139, 2198, 2257, 2502, 2802, 3131, 3376, 3438, 3756, 4046, 4280, 4328)
 )
)

fix_ladders_manual(
  fsa_list,
```

```
  example_list
)
```

`fragments` *fragments object*

Description

An R6 Class representing a fragments object.

Details

This is the parent class of both fragments_trace and fragments_repeats object. The idea is that shared fields and methods are both inherited from this object, but it is not itself directly used.

Public fields

`unique_id` unique id of the sample usually the file name

`metrics_group_id` sample grouping for metrics calculations. Associated with `add_metadata()`.

`metrics_baseline_control` logical to indicate if sample is the baseline control. Associated with `add_metadata()`.

`batch_run_id` fragment analysis run. Associated with `add_metadata()`.

`batch_sample_id` An id for the sample used as size standard for repeat calculation. Associated with `add_metadata()`.

`batch_sample_modal_repeat` Validated repeat length for the modal repeat repeat in that sample. Associated with `add_metadata()`.

Methods

Public methods:

- `fragments$new()`
- `fragments$print()`
- `fragments$plot_trace()`
- `fragments$clone()`

Method `new()`**:** initialization function that is not used since the child classes are the main object of this package.

Usage:

```
fragments$new(unique_id)
```

Arguments:

`unique_id` unique_id

Method `print()`**:** A function to print informative information to the console

Usage:

```
fragments$print()
```

Method `plot_trace()`**:** plot the trace data

Usage:

```
fragments$plot_trace(
  show_peaks = TRUE,
  x_axis = NULL,
  ylim = NULL,
  xlim = NULL,
  signal_color_threshold = 0.05,
  plot_title = NULL
)
```

Arguments:

`show_peaks` A logical to say if the called peaks should be plotted on top of the trace. Only valid for fragments_repeats objects.

`x_axis` Either "size" or "repeats" to indicate what should be plotted on the x-axis.

`ylim` numeric vector length two specifying the y axis limits

`xlim` numeric vector length two specifying the x axis limits

`signal_color_threshold` A threshold value to colour the peaks relative to the tallest peak.

`plot_title` A character string for setting the plot title. Defaults to the unique id of the object

Returns: A base R plot

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
fragments$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`fragments_repeats` *fragments_repeats object*

Description

An R6 Class representing a fragments_repeats object.

Details

The idea behind this class is to store data for processing of the peak level data towards calculation of repeat instability metrics.

It contains important setters and getters for alleles and index peaks. It's very important that the exactly correct size and repeat value is set for the alleles and index peak. This is used for subsetting etc, so if it's not exactly correct many functions would break.

It also contains methods for plotting the ladder and traces (if available).

Super class

trace::fragments -> fragments_repeats

Public fields

trace_bp_df A dataframe of bp size for every scan from find_ladders().

peak_table_df A dataframe containing the fragment peak level information.

repeat_table_df A dataframe containing the fragment peak level information with the repeat size added. May or may not be the same as peak_table_df depending on what options are chosen in call_repeats.

Methods

Public methods:

- fragments_repeats$get_allele_peak()
- fragments_repeats$set_allele_peak()
- fragments_repeats$get_index_peak()
- fragments_repeats$set_index_peak()
- fragments_repeats$plot_fragments()
- fragments_repeats$clone()

Method get_allele_peak(): This returns a list with the allele information for this object.

Usage:

```
fragments_repeats$get_allele_peak()
```

Method set_allele_peak(): This sets a single allele size/repeat. It searches through the appropriate peak table and finds the closest peak to the value that's provided.

Usage:

```
fragments_repeats$set_allele_peak(allele, unit, value)
```

Arguments:

allele Either 1 or 2, indicating which allele information should be set. Allele 1 is the only one used for repeat instability metrics calculations.

unit Either "size" or "repeats" to indicate if the value you're providing is bp size or repeat length.

value Numeric vector (length one) of the size/repeat length to set.

Method get_index_peak(): This returns a list with the index peak information for this object.

Usage:

```
fragments_repeats$get_index_peak()
```

Method set_index_peak(): This sets the index repeat length. It searches through the repeat table and finds the closest peak to the value that's provided.

Usage:

```
fragments_repeats$set_index_peak(value)
```

Arguments:

`value` Numeric vector (length one) of the repeat length to set as index peak.

Method `plot_fragments()`**:** This plots the peak/repeat table as a histogram

Usage:

```
fragments_repeats$plot_fragments(ylim = NULL, xlim = NULL, plot_title = NULL)
```

Arguments:

`ylim` numeric vector length two specifying the y axis limits

`xlim` numeric vector length two specifying the x axis limits

`plot_title` A character string for setting the plot title. Defaults to the unique id of the object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
fragments_repeats$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`fragments_trace` *fragments_trace object*

Description

An R6 Class representing a fragments_trace object.

Details

The idea behind this class is to store data for processing of the continuous trace-level information from an fsa file towards peak level data.

It also contains methods for plotting the ladder and traces

Super class

`trace::fragments` -> `fragments_trace`

Public fields

`unique_id` unique id of the sample usually the file name

`fsa` The whole fsa file, output from seqinr::read.abif()

`raw_ladder` The raw data from the ladder channel

`raw_data` The raw data from the sample channel

`scan` The scan number

`off_scale_scans` vector indicating which scales were too big and off scale. Note can be in any channel

`ladder_df` A dataframe of the identified ladder from `find_ladders()`. Scan is the scan number of peak and size is the associated bp size.

`trace_bp_df` A dataframe of bp size for every scan from `find_ladders()`.

Methods

Public methods:

- `fragments_trace$new()`
- `fragments_trace$plot_ladder()`
- `fragments_trace$plot_data_channels()`
- `fragments_trace$clone()`

Method `new()`**:** Create a new fragments_trace.

Usage:

```
fragments_trace$new(unique_id, fsa_file)
```

Arguments:

`unique_id` usually the file name

`fsa_file` output from seqinr::read.abif()

Returns: A new `fragments_trace` object.

Method `plot_ladder()`**:** plot the ladder data

Usage:

```
fragments_trace$plot_ladder(xlim = NULL, ylim = NULL, plot_title = NULL)
```

Arguments:

`xlim` numeric vector length two specifying the x axis limits

`ylim` numeric vector length two specifying the y axis limits

`plot_title` A character string for setting the plot title. Defaults to the unique id of the object

Returns: A base R plot

Method `plot_data_channels()`**:** plot the raw data channels in the fsa file. It identifies every channel that has "DATA" in its name.

Usage:

```
fragments_trace$plot_data_channels()
```

Returns: A base R plot

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
fragments_trace$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`generate_trace_template` *Generate a Quarto file that has the instability pipeline preset*

Description

Generate a Quarto file that has the instability pipeline preset

Usage

```
generate_trace_template(
  file_name = NULL,
  correction = "batch",
  samples_grouped = TRUE
)
```

Arguments

`file_name`	Name of file to create
`correction`	select either "none", "batch" or "repeat" to indicate if the functionality for correcting repeat size using size standards across batches be included in the pipeline. See `add_metadata()` & `call_repeats()` for more info.
`samples_grouped`	Indicates if the functionality for grouping samples for metrics calculations should be included in the pipeline. See `add_metadata()` & `assign_index_peaks()` for more info.

Value

A Quarto template file for repeat instability analysis

Examples

```
if (interactive()) {
   generate_trace_template("test")
}
```

metadata *metadata*

Description

This is a dataframe containing the metadata information for the example data

Usage

```
metadata
```

Format

`metadata`:
A genemapper output dataframe

Source

doi:10.1038/s41467024474850

peak_table_to_fragments *Convert Peak Table to Fragments_repeats class*

Description

This function converts a peak table data frame into a list of fragments_repeats objects.

Usage

```
peak_table_to_fragments(
  df,
  data_format = NULL,
  peak_size_col = NULL,
  peak_signal_col = NULL,
  unique_id = NULL,
  dye_col = NULL,
  dye_channel = NULL,
  allele_col = NULL,
  min_size_bp = 200,
  max_size_bp = 1000
)
```

Arguments

df	A data frame containing the peak data.
data_format	The format that the data frame is in (for example, a genemapper peak table). Choose between: genemapper5, generic.
peak_size_col	A character string specifying column name giving the peak size.
peak_signal_col	A character string specifying column name giving the peak signal.
unique_id	A character string specifying column name giving the unique sample id (often the file name).
dye_col	Genemapper specific. A character string specifying column name indicating the dye channel.
dye_channel	Genemapper specific. A character string indicating the channel to extract data from. For example, 6-FAM is often "B".
allele_col	Genemapper specific. A character string specifying column name indicating the called alleles. This is often used when the peaks have been called in genemapper.
min_size_bp	Numeric value indicating the minimum size of the peak table to import.
max_size_bp	Numeric value indicating the maximum size of the peak table to import.

Details

This function takes a peak table data frame (eg. Genemapper output) and converts it into a list of fragment objects. The function supports different data formats and allows specifying column names for various attributes.

Value

A list of fragments_repeats objects.

See Also

repeat_table_to_repeats

Examples

```
gm_raw <- trace::example_data

test_fragments <- peak_table_to_fragments(
  gm_raw,
  data_format = "genemapper5",
  dye_channel = "B",
  min_size_bp = 400
)
```

plot_batch_correction_samples *Plot correction samples*

Description

Plot the overlapping traces of the batch control samples

Usage

```
plot_batch_correction_samples(fragments_list, selected_sample, xlim = NULL)
```

Arguments

fragments_list	A list of fragments_repeats objects containing fragment data. must have trace information.
selected_sample	A character vector of batch_sample_id for a subset of samples to plot. Or alternatively supply a number to select batch sample by position in alphabetical order.
xlim	the x limits of the plot. A numeric vector of length two.

Details

A plot of the raw signal by bp size or repeats for the batch correction samples.

When plotting the traces before repeat correction, we do not expect the samples to be closely overlapping due to run-to-run variation. After repeat correction, the traces should be basically overlapping.

These plots are made using base R plotting. Sometimes these fail to render in the viewing panes of IDEs (eg you get the error 'Error in `plot.new()`: figure margins too large)'. If this happens, try saving the plot as a pdf using traditional approaches (see grDevices::pdf).

Value

plot of batch corrected samples

See Also

`call_repeats()` for more info on batch correction.

Examples

```
fsa_list <- lapply(cell_line_fsa_list[16:19], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

fragments_list <- find_fragments(fsa_list, min_bp_size = 300)
```

```
test_alleles <- find_alleles(
  fragments_list
)

add_metadata(
  fragments_list,
  metadata
)


call_repeats(
  fragments_list = fragments_list,
  correction = "batch"
)

# traces of bp size shows traces at different sizes
plot_batch_correction_samples(
  fragments_list,
  selected_sample = "S-21-212", xlim = c(100, 120)
)
```

`plot_data_channels` *plot_data_channels*

Description

Plot the raw data from the fsa file

Usage

```
plot_data_channels(fragments_list, sample_subset = NULL, n_facet_col = 1)
```

Arguments

`fragments_list`	A list of fragments_trace objects.
`sample_subset`	A character vector of unique ids for a subset of samples to plot
`n_facet_col`	A numeric value indicating the number of columns for faceting in the plot.

Details

A plot of the raw data channels in the fsa file.

These plots are made using base R plotting. Sometimes these fail to render in the viewing panes of IDEs (eg you get the error 'Error in `plot.new()`: figure margins too large)'. If this happens, try saving the plot as a pdf using traditional approaches (see grDevices::pdf). To get it to render in the IDE pane, trying matching `n_facet_col` to the number of samples you're attempting to plot, or using `sample_subset` to limit it to a single sample.

Value

a plot of the raw data channels

Examples

```
plot_data_channels(cell_line_fsa_list[1])
```

`plot_fragments` *Plot Peak Data*

Description

Plots peak data from a list of fragments.

Usage

```
plot_fragments(
  fragments_list,
  n_facet_col = 1,
  sample_subset = NULL,
  xlim = NULL,
  ylim = NULL
)
```

Arguments

`fragments_list`	A list of fragments_repeats objects containing fragment data.
`n_facet_col`	A numeric value indicating the number of columns for faceting in the plot.
`sample_subset`	A character vector of unique ids for a subset of samples to plot
`xlim`	the x limits of the plot. A numeric vector of length two.
`ylim`	the y limits of the plot. A numeric vector of length two.

Value

A plot object displaying the peak data.

Examples

```
gm_raw <- trace::example_data

fragments_list <- peak_table_to_fragments(gm_raw,
  data_format = "genemapper5",
  dye_channel = "B",
  min_size_bp = 300
)
```

```
find_alleles(
  fragments_list
)

plot_fragments(fragments_list[1])
```

---

plot_ladders *Plot ladder*

Description

Plot the ladder signal

Usage

```
plot_ladders(
  fragments_trace_list,
  n_facet_col = 1,
  sample_subset = NULL,
  xlim = NULL,
  ylim = NULL
)
```

Arguments

fragments_trace_list	A list of fragments_trace objects containing fragment data.
n_facet_col	A numeric value indicating the number of columns for faceting in the plot.
sample_subset	A character vector of unique ids for a subset of samples to plot
xlim	the x limits of the plot. A numeric vector of length two.
ylim	the y limits of the plot. A numeric vector of length two.

Value

a plot of ladders

Examples

```
fsa_list <- lapply(cell_line_fsa_list[1], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

# Manually inspect the ladders
plot_ladders(fsa_list[1])
```

plot_repeat_correction_model
Plot Repeat Correction Model

Description

Plots the results of the repeat correction model for a list of fragments.

Usage

```
plot_repeat_correction_model(
  fragments_list,
  batch_run_id_subset = NULL,
  n_facet_col = 1
)
```

Arguments

fragments_list	A list of fragments_repeats class objects obtained from the call_repeats() function when the correction = "repeat" parameter is used.
batch_run_id_subset	A character vector for a subset of batch_sample_id to plot. Or alternatively supply a number to select batch sample by position in alphabetical order.
n_facet_col	A numeric value indicating the number of columns for faceting in the plot.

Details

This function makes plots for the model used to correct samples for each batch_run_id. The repeat correction algorithm assigns the user supplied repeat length to the modal peak of the sample, then pulls out a set of robust neighboring peaks to help get enough data to build an accurate linear model for the relationship between base-pair size and repeat length. So on this plot, each dot is an individual peak, with the colour indicating each sample, with the y-axis is the repeat length called from the user-supplied value in the metadata and the value assigned to each peak, with the x-axis showing the corresponding base-pair size.

Value

A base R graphic object displaying the repeat correction model results.

Examples

```
fsa_list <- lapply(cell_line_fsa_list[16:19], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

fragments_list <- find_fragments(fsa_list, min_bp_size = 300)
```

```
test_alleles <- find_alleles(
  fragments_list
)

add_metadata(
  fragments_list,
  metadata
)


call_repeats(
  fragments_list = fragments_list,
  correction = "repeat"
)

# traces of bp size shows traces at different sizes
plot_repeat_correction_model(
  fragments_list,
  batch_run_id_subset = "20230414"
)
```

---

plot_traces *Plot sample traces*

Description

Plot the raw trace data

Usage

```
plot_traces(
  fragments_list,
  show_peaks = TRUE,
  n_facet_col = 1,
  sample_subset = NULL,
  xlim = NULL,
  ylim = NULL,
  x_axis = NULL,
  signal_color_threshold = 0.05
)
```

Arguments

fragments_list	A list of fragments_repeats or fragments_trace objects containing fragment data.
show_peaks	If peak data are available, TRUE will plot the peaks on top of the trace as dots.

n_facet_col	A numeric value indicating the number of columns for faceting in the plot.
sample_subset	A character vector of unique ids for a subset of samples to plot
xlim	the x limits of the plot. A numeric vector of length two.
ylim	the y limits of the plot. A numeric vector of length two.
x_axis	A character indicating what should be plotted on the x-axis, chose between size or repeats. If neither is selected, an assumption is made based on if repeats have been called.
signal_color_threshold	Threshold relative to tallest peak to color the dots (blue above, purple below).

Details

A plot of the raw signal by bp size. Red vertical line indicates the scan was flagged as off-scale. This is in any channel, so use your best judgment to determine if it's from the sample or ladder channel.

If peaks are called, green is the tallest peak, blue is peaks above the signal threshold (default 5%), purple is below the signal threshold. If force_whole_repeat_units is used within call_repeats(), the called repeat will be connected to the peak in the trace with a horizontal dashed line.

The index peak will be plotted as a vertical dashed line when it has been set using assign_index_peaks().

Value

plot traces from fragments object

Examples

```
fsa_list <- lapply(cell_line_fsa_list[1], function(x) x$clone())

find_ladders(fsa_list, show_progress_bar = FALSE)

fragments_list <- find_fragments(fsa_list,
  min_bp_size = 300
)

find_alleles(
  fragments_list
)

# Simple conversion from bp size to repeat size
call_repeats(
  fragments_list
)

plot_traces(fragments_list, xlim = c(105, 150))
```

read_fsa *Read fsa file*

Description

Read fsa file into memory and create fragments_trace object

Usage

```
read_fsa(files)
```

Arguments

files	a chr vector of fsa file names. For example, return all the fsa files in a directory with 'list.files("example_directory/", full.names = TRUE, pattern = ".fsa")'.

Details

read_fsa is just a wrapper around `seqinr::read.abif()` that reads the fsa file into memory and stores it inside a fragments_trace object. That enables you to use the next function `find_ladders()`.

Value

A list of fragments_trace objects

See Also

`find_ladders()`, `plot_data_channels()`

Examples

```
fsa_file <- read_fsa(system.file("abif/2_FAC321_0000205983_B02_004.fsa", package = "seqinr"))
plot_data_channels(fsa_file)
```

remove_fragments *Remove Samples from List*

Description

A convenient function to remove specific samples from a list of fragments.

Usage

```
remove_fragments(fragments_list, samples_to_remove)
```

Arguments

fragments_list	A list of fragments_repeats objects containing fragment data.
samples_to_remove	A character vector containing the unique IDs of the samples to be removed.

Value

A modified list of fragments with the specified samples removed.

Examples

```
gm_raw <- trace::example_data
metadata <- trace::metadata

test_fragments <- peak_table_to_fragments(
  gm_raw,
  data_format = "genemapper5",
  dye_channel = "B",
  min_size_bp = 300
)

all_fragment_names <- names(test_fragments)

# pull out unique ids of samples to remove
samples_to_remove <- all_fragment_names[c(1, 5, 10)]

samples_removed <- remove_fragments(test_fragments, samples_to_remove)
```

---

repeat_table_to_repeats

Convert Repeat Table to Repeats Fragments

Description

This function converts a repeat table data frame into a list of fragments_repeats. class.

Usage

```
repeat_table_to_repeats(df, unique_id, repeat_col, frequency_col)
```

Arguments

df	A data frame containing the repeat data.
unique_id	A character string indicating the column name for unique identifiers.
repeat_col	A character string indicating the column name for the repeats.
frequency_col	A character string indicating the column name for the repeat frequencies.

Details

This function takes a repeat table data frame and converts it into a list of repeats fragments. The function allows specifying column names for repeats, frequencies, and unique identifiers.

Value

A list of fragments_repeats objects.

Examples

```
repeat_table <- trace::example_data_repeat_table
test_fragments <- repeat_table_to_repeats(
  repeat_table,
  repeat_col = "repeats",
  frequency_col = "height",
  unique_id = "unique_id"
)
```

Index