Package ‘Dimodal’

December 18, 2025

Version 1.0.0

Date 2025-11-20

Title Spacing Tests for Multi-Modality

Copyright Primordial Machine Vision Systems, Inc.

Maintainer Greg Kreider <support@primachvis.com>

Description Tests for modality of data using its spacing. The main approach evaluates features (peaks, flats) using a combination of parametric models and non-parametric tests, either after smoothing the spacing by a low-pass filter or by looking over larger intervals. The library can also use any changepoint detectors available to look for transitions between features in the spacing. The Suggested list of libraries is misnamed. It contains all supported changepoint detectors, and all are considered optional. A good minimal set would be the first three entries. Some of the suggestions may no longer be current on CRAN, with the last source tarball found in its archives or at the additional repository. These packages will be used if they are already installed on older installations, but will not be available to newer.

License BSD_3_clause + file LICENSE

NeedsCompilation yes

Depends statmod

Suggests changepoint.np, ICSS, jointseg, anomaly, astsa, bcp, breakfast, bwd, ccid, changepoint, cpm, cpss, ecp, mosum, ocp, otsad, Rbeast, strucchange

Additional_repositories https://www.primachvis.com/data/Rpkg

Author Greg Kreider [aut, cre],
Melissa O'Neill [cph],
Primordial Machine Vision Systems, Inc. [cph]

Repository CRAN

Date/Publication 2025-12-18 14:00:07 UTC

Contents

`Dimodal-package` *Detection of Multi-Modal Data Using Spacing*

Description

The **Dimodal** package uses the spacing of data, or difference between order statistics, to detect and locate modes or the transition between them. Consistent spacing with stable values appears within a mode, while it increases at the anti-modes. The package contains parametric and non-parametric tests of features detected within the spacing, and can use any changepoint detectors installed on the system as a separate check.

Main Interface

The package has three top-level commands.

- `Dimodal` runs the modality analysis. It supports print, summary, and plot methods.
- `Diopt` provides a persistent database of options controlling the feature detectors, tests, and display of results.
- `Ditrack` displays the position and probability of features as filter sizes change, to help in selecting the best size for analysis. It has a plot method to graphically show the results.

Feature Detectors

The package has five feature detectors. They work with any data, not just the spacing.

- `find.runs` identifies fuzzy runs, sequences of nearly-equal numeric values or of equal discrete values or symbols.
- `find.peaks` identifies local extrema, merging small minor peaks into larger.
- `find.flats` identifies flat or consistent stretches of values.
- `find.cpt` is a majority voting scheme using external changepoint algorithms to identify a common set of points where the behavior of data changes.
- `find.level.sections` is an inversion of the **modehunt** changepoint detector and can be added to the voting list.

Feature Tests

`Dimodal` includes three groups of tests to evaluate features.

- `Dipeak.test` and `Diflat.test` are parametric models of the peak and flat distributions after low-pass filtering. `Dipeak.critval` and `Diflat.critval` provide critical values of the peak (height) and flat (length) for a significance level.
- `Dinrun.test` and `Dirunlen.test` are runs-based tests (up, down, equal trends) performed on the signed difference of a signal. They include the Kaplansky-Riordan test of the number of runs and a Markov chain model for the longest run.
- `Dipermht.test` and `Diexcurht.test` are bootstrap tests simulating a feature from the actual data. They include a permutation test of runs and a general excursion test from the difference of a signal.

Data

The `kirkwood` dataset has the multi-modal distribution of asteroid orbital radii, where modes identify families of asteroids within the main belt and anti-modes the Kirkwood gaps cleared out by regular perturbation from Jupiter.

Classes

The return value of each command, feature detector, and test is given an S3 class that supports printing, summarizing, and perhaps plotting. Links can be found in the return value section of the command.

Utility Functions

The package includes several functions to help work with the results of the analysis.

- `midquantile` uses piecewise linear segments to convert quantiles of discrete or heavily quantized data back to data.
- `runs.as.rle` converts the `find.runs` result to the `"rle"` class.
- `select.peaks` returns just the local maxima from the extrema detected by `find.peaks`.

- center.diw shifts the indices of features in the interval spacing, normally located at the end of the interval, into the center, to align with the low-pass features and actual data.
- match.features uses distance and overlap criteria to identify features found in both the low-pass and interval spacing.
- shiftID.place moves the results of find.peaks, find.flats, and find.cpt to the original data grid and converts any indices into raw values using the midquantile approximation.

Author(s)

Greg Kreider.

The package compiles by default with the PCG random number generator, written by Melissa O'Neill, for sampling during the excursion tests.

The kirkwood dataset is taken from the Lowell Observatory asteroid ephemeris.

Dicpt *Class methods for Dicpt Objects*

Description

Common functions to handle "Dicpt" class results which store information about changepoints, both the individual library results and the majority vote for classifier fusion.

Usage

```
## S3 method for class 'Dicpt'
print(x, ...)
## S3 method for class 'Dicpt'
summary(object, ...)
## S3 method for class 'Dicpt'
mark(x, hist=NULL, opt=Diopt(), ...)
## S3 method for class 'Dicpt'
plot(x, opt=Diopt(), ...)
```

Arguments

x	an object of class "Dicpt"
object	an object of class "Dicpt"
hist	the histogram being marked or NULL for a low-pass or interval spacing graph
opt	local version of options for color scheme
...	extra arguments, ignored for all methods

Details

The "Dicpt" class stores several different items in a list. The analysis in find.cpt takes two steps, first running the spacing without any filtering through all changepoint detectors available on the system and then combining the results into a master list by majority vote over all libraries.

The plot needs a tall window. It shows the changepoints from each variant within a library as dots, labelled on the right side, and the roll-up for each detector as crosses, labelled on the left. The final changepoint list is in the first row at top, and vertical bands show the region within which library points match the final. The plot uses the "palette" option to set its colors.

Changepoints are marked by ticks along the upper edge of the current graph, using option color "colID.cpt" in the palette. Marking the histogram assumes that Dimodal has added the "x" column to the cpt data frame.

The print method gives a table with the changepoints, the spacing at that index and corresponding raw data value, and the number of votes. It also lists the libraries that take part in the voting. The summary gives just the position of the changepoint and the library list. These methods use option "digits" to control the number of significant digits in the table. If the option is set to zero then the usual options()$digits is used.

Value

The Dicpt list contains seven elements. Two describe the individual algorithms that have run and their results.

test	is a data frame with one row per library and detector combination. It contains four columns, "library" with the full changepoint library name, "detector" the function name and/or method and/or test statistic, "abbrev" an abbreviation of the detector used for plot labels, and "npt" the number of changepoints found. The number may be zero.
rawpts	is a list whose length is the same as the number of rows of test. Its elements are each a vector of points whose length matches the "npt" column, or NULL if the detector returned no points or if there were too many per the "maxfncpt" option or if the detector timed out or had some other error.

test will have 0 rows if no detectors were available, and rawpts will be an empty list. Summarizing the arguments for this first step, the Dicpt list element

rawparam	is a list with elements "cptlibs" the specification of which libraries to use, "maxfncpt" the maximum fraction of the data that can be identified as changepoints, "ncpt" the conversion of maxfncpt to points, "timeout" the run time in seconds allowed for each detector, and "nx" the number of data points. cptlibs, maxfncpt, and timeout come from Diopt options.

To describe the fusion of the detectors the Dicpt list adds four elements.

libpts	is a list of the changepoints found by combining the results of each detector within a library. It has a length equal to the number of libraries that were run, the unique entries of the first column of the test data frame.
voting	is a boolean vector with the same length as libpts and TRUE if that library's points were counted in the voting or FALSE if not.

cpt	is a data frame with three columns that gives the final changepoint list. It has one row per changepoint. Column "pos" is the index of the changepoint in the original data. "val" is the data at the changepoint. "nvote" gives the number of votes the point received.
cptparam	is a list with elements "qvote" the pair of quantile bounds on the library point count used to discard libraries with unusually few or many changepoints, "qpts" the pair translated into an absolute count, "sep" the minimum absolute separation in data points between changepoints, "fsep" the minimum separation as a fraction of the data length, "libsep" the separation while merging library variants, "near" the final minimum separation in data points, and "minvote" the minimum number of votes a point must receive to appear in the final result.

`Dimodal` adds one column to the `cpt` data frame. It shifts the changepoints to the original data grid, without the loss of the initial index.

x	the location of the peak in the original data, using the mid-distribution function

This can be done by calling the utility function `shiftID.place` on the find.cpt result. Dimodal also adds an attribute "`source`" with value "`Di`" to show the changepoints lie in the raw spacing.

The methods return the Dicpt object invisibly.

See Also

`find.cpt`, `Dimodal`, `Diopt`, `shiftID.place`

Examples

```
## Not run because detectors may not be available.
## Not run:  m <- Dimodal(faithful$eruptions, Diopt.local(analysis='cpt'))
## Not run:  m$cpt
## Not run:  summary(m$cpt)
## You want a tall plot window for this.
## Not run:  dev.new(width=4, height=12) ; plot(m$cpt)
```

Didata *Class methods for Didata Objects*

Description

Common functions to handle "`Didata`" class results, which store all data used in the modality analysis.

Usage

```
## S3 method for class 'Didata'
print(x, ...)
## S3 method for class 'Didata'
summary(object, ...)
```

Arguments

`x`	an object of class `"Didata"`
`object`	an object of class `"Didata"`
`...`	extra arguments, ignored for all methods

Details

Didata is a matrix that contains all data used in the modality analysis, with contents that depends on which analyses are run. The matrix has one column for each raw data point. Dimodal creates the Didata object internally; there is no public function to generate it. Row `"xmid"` is the interpolated raw value for each index, as calculated by `midquantile(data["xsort",], type=#)`, where the algorithm type is provided by the option `"data.midq"`. Row `"signed"` added by the interval spacing analysis is its signed difference, where differences within an order of magnitude of `.Machine.double.eps` are treated as ties to avoid numerical precision errors.

The print method shows the source of the data and any filters applied to the spacing. It then prints a table with the valid columns, range of values, and standard deviation of the original data and spacing. If low-pass and interval spacing were needed by `Dimodal`, then information about them is added to the table. The method uses option `"digits"` to control the number of significant digits for raw values. If it is set to zero then the standard `options()$digits` is used.

The summary method prints just the setup information.

Value

The basic data stored in the matrix contains the rows

`x`	the original data
`xsort`	sorted data
`xmid`	the mid-distribution from `midquantile`
`Di`	the spacing; index 1 is NA

This is enough to run the changepoint analysis.

If low-pass spacing is required, `Dimodal` adds the row

`lp`	spacing after low-pass filtering

where the leading and trailing columns are set to NA when part of the filter falls outside the data.

If interval spacing is needed, the matrix also contains rows

`Diw`	the interval spacing
`signed`	the signed difference of the interval spacing, with values -1, 0, +1 $

where the leading columns up to the width of the interval are set NA. The signed difference has an additional NA.

Several bits of metadata are stored as attributes with the matrix, in addition to any generated by R. The basic analysis provides

`data.name`	the source of the data in the call to `Dimodal`

Low-pass filtering adds

lp.kernel	the name of the filter
lp.window	the window or size of the filter, as provided by Diopt, which is normally a fraction of the data size but may also be in data points
wlp	the actual window width in data points
lp.stID	the first column in the matrix with valid low-pass values
lp.endID	the last column (inclusive) that is valid

Interval spacing adds

diw.window	the interval size, as provided by Diopt, which is normally a fraction of the data size but may also be in data points
wdiw	the interval in data points
diw.stID	the first column in the matrix with valid interval spacing; the last valid column is the end of the matrix

Note that the signed row starts one column after the interval spacing, at diw.stID + 1, and runs to the end of the matrix.

The methods return the object, invisibly.

See Also

Dimodal, Diopt, midquantile

Examples

```
## We override the analysis option to avoid changepoints,
## which may not be available.
m <- Dimodal(faithful$eruptions, Diopt.local(analysis=c('diw','lp'), diw.window=16))
m$data
```

Diflat   *Class methods for Diflat Objects*

Description

Common functions to handle "Diflat" class results, which store information about local flats and their probability.

Usage

```
## S3 method for class 'Diflat'
print(x, ...)
## S3 method for class 'Diflat'
summary(object, ...)
## S3 method for class 'Diflat'
mark(x, hist=NULL, opt=Diopt(), ...)
```

Arguments

`x`	an object of class `"Diflat"`
`object`	an object of class `"Diflat"`
`hist`	the histogram being marked or NULL for a low-pass or interval spacing graph
`opt`	local version of options for color scheme
`...`	extra arguments, ignored for all methods

Details

A `"Diflat"` object is a data frame that describes local flats, including their position or span in the low-pass or interval spacing and the equivalent raw values, their length, and any tests that have run to judge the significance of the feature. The basic information comes from the `find.flats` detector. The analysis in `Dimodal` moves the positions to the original data and adds the test results. The class inherits from `"data.frame"`.

The print method shows the endpoints (incl.) of the flat and the raw value at each end, the statistics (test values) and separately the probability from each. The acceptance level of the tests are also given underneath. These come from the options `"alpha.len"`, `"alpha.ftexcur.lp"`, and `"alpha.ftexcur.diw"`. If the option `"mark.alpha"` is TRUE then probabilities at or below the acceptance level are underlined. Set the option FALSE if your terminal does not support ANSI escape codes, or see the **crayon** package for the extensive tests needed to determine this automatically. These options are set at the global level and cannot be overridden for a single call to print; Diopt is called internally. The pass column after the overall flat probability will contain the number of accepted/passing tests, if any. Raw values and statistics are printed with option `"digits"` significant digits. If the option is set to zero then the default `options()$digits` is used.

The summary method gives the endpoints, the overall probability or best result from all tests, and a list of the tests that pass per the `Diopt` acceptance levels.

Flats are marked in a spacing graph according to the `"mark.flat"` option, either as a box around the spacing or as a bar above or below it. The border or line is thicker if the flat passes any test at the option acceptance levels. When annotating a histogram the flats are always drawn as bars at the top of the graph. The marks are colored per the option `"colID.flat"` index of the current palette. Marking the histogram assumes that Dimodal has added the `xst` and `xend` columns.

Value

The basic data frame returned from `find.flats` contains the columns

`src`	the data point that triggered the flat, whose value is nearly centered in the height. This is a unique identifier for the feature.
`stID`	first data point in the flat
`endID`	last data point (incl.) in the flat
`len`	the length of the flat, or endID - stID + 1
`srcval`	the data value at src
`ht`	the height of the flat
`htsd`	the flat height standardized by the data's standard deviation, or ht / sd(x)

There is one row per flat, and the result is an empty data frame with zero rows if there are no flats. The indices of stID and endID are located in the data passed to the detector, which excludes the filter's NA columns.

`Dimodal` adds several columns. It shifts all indices to the original data, adjusting for the filter or interval size. Mapping indices back to the raw data, it adds

`xst`	the location of the start point in the original data
`xend`	the location of the end point (incl.) in the original data

This can be done by calling the utility function `shiftID.place` on the `find.flats` result. It also adds an attribute `"source"` with value "LP" or "Diw" depending on the filter used. As a summary of the results it adds

`pflat`	the best probability of all tests run, NA if no test results are available
`naccept`	number of tests passing their acceptance level, 0 if none or pflat is NA

For low-pass spacing the test results may include

`plen`	the probability of the feature per the length model
`hexcur`	the flat height used for the excursion test
`pexcur`	fraction of bootstrap samples from the low-pass spacing generating smaller heights than hexcur

depending on what has been run.

The interval spacing omits the length model test and may perform the excursion test on the difference of the interval spacing.

`hexcur`	the feature height used for the excursion test
`pexcur`	fraction of bootstrap samples from the interval spacing that generate smaller heights than hexcur

The methods return the object, invisibly.

See Also

`find.flats`, `Dimodal`, `Diopt`, `shiftID.place`, `data.frame`

Examples

```
## We override the analysis option to avoid changepoints,
## which may not be available.
m <- Dimodal(faithful$eruptions, Diopt.local(analysis=c('diw','lp'), diw.window=16))
m$lp.flats
summary(m$diw.flats)
```

`Dimodal` *Detect modality in the spacing of data.*

Description

Dimodal studies the modality of data using its spacing. The presence of peaks or local increases in it indicates the data is multi-modal and locates the anti-modes. Flats or consistent spacing cover the modes. **Dimodal** finds these features after smoothing the spacing by low-pass filtering, which supports discrete or heavily quantized data, or in the interval spacing. Several tests, using parametric models, runs, and bootstrap sampling, evaluate these features. The package also can use changepoint detectors to find changes in the spacing's behavior from multi-modality, without being able to locate the modes.

Usage

```
Dimodal(x, opt=Diopt())
## S3 method for class 'Dimodal'
print(x, feature=c('peaks', 'flats', 'cpt'), ...)
## S3 method for class 'Dimodal'
summary(object, feature=c('peaks', 'flats', 'cpt'), ...)
## S3 method for class 'Dimodal'
plot(x, show=c('lp', 'histogram', 'diw'),
     feature=c('peaks', 'flats', 'cpt'), opt=Diopt(), ...)
```

Arguments

`x`	for `Dimodal` the (numeric) data vector to analyze; for the methods an object of class `"Dimodal"`
`object`	an object of class `"Dimodal"`
`opt`	local version of options to guide analysis
`feature`	display only the indicated feature(s) in all methods that were run, or for plots mark only them in the graph
`show`	plot the low-pass spacing, a histogram of the raw data, and/or the interval spacing, in separate graphs in the order given
`...`	extra arguments, ignored for all methods

Details

Changes in the spacing of data can indicate a change in its modality, and `Dimodal` is a general interface to feature detectors and tests to evaluate such changes. Spacing, the difference between consecutive order statistics or the delta after sorting the data, takes on a 'U' form, increasing rapidly in the tails and remaining stable in the center (for single-sided variates it forms half the U; uniform variates have constant spacing). The transition between modes is marked by local increases in the spacing while the center of modes see stable values. Dimodal therefore looks for local maxima or peaks in the spacing, or locally flat regions.

The spacing, designated `Di`, is often very noisy, and may be quantized to a few values if the data is discrete or taken with limited precision. Smoothing is necessary, which Dimodal can do either by apply a low-pass (`lp`) filter or by taking the difference over more than one order statistic. The latter is called the interval spacing `Diw` and is generated as a difference with lag; it is equivalent to a running mean or rectangular filter of the raw spacing. The recommended low-pass filter is a Kaiser kernel, which offers good high-frequency suppression and main lobe width; other available filters are the Bartlett or triangular (synonyms), Hanning, Hamming, Gaussian or normal (synonyms), and Blackman. Filtering is done by convolving the data with the filter's kernel, rather than moving to the Fourier domain. Points at the start and finish that are partially covered by the kernel or interval are set to NA and attributes attached to the data give the valid range. Indexing from the two spacings is different. The low-pass kernel is centered, with partial overlaps at both ends. The interval spacing is defined as trailing from the upper index, which runs to the end of the data, so the partial overlap occurs only at the start. This will be seen in the position of the smoothed curves when plotting results and the shift in indices needed to align the two schemes will be printed with the data summary. The raw values corresponding to a feature automatically compensate for the difference.

The feature detectors `find.peaks` and `find.flats` have separate help pages describing their algorithms and the parameters that control their analysis. These features are local and therefore not only indicate whether data may be multi-modal, but provide the location of the modes and the transitions between them.

Dimodal uses three main strategies to evaluate the features. First, the models tests are `Dipeak.test` and `Diflat.test`, with critical values at a significance level also available. These models are based on simulations of the peak heights and flat lengths in a univariate null distribution and offer a parametric assessment of their significance. They are less conservative than other modality detectors. Second, the bootstrap test is `Diexcurht.test`. The bootstrap simulates the features drawing from a pool of the difference of the spacing, estimating their probability without assuming any underlying distribution. Finally, the runs tests are `Dinrun.test`, `Dirunlen.test`, and `Dipermht.test`. Quantizing the filtered spacing into a few levels by taking the sign of the difference (in other words, if the signal is increasing, decreasing, or constant) allows us to consider runs in the symbols. We can test how many there are, or the longest, or if a permutation of them recreates the feature.

The bootstrap test extends a peak to its support, defined by the `"peak.fhsupp"` option, a fraction of the peak's height. A value of 0.9 is enough to back the away from minima placed in a long flat while not distorting the peak's width if the minima are well-defined. 0.5 corresponds to Full Width at Half Maximum (FWHM), and 1.0 extends the peak to the minima.

A third option for analyzing the spacing is to look for changepoints in it. Changepoints or anomalies or outliers respond to the transition between features, for example the edges of flats or the increase in spacing towards but not at a peak, and are not as good at locating the modes. Dimodal itself does not do changepoint detection. Rather, it collects the results from other libraries and uses majority voting to decide on a final list. `find.cpt` documents the process, which occurs in two steps, first calling whichever detectors are installed on the system and then merging their results. As part of this project we investigated inverting the post-processing of the **modehunt** detector to find level sections instead of sloping. This is available as an additional changepoint algorithm to take into the voting. The changepoint analysis is entirely optional and Dimodal will run only the detectors it finds installed on your system. If none, then leave off "cpt" from the `"analysis"` options.

The analysis of each feature is gathered into separate S3 class objects which support printing and marking plots; the changepoint results can be plotted separately to show the voting. The generic functions on the Dimodal result route to these objects if they are selected by the `features` argument.

A plot may contain the filtered spacing or interval spacing plus a histogram of the raw data, with features annotated on each. It uses layout to create a row of the shown graphs, as specified by the `show` argument. The histogram annotations will come from the first, leftmost, spacing shown.

The raw data must be numeric or integer. Non-finite values, including NA, will be dropped.

Dimodal needs a complete list of options for the `opt` argument. Do not make changes in the call, as `Diopt` will return only the changed values. Use `Diopt.local` instead.

The option `"analysis"` controls which smoothed spacing to generate, one or more of 'lp', 'diw', and 'cpt'. If none of these are specified the data will contain only the spacing and mid-quantile function, without any features or their analysis.

Dimodal uses options `"lp.param"` and `"diw.param"` to override the detector options for each method, and `"lp.tests"` and `"diw.tests"` to determine which feature tests to carry out. If these are empty lists then the data will contain the smoothed spacing but there will be no features. While generating the data it uses options `"lp.kernel"` and `"lp.window"` to set up the low-pass filter, and `"diw.window"` for the interval width. It uses `"excur.ntop"` when creating the base set of draws for excursion tests. Option `"data.midq"` determines the approximation method (`type` argument to the `midquantile` function), when converting indices in the spacing back to order statistics.

The default values of the detector options come from the development of the low-pass models. We do not know how different values will affect the models. The interval spacing is much rougher than low-pass filtering, which may require looser ripple and height parameters to find any flat, or reduce the number of peaks. The excursion tests will accommodate this.

Value

A list assigned to class `"Dimodal"` with elements

`data`	an object of class `"Didata"` with all data used in the analysis
`lp.peaks`	an object of class `"Dipeak"` capturing the local extrema in the low-pass spacing and their evaluation, with test results and raw data locations added to the features from `find.peaks`
`lp.flats`	an object of class `"Diflat"` capturing the local flats in the low-pass spacing and their evaluation, with test results and raw data locations added to the features from `find.flats`
`diw.peaks`	an object of class `"Dipeak"` containing the local extrema in the interval spacing and their evaluation, with test results and raw data locations added to the features from `find.peaks`
`diw.flats`	an object of class `"Diflat"` capturing the local flats in the interval spacing and their evaluation, with test results and raw data locations added to the features from `find.flats`
`cpt`	an object of class `"Dicpt"` with the per-library changepoints in the spacing and their combination into a master list
`opt`	the list passed as the opt argument, per `Diopt`

These elements will have empty data structures if the analysis is not run.

Dimodal will automatically call `shiftID.place` on each detector's results and will summarize the tests, as described with each data class. Dimodal adds an attribute `"source"` to each of the features, with value LP, Diw, or Di.

See Also

Diopt for the parameters controlling the analysis.

find.peaks, find.flats for feature detection.

Dipeak.test, Diflat.test for parametric models to evaluate the features, Diexcurht.test for a bootstrap test of feature significance, Dinrun.test, Dirunlen.test for tests of runs (here for sequences in the sign of the difference in the interval spacing), and Dipermht.test for a permutation test of the runs making a feature.

find.cpt for changepoint detection in the spacing, without filtering. find.level.sections is a modification of the **modehunt** changepoint detector that can be added to the voting.

Didata, Dipeak, Diflat, Dicpt for the data structures generated by the feature detectors and their evaluation.

center.diw to further shift the position of interval spacing features to the middle of the interval to align with low-pass features.

match.features to identify common peaks and flats in both spacings.

shiftID.place to move indices in either spacing to the original data grid and add the corresponding raw values.

midquantile for the mid-quantile mapping from index to raw data.

Examples

```
## Not running cpt analysis because local setup is not known.
## The interval spacing is noisy with the default options, so require a
## larger peak height with a temporary value to Diopt.
oldopt <- Diopt(analysis=c('lp','diw'), diw.param=list(peak.fht=0.125))
## Run the analysis.
m <- Dimodal(faithful$waiting)
## If printing the results, the interval spacing peaks have a probability
## just under 0.05 but fail the acceptance levels.
summary(m)
## Details about the peaks in both spacings.
print(m, feature="peaks")
## We find one peak in both spacings, but only the low-pass is significant.
match.features(m)
## Three plots side by side.  The limited resolution of the data is clear
## in the interval spacing.
dev.new(width=12, height=4) ; plot(m)
## Restore the old option values.  Diopt(NULL) returns to defaults.
oldopt <- Diopt(oldopt)
```

Dimodal Excursion and Permutation Tests

Feature significance test using permutation or bootstrap (excursion) sampling.

Description

Repeatedly draw samples from a base set of differential values, with or without replacement, and regenerate the signal, measuring its height or range. Compare the feature height against this distribution to estimate its probability.

Usage

```
Diexcurht.test(ht, ndraw, xbase, nexcur, is.peak, lower.tail=TRUE, seed=0)
Dipermht.test(ht, xbase, nperm, lower.tail=TRUE, seed=0)
```

Arguments

`ht`	one or more feature heights
`ndraw`	the size of the feature, a vector as long as ht or a single value
`xbase`	a vector of values to draw from, differential values that can be summed to reconstruct the signal
`nexcur`	the number of times to repeat the draw
`nperm`	the number of times to permute the draw
`is.peak`	a boolean, TRUE to calculate height for peaks, FALSE for flats
`lower.tail`	a boolean, TRUE to count the number of simulated heights greater than ht, FALSE the number less than or equal to
`seed`	if positive, value to set the RNG seed before any sampling

Details

The excursion test determines the distribution of a feature's height by sampling from a set of differential data and doing a cumulative sum. The reconstructed feature height for a peak is the maximum sum above the lower of the first or last point, or the total range for flats, using `is.peak` to choose. As a bootstrap excursion this can be used with the difference of the low-pass or interval spacing for either peaks or flats. The resolution of the probability returned from the test is `1/nexcur`.

The permutation test shuffles the values in `xbase` before re-summing them to simulate the signal. The test automatically uses the peak height definition. The base set should be the length of the runs within the feature, positive or negative or zero (ignoring length). If generated using `rle`, multiply `$lengths` by `$values`. Permutations are restricted so that adjacent values do not have the same sign, because this would form a longer run. This eliminates a large fraction of the possible permutations, 90% or more. If 5% of the possible number of permutations, the factorial of the length of `xbase`, is less than the requested sample size, the test will exhaustively check all possibilities rather than sampling. For nperm=5000 this happens at eight runs.

These tests are general and do not depend on the type of feature or spacing being analyzed. The caller, in this library the `Dimodal` function, is responsible for identifying features and their height and setting up the base set and draw size. NA or NaN heights will generate NA probabilities, as do 0 heights and draw sizes that are too small (minimum 3 for excursions, 2 for permutations). All base values must be finite. Bad argument values will raise errors.

If more than one height is provided the tests will evaluate them against one set of draws, unless the draw sizes also vary, in which case pairs of the arguments will be used.

A FALSE `lower.tail` is appropriate for peaks, TRUE for flats. The probability can have a large uncertainty if the tests generate tied values matching the height. This is mostly seen with the runs permutations, which generate discrete heights. The count uses half the ties, which is a simple form of mid-distribution correction.

The RNG seed, if not zero, is added to the draw counter and used to set up the random number generator before sampling, so that the results are repeatable. A value of zero leaves the setup alone.

The `nexcur` argument corresponds to option `"excur.nrep"` and `seed` to `"excur.seed"`. Dimodal uses `"excur.ntop"` when generating xbase for excursions. The equivalents for the permutation test are `"perm.nrep"` and `"perm.seed"`. The default number of excursions is higher than for permutations, based on an evaluation of the stability on artificial and real data. The distribution of the excursion heights is slow to settle, and has a large confidence interval; by 15,000 repetitions the median height has settled to within half a percent and the 90% confidence interval is two percent of that median. Fewer repetitions may suffice, although below 5,000 you may notice the variation in the test probabilities. The permutation test is more stable and fewer repetitions are needed, so that count has been given its own option. Internally Dimodal will adjust the excursion seed differently for each of the peak and flat tests, so that the results will be repeatable but different sequences will be used for each. The probabilities will be evaluated against `"alpha.pkexcur.[lp|diw]"`, `"alpha.runht"`, and `"alpha.ftexcur.[lp|diw]"`.

Each draw runs in `O(ndraw)` time and memory, where ndraw is the size of the xbase set for the permutations. The permutation test takes noticeably longer with more than two symbols (ie. if there are ties) because a different sampling strategy is needed to guarantee the non-adjacency condition. Unless the package has been compiled to use the R sampling function we use the PCG random number generator from Melissa O'Neill for the excursion draws because it is much faster than the built-in sampling while offering greater bit depth.

Value

Both functions return a list of class `"Ditest"` with elements

`method`	a string describing the test
`statfn`	function used to evaluate significance level/probability
`statistic`	what is tested, the height of the feature
`statname`	text string describing the statistic
`parameter`	other arguments provided to the function, named as such
`p.value`	probability of feature
`alternative`	a string describing the direction of the test vs. the null distribution
`critval`	critical values of the height, at quantiles 0.005, 0.01, 0.05, and 0.10, or 1-these if lower.tail is FALSE
`xbase`	a copy of the argument

`statistic` and `p.value` will have the length of the `ht` argument. NA or 0 heights or draws will generate NA for the probabilities.

References

https://pcg-random.org for the PCG random number generator.

See Also

Dimodal, Diopt, rle, .Random.seed

Examples

```
## Recommended number of excursions/permutations is larger.
## 2000 reduces the run-time.
set.seed(2) ; x <- round(runif(50,-1,1) * 10) / 10
Diexcurht.test(6.5, 30, x, 2000, TRUE, lower.tail=FALSE, seed=3)
xrun <- rle(sign(diff(x)))$values * rle(sign(diff(x)))$lengths
Dipermht.test(6.5, xrun, 2000, FALSE, 3)
```

`Dimodal Model Tests` *Significance models of features in the low-pass spacing.*

Description

Return the probability of the characteristic feature value, the height of a peak or length of a flat, using parametric models developed for the low-pass spacing, or determine the feature value at some significance level.

Usage

```
Dipeak.test(ht, n, flp, filter, lower.tail=TRUE)
Dipeak.critval(pval, n, flp, filter)
Diflat.test(len, n, flp, filter, basedist, lower.tail=TRUE)
Diflat.critval(pval, n, flp, filter, basedist)
```

Arguments

`ht`	difference(s) between the standardized data value at the peak and deepest minimum to either side
`len`	length(s) of flat in data points
`pval`	the significance level(s) to find the corresponding height or length, the quantile of the feature value is `1-pval`
`n`	number of data points before filtering
`flp`	the size of the FIR kernel, either as a fraction of `n` or as an integer
`filter`	the FIR kernel used to smooth the spacing
`basedist`	for the flat models, the distribution used to generate the length quantiles
`lower.tail`	a boolean, if TRUE the test returns the probability the null distribution is less than or equal to the feature value, if FALSE greater than

Details

The test functions convert the feature value into a quantile or significance level based on null distribution models. The critval functions do the opposite. The models are parametric because they are built on draws of specifically chosen variates and the size of features that appear after low-pass filtering the data. The features depend on the size of the draw `n` and the smoothing done, set by the Finite Impulse Response (FIR) `filter` and the size `flp` of the kernel. Implicitly they depend on the feature detectors, but variations in the parameters controlling those have neither been studied nor incorporated in the model.

The peak height model comes from draws of an asymmetric Weibull variate with scale 2 and shape 4, which proved to give reasonable, conservative quantiles against other distributions. The preferred filter uses a Kaiser kernel. The other filters available, the Bartlett or triangular (synonyms), Hanning, Hamming, Gaussian or normal (synonyms), and Blackman kernel, are handled by scaling the Kaiser model. The filter size is typically expressed as a fraction of the draw size, with `flp=0.15` a good default; spans in data points are also accepted. Smaller kernels will produce rougher data with more peaks and fewer flats and can be tolerated if the spacing is already smooth, as happens with very large data sets. The test height for the model is scaled by the standard deviation of the total signal.

The peak test models the distribution of heights with an inverse Gaussian, a.k.a. Wald distribution. The height is corrected for the filter and its size, and the inverse Gaussian location and scale parameters depend on the data and filter sizes. These values are provided in the returned list.

The flat length model varies much more with the parametric distribution chosen as the base, and the recommended `basedist`, a logistic variate, is a compromise. Models for normal or Gaussian (synonyms), Gumbel, and Weibull distributions are also available, but there is little overlap between the quantiles of lengths within them; the logistic falls in the middle. The Weibull variant is more liberal, accepting lengths that are two-thirds those needed to pass at the same level as the logistic. The Gumbel lengths are four-thirds longer. The filter type, size, and draw size are the same as for the peak height model. Unlike the peak model, different filters require different models internally.

The length distribution varies smoothly with the data size and filter, and the flat model can calculate the probability directly without going through a distribution function.

The models come from simulations over the ranges `n = 50 ... 500` and `flp = 0.05 ... 0.5`, measuring quantiles between `q = 0.90 ... 0.99999`. They fit the critical values within 5% over most of these values, degrading to 10% at the edges. The spread in the reported probability also increases at the edges of the parameter space. In particular, data sets of less than 60 points or windows larger than 30% are less trustworthy, as are quantiles beyond 0.9999. The models will generate a warning under these conditions and a tighter significance level should be used to judged the results. For data sizes much beyond 500, it is better to switch to the normal or Weibull base distribution when testing flats.

Bad values passed for the draw and LP kernel sizes will raise errors. The filter name will default to Kaiser if the argument does not match a supported kernel or if it is a bad value (NA, empty, or non-character). The base distribution similarly defaults to the logistic. The arguments correspond to options `"lp.kernel"`, `"lp.window"` or `"diw.window"`, and `"flat.distrib"`. The probabilities should be evaluated against `"alpha.ht"` and `"alpha.len"` for the minimum passing level.

All four functions can take vectors as their first argument, which are evaluated one by one for the given filter and draw set-up.

Value

`Dipeak.test` and `Diflat.test` return lists of class `"Ditest"` with elements

`method`	a string describing the test
`statfn`	function used to evaluate significance level/probability
`statistic`	what is tested, the height of the peak or length of the flat
`statname`	text string describing the statistic
`parameter`	distributional arguments, for the peak the corrected height `corrht` and the `mu` and `lambda` for the inverse Gaussian; omitted for flats
`p.value`	probability of feature
`alternative`	a string describing the direction of the test vs. the null distribution
`model`	parameters for the feature model, `n` the data size, `flp` the low-pass filter size as a fraction of `n`, `filter` the low-pass kernel, and for flats `basedist` the distribution used to build the model

`statistic` and `p.value` will have the length of the `ht` or `len` argument. NA and NaN values in the first argument will propagate to `p.value`, NULL produces an empty vector, and non-numeric values an NA. If `pval` is less than 0 or greater than 1 the `p.value` is NaN.

See Also

`Dimodal`, `Diopt`, `find.peaks`, `find.flats`

Examples

```
pval <- Dipeak.test(0.25*(1:16), 200, 0.15,'kaiser', lower.tail=FALSE)
pval$p.value
## Recovers pval.
Dipeak.critval(pval$p.value, 200, 0.15,'kaiser')

pval <- Diflat.test(10*(1:12), 200, 0.15,'kaiser', 'logistic', lower.tail=FALSE)
pval$p.value
Diflat.critval(pval$p.value, 200, 0.15,'kaiser', 'logistic')
```

`Dimodal Runs Tests` *Tests of runs within subsets of data*

Description

Check the significance of a subsequence of data based on the number of runs or the length of the longest. These tests work on any data with a limited number of symbols. They are used within the Dimodal package to test peaks in the signed difference of the interval spacing, which has values -1, 0, and +1.

Usage

```
Dinrun.test(x, stID, endID, feps, lower.tail=TRUE)
Dirunlen.test(x, stID, endID, feps)
```

Arguments

`x`	a vector with a limited number of distinct values, including numeric, integer, character, factor, or logical
`stID`	a scalar or vector of data indices of the start indices within x of the features
`endID`	a scalar or vector of data indices of the end indices (incl.) of the features
`feps`	closeness of tied real values, as per `find.runs`, and ignored for other types
`lower.tail`	a boolean, if TRUE the test returns the probability that the run count is not more than the observed, if FALSE is more than

Details

`Dinrun.test` compares the number of runs within each sequence defined by the endpoints to the expected, based on a combinatorial counting of all possible sequences of the symbols. The number of runs is distributed normally, with an expected value and variance that depends on the number of each symbol. Wolf and Wolfowitz derived formulas for these values in the case of two symbols, and Kaplansky and Riordan generalized this to arbitrary sets of symbols. This test implements that general version. Its value is the probability of getting the actual number of runs or fewer, i.e. the lower tail.

Filtering introduces correlation between symbols, which we can account for by using a Markov chain model. The length of a run can be estimated by separating the symbol transition matrix into two parts, the diagonal which generates a matching symbol and the off-diagonal elements which switch to another. A new Markov chain modeling a run uses these two sub-matrices with the advancing steps placed on the diagonal of the run's transition matrix and the resetting steps in the first column. An absorbing state, or identity matrix, captures all runs longer than that being tested. The run length probability follows from the chance of entering this absorbing state after stepping the new chain over the length of the feature. This amounts to a recursion with the sub-matrices followed weighting by the steady-state or symbol's stationary state vector to sum over all possible starting symbols. We assume the symbol's Markov chain has order one.

These tests use `find.runs` to determine the number of runs and longest within the endpoints, and ignores any NA and NaN values in `x`. Its `feps` argument is used to consider which real values are the same. NA and NaN start or end indices generate NA statistics and probabilities, so that the features from `Dipeak` and `Diflat` can be passed directly. Note that if doing so, and passing the difference of the spacing as `x`, `stID` should be increased by 1 to account for the point lost in the difference. Numeric indices are rounded to integers, and other types or values that are out of bounds to the vector raise errors. If the runs are based on a discrete set of values, as they are in Dimodal, then feps can be set to 0, otherwise the option `"peak.fhtie"` could be used.

The runs statistic test involves an O(`n`) scan of each sequence to count the number of symbols within; there is therefore little advantage to calling this with vectors of indices, rather than one making separate calls one feature at a time. The longest runs test requires estimating the transition matrix over all data, which is O(`n`), and this overhead can be shared by calling it once for all features. The actual recursion is expensive, involving O(`2 l^2 n`) matrix multiplications and additions, where `l` is the longest run length and `n` the sequence length; the matrix size equals the number of symbols (3x3 for the signed difference).

The probabilities should be evaluated against options `"alpha.nrun"` and `"alpha.runlen"` for the minimum passing level.

Value

`Dinrun.test` and `Dirunlen.test` return lists of class `"Ditest"` with elements

`method`	a string describing the test
`statfn`	function used to evaluate significance level/probability
`statistic`	what is tested, the number of runs or maximum length
`statname`	text string describing the statistic
`parameter`	other distribution arguments, for the runs test `Erun` the expected count and `Vrun` its variance, for the maximum run length `featlen` the feature length
`p.value`	probability of feature
`tmat`	for the run length test, the transition matrix of the Markov chain
`wt`	for the run length test, the weight applied to starting to each state

`statistic`, `parameter`, and `p.value` are vectors with the length of the stID and endID vectors.

References

A. Wald, J. Wolfowitz (1940), On a test whether two samples are from the same population. *The Annals of Mathematical Statistics* 11, pp. 147–162.

I. Kaplansky and J. Riordan (1945), Multiple matching and runs by the symbolic method, *The Annals of Mathematical Statistics*, 16, pp. 272–277.

See Also

`find.runs`

Examples

```
## The inner diff generates the spacing, the outer the signed difference.
xrun <- sign(diff( diff(sort( iris$Petal.Width )) ))
## No epsilon needed for signed values.
Dinrun.test(xrun, 1, length(xrun), 0)
Dirunlen.test(xrun, 1, length(xrun), 0)
```

`Dimodal Utility Functions` *Dimodal Utility Functions*

Description

Miscellaneous functions for working with Dimodal results.

Usage

```
midquantile(x, q=((1:length(x))-1)/(length(x)-1), type=0L, feps=0.0)
runs.as.rle(runs, x)
select.peaks(pk)
center.diw(m)
match.features(m, near=10, foverlap=0.70, nomatch=NA_integer_, quiet=FALSE)
shiftID.place(feat, offset, xmid, midoff)
```

Arguments

`m`	a `"Dimodal"` object returned from `Dimodal`
`x`	the original data, with the same length as the members of `runs`
`runs`	the list returned from `find.runs`
`pk`	a `"Dipeak"` object
`near`	maximum distance in points between matching peaks, or as a fraction of the length of the original data
`foverlap`	minimum fraction of the length of either flat that the common segment must cover
`nomatch`	value to use when a feature has no match in the other spacing, treated as integer internally
`quiet`	a boolean, TRUE to only determine the matching, FALSE to also print the aligned features
`q`	quantile(s) for mid-quantile approximation, by default at the data indices
`type`	algorithm determining segments approximating `x`, an integer from 0 to 4 as described in Details
`feps`	tolerance for matching values, per `find.runs`
`feat`	a `"Dipeak"`, `"Diflat"`, or `"Dicpt"` data frame
`offset`	an integer, the amount to shift position of peaks or points or endpoints of flats
`xmid`	a vector of interpolated quantiles to convert indices back to raw data, as stored in the `"Didata"`
`midoff`	an integer, the amount to shift positions in addition to offset

Details

The midquantile function approximates the quantile function by replacing the steps of the ecdf distribution with piecewise linear segments; see Ma, Genton, and Parzen (2011). This creates a ramp over tied or discrete values, giving a better estimate of the position of features, especially when there are large gaps between modes and few or no data points within them. The function determines the segment endpoints and by default evaluates them on the original data grid, scaling the vector indices to run from 0 through 1. It first converts the data to runs using find.runs, with feps defining ties. Segmentation type 1 is the mid-distribution function of Ma, with the data value at the ends of runs shifted to the middle of the change. Segmentation type 2 instead shifts the quantiles by half an index, extending the step in the ecdf. These two approaches can create an envelope around the quantile function, with the type 1 offset from the data at q = 0 and the type 2 at q = 1. Segmentation type 3 combines both shifts, interpolating on a half grid for both x and q. It follows the quantile function better, but does round off the curve at single data points. In practice types 1 and 3 are close. Type 4 runs segments between the middle of runs, or through the data points when there are none. This reduces its estimation error, but the strategy does assume that the step in data to either side of the run is about the same. If not, the other approximations would move away from the center of the run.

Type 4 is best when the data has very few ties. Use types 3 or 1 when there are. Type 0 will automatically select the strategy, using 3 when there are ties and 4 when not. It uses a simple check, whether the number of unique values is a tenth of the data, to decide if there are enough ties.

Internally the function makes two calls C-side. .Call("C_midq", x, type, feps, PACKAGE="Dimodal") returns a vector with the piecewise linear segments, with $x the endpoints along the data and $q along the quantiles. .Call("C_eval_midq", pts$x, pts$q, q, PACKAGE="Dimodal") uses these segments as the first two arguments and new quantiles as the third to interpolate data values.

The find.runs returned value has two vectors with the length of the data. One has non-zero values at the start of runs, the other counts skipped invalid points. The "rle" class is more compact, storing only the runs and the data values at the start. The runs.as.rle function does this compaction.

find.peaks returns a data frame with not only maxima but also the minima between them. It includes maxima even if they are at the first or last point, with minima to only one side. select.peaks selects only those peaks surrounded by minima. It may return a "Dipeak" object with no rows. pk need not include the modifications from Dimodal; select.peaks keeps all columns of its argument.

Indexing in interval spacing is at the end of the interval but the low-pass filter is centered. center.diw shifts the interval spacing features to align with the data, including peak positions, flat source identifiers, and flat start and end points. Note that the raw value is already shifted when set by Dimodal and will not change.

match.features aligns peaks and flats between the low-pass and interval spacing. It compares only valid maxima, as per find.peaks, and shifts interval spacing positions with center.diw before matching them. Peaks must lie within near points to match. Flats must overlap, and the common segment must be at least foverlap of the length of either flat. The function prints the position, raw value, and the number of tests that have passed their acceptance level, unless quiet is TRUE. The nomatch value is cast internally to an integer and cannot be between 1 and the number of features in either spacing, to prevent conflicts. NA, 0, and negative values are acceptable.

The shiftID.place function is used in Dimodal to modify the placement of features, and is provided separately if the detectors find.peaks, find.flats, or find.cpt are called directly. It adds offset to the columns "pos", "stID", "endID", "lminID", "rminID", "lsuppID", and "rsuppID" if they exist in the features data frame to account for values skipped during filtering. Use the "lp.stID" attribute for low-pass features, "diw.stID" for interval spacing, and 2 for changepoints. If pos, stID, or endID are in the data frame the function also adds columns "x", "xst", and "xend" respectively with the original data value for the index by using the midquantile result xmid. Here the index is further modified by midoff; use 0 for low-pass features and changepoints, and half the interval width, stored as attribute "wdiw".

Value

midquantile returns a vector the same length as the data. Quantiles outside the range [0,1] return the first or last data point, even if this is discontinuous with the values at 0 or 1. In other words, the function does not follow the piecewise linear segment outside the valid range, but clips it. NA or NaN quantiles propagate.

runs.as.rle returns a list of class "rle" with members "lengths" and "values", as per the rle command. It also adds a member "nskip" with the number of non-finite values in the data within the run.

select.peaks returns a subset of the argument, possibly with zero rows. If the argument is not a "Dipeak" object, it returns a dummy empty object.

center.diw returns its argument with modified diw.peaks and diw.flats, if they exist.

match.features returns a list with four elements. "peak.lp2diw" is a vector with one element per row in lp.peaks whose value is the matching row number in diw.peaks, or nomatch if there is no match or the lp.peaks row is not a valid peak. "peak.diw2lp" is a similar map from diw.peaks to lp.peaks. "flat.lp2diw" and "flat.diw2lp" are the equivalent maps for flats.

shiftID.place returns the modified feat data frame.

References

Y. Ma, M. Genton, E. Parzen (2011), Asymptotic properties of sample quantiles of discrete distributions. *Ann Inst Stat Math* 63, pp. 227–243.

See Also

Dimodal, Didata, find.runs, rle, find.peaks, Dipeak, find.flats, Diflat, find.cpt, Dicpt, ecdf

Examples

```
m <- Dimodal(faithful$eruptions, Diopt.local(analysis=c('lp','diw')))
# How many peaks were found?  Use print.data.frame to see the full structure.
nrow(select.peaks(m$lp.peaks))
nrow(select.peaks(m$diw.peaks))
# Compare to m$diw.peaks.
m$diw.peaks
center.diw(m)$diw.peaks
# Flats do not match because the Diw feature only covers 50% of the LP.
match.features(m)

plot(sort(iris$Petal.Length))
lines(midquantile(iris$Petal.Length, type=1L), col='red')
lines(midquantile(iris$Petal.Length, type=2L), col='blue')
lines(midquantile(iris$Petal.Length, type=3L), col='green')
lines(midquantile(iris$Petal.Length, type=4L), col='orange')

# See the Dimodal.R source code for the use of shiftID.place.

# To simplify the runs in the signed difference of the interval spacing
# runs.as.rle(Dimodal:::find.runs(m$data['signed',], 0.01), m$data['signed',])
```

Diopt *Options for Dimodal package*

Description

Analysis, test, and display parameters for the Dimodal package. Modeled on the `par` options storage.

Usage

```
Diopt(...)
Diopt.local(...)
```

Arguments

`...`	access or change or override options

Details

Diopt provides a global database of options for the Dimodal analysis. The function can be called in five ways:

`Diopt()` returns the current options set as a list of tag=value pairs

`Diopt(NULL)` resets all options to the package defaults, returning values before reset

`Diopt(tag1=val1, tag2=val2, ...)` checks the option values and stores those that are valid, returning in a list the old values of the tags or NULL if a value is rejected

`Diopt(list(tag1=val1, tag2=val2, ...))` checks the option values and stores those that are valid, returning in a list the old values of the tags or NULL if a value is rejected

`Diopt("tag1", "tag2", ...)` returns the requested option(s) in a list

The function does sanity checking on the passed values for each option, and is preferred to directly modifying the elements in the returned list. Like `par`, `Diopt(Diopt(tag1=val1, tag2=val2))` will restore the original state.

Diopt.local returns all current values of the options overridden by the tag=value pairs in the argument, if the values are valid. Like Diopt, the arguments can be wrapped in an unnamed list. The options stored do not change; this is for a local, one-off edit to the values. Do not use Diopt when setting an `opt` argument for other functions, which expect the full list of values and not just those changed.

String options are partially matched. Fraction values are between 0 and 1, exclusive.

Value

As described in Details.

Parameters: Data Preparation

`analysis` **<string>**
: which type of spacing to analyze, one or more of "lp", "diw", and "cpt" (case-insensitive) for using a low-pass filter, interval spacing, or raw spacing and changepoints

`data.midq` **<integer with valid midquantile type>**
: approximation method to convert quantiles or indices to original data, if 0 determines automatically

Parameters: Low-Pass Filter Setup

`lp.kernel` **<string>**
: filter type, one of "kaiser", "triangular" or "bartlett" (synonyms), "hanning", "hamming", "gaussian" or "normal" (synonyms), "blackman" (case-insensitive)

`lp.window` **<fraction between 0 and 1 or positive integer>**
: kernel size, either as fraction of the data or in points

`lp.tests` **<string>**
: tests to run on detected features, one or more of "ht", "pkexcur", "len", and "ftexcur" (case-insensitive)

`lp.param` **<list>**
: list of tag=value pairs of detector or test or acceptance parameters that will be overridden for the low-pass spacing checks

Parameters: Interval Spacing Setup

`diw.window` **<fraction or positive integer>**
: interval size, either as a fraction of the data or in points

`diw.tests` **<string>**
: tests to run on detected features, one or more of "pkexcur", "runht", "nrun", "runlen", "ftexcur" (case-insensitive)

`diw.param` **<list>**
: list of tag=value pairs of detector or test or acceptance parameters to override for the interval spacing checks

Parameters: Local Extrema Detector

`peak.fht` **<fraction>**
: minimum peak height as a fraction of the signal's range

`peak.frelht` **<fraction>**
: minimum peak height as a fraction of the average with the adjacent minima

`peak.fhttie` **<fraction>**
: maximum relative difference to treat values around extrema as the same

`peak.fhsupp` **<fraction>**
: fraction of peak height to determine the support range, used for excursion test (ex. 0.5 is the Full Width at Half Maximum, 1.0 extends to the minima)

Parameters: Flat Detector

`flat.fripple` **<fraction>**
: ripple specification as fraction of the signal's range

`flat.minlen` **<positive integer incl. 0>**
: shortest length of flats in data points

`flat.fminlen` **<fraction>**
: shortest length of flats as fraction of the data

`flat.noutlier` **<positive integer incl. 0>**
: number of outliers beyond the ripple allowed between source of flat and each endpoint

`flat.distrib` **<string>**
: null distribution model for length model test, one of "logistic", "weibull", "normal" or "gaussian" (synonyms), or "gumbel" (case-insensitive)

Parameters: Excursion/Permutation Tests

`excur.nrep` **<positive integer>**
: number of trial draws for peak and flat excursion tests

`excur.ntop` **<positive integer incl. 0>**
: number of points at start and end of data to ignore if they are the largest, to avoid adding the large tails of the spacing to the draw pool; can compare the first and last spacings against its standard deviation

`excur.seed` **<positive integer incl. 0>**
: RNG seed for each excursion test, 0 to not set

`excur.nrep` **<positive integer>**
: number of trial draws for run permutation test

`excur.seed` **<positive integer incl. 0>**
: RNG seed for each permutation test, 0 to not set

Parameters: Test Significance (Acceptance) Levels

`alpha.ht` **<fraction>**
: acceptance level of the peak height model test

`alpha.pkexcur.lp` **<fraction>**
: acceptance level of the peak height excursion (bootstrap) test for features found in the low-pass spacing

`alpha.pkexcur.diw` **<fraction>**
: acceptance level of the peak height excursion (bootstrap) test for features found in the interval spacing

`alpha.len` **<fraction>**
: acceptance level of the flat length model test

`alpha.ftexcur.lp` **<fraction>**
: acceptance level of the flat height excursion (bootstrap) test for features found in the low-pass spacing

`alpha.ftexcur.diw` **<fraction>**
: acceptance level of the flat height excursion (bootstrap) test for features found in the interval spacing

`alpha.runht` **<fraction>**
: acceptance level of the runs permutation test

`alpha.nrun` **<fraction>**
: acceptance level of the runs count statistics test

`alpha.runlen` **<fraction>**
: acceptance level of the longest run test

Parameters: Changepoint Detectors

`cpt.libs` **<string>**
: vector of changepoint packages to use (name preceded by +) or ignore (by -)

`cpt.maxfncpt` **<fraction>**
: maximum fraction of data allowed to be changepoints, before ignoring library results

`cpt.qvote` **<pair of fractions>**
: vector of two quantiles specifying range of number of changepoints per library allowed based on counts from all libraries

`cpt.fsep` **<fraction>**
: maximum separation between changepoints as fraction of data to consider as same point

`cpt.sep` **<positive integer>**
: separation in data (number of points) between changepoints before merging or treating library results as one

`cpt.libsep` **<positive integer incl. 0>**
: separation in data (number of points) between changepoints when merging the variants within a library, if zero take union of all points

`cpt.timeout` **<positive value>**
: time-out in seconds per CPT algorithm before abandoning library method, 0 for no limit

Parameters: Other

`track.maxwindow` **<fraction or positive integer>**
: maximum low-pass or interval window size for Ditrack variations

Parameters: Display

`palette` **<string>**
: the color palette to use when plotting the spacing or histogram and marking features, either a name from `palette.pals` or if starting with the string "hcl:" from `hcl.pals`, where the matching of the name follows the same rules as the palette functions

`colID.data` **<integer 1 t/m 8>**
: index in the palette (of the 8 colors generated) for drawing the data or its spacing

`colID.filter` **<integer 1 t/m 8>**
  index in the palette for drawing the LP or interval spacing

`colID.hist` **<integer 1 t/m 8>**
  index in the palette for drawing the bars in the data histogram

`colID.cdf` **<integer 1 t/m 8>**
  index in the palette for drawing the distribution atop the histogram, and the decile axes to align the spacing index with the data

`colID.peak` **<integer 1 t/m 8>**
  index in the palette when marking the peaks with vertical dashed or dotted lines

`colID.flat` **<integer 1 t/m 8>**
  index in the palette when marking the flats with boxes around the feature

`colID.cpt` **<integer 1 t/m 8>**
  index in the palette when marking the changepoints with ticks along the upper edge of the graph

`digits` **<positive integer incl. 0>**
  number of significant digits to use when printing raw values, if 0 then use `options("digits")`

`mark.alpha` **<logical>**
  TRUE to use ANSI escape codes when printing probabilities to compare them against the alpha levels, FALSE to not add any formatting

`mark.flat` **<string>**
  how to indicate flats in a plot, one of "box" or "bar" (case-insensitive)

See Also

`Dimodal`, `palette.pals`, `hcl.pals`

Examples

```
## Use the triangular kernel spanning 10% of the data for the
## low-pass analysis.  Set the interval to 30 and override the
## ripple spec in the interval spacing but not for the low-pass.
Diopt(lp.kernel='bartlett', lp.window=0.10, diw.window=30, diw.param=list(flat.fripple=0.02))
## Diopt()$lp.kernel also accesses the new value.
Diopt('lp.kernel', 'diw.window')
## Temporarily override the kernel, without changing default.
Diopt.local(list(lp.kernel='Hanning'))$lp.kernel
Diopt()$lp.kernel
## Reset.
Diopt(NULL)
```

`Dipeak` *Class methods for Dipeak Objects*

Description

Common functions to handle `"Dipeak"` class results, which store information about local extrema and their probability.

Usage

```
## S3 method for class 'Dipeak'
print(x, ...)
## S3 method for class 'Dipeak'
summary(object, ...)
## S3 method for class 'Dipeak'
mark(x, hist=NULL, opt=Diopt(), ...)
```

Arguments

`x`	an object of class `"Dipeak"`
`object`	an object of class `"Dipeak"`
`hist`	the histogram being marked or NULL for a low-pass or interval spacing graph
`opt`	local version of options to guide plots
`...`	extra arguments, ignored for all methods

Details

A `"Dipeak"` object is a data frame that describes local extrema, including their position in the low-pass or interval spacing and equivalent raw value, their height and width, and any tests that have run to judge the significance of the feature. The basic information comes from the `find.peaks` detector, and the `Dimodal` analysis adds the feature analysis. The class inherits from `"data.frame"`.

The print method shows the position (index) and raw value of each maximum, the minimum to either side, and if different the support. It then has two tables with the statistic (test value) and probability for any tests that have been run, as well as the acceptance levels. These come from the options `"alpha.ht"` for the model test, `"alpha.pkexcur.lp"` and `"alpha.pkexcur.diw"` for the excursion test, and `"alpha.nrun"` and `"alpha.runlen"` and `"alpha.runht"` for the runs tests. If the option `"mark.alpha"` is TRUE then probabilities at or below the acceptance level are underlined. Set the option FALSE if your terminal does not support ANSI escape codes, or see the **crayon** package for the extensive tests needed to determine this automatically. These options are set at the global level and cannot be overridden for a single call to print. The pass column after the overall peak probability will contain the number of accepted/passing tests, if any. Raw values and statistics are printed with option `"digits"` significant digits. If the option is set to zero then the standard `options()$digits` is used.

The summary method lists the position and raw value of each maximum, the overall probability or best result from all tests, and a list of the tests that pass per the `Diopt` acceptance levels.

Peaks and minima are marked in a spacing graph by vertical lines. Maxima are drawn with dashes and are made thick if the peak passes any test. Minima are drawn with dotted lines. The marks are colored per the option `"colID.peak"` of the current palette. Histogram annotations include only lines for peaks, drawn thick if significant. They run from the quantile axis on the right to the distribution function curve, then down to the data value. Marking the histogram assumes that Dimodal has added the `x` column with the peak's original data value and has shifted positions.

`select.peaks` is a utility function that extracts only those rows with valid peaks, which have a minima to each side.

Value

The basic data frame returned from `find.peaks` includes the columns

`pos`	index in data of local extremum
`ismax`	TRUE if a local maximum, FALSE if a minimum
`valsd`	data at extremum, standardized by `sd(x)`
`lht`	difference in valsd between maximum and minimum to its left, NA if no minimum or if a minimum
`rht`	difference in valsd between maximum and minimum to its right, NA if no minimum or if a minimum
`lminID`	position/index of minimum to the left, NA if not a valid maximum
`rminID`	position/index of minimum to the right, NA if not a valid maximum
`lsuppID`	left index of support, NA if not a valid maximum
`rsuppID`	right index (incl.) of support, NA if not a valid maximum

The indices are located in the data passed to the detector. Valid maxima have minima on both sides. There will always be two extrema at the first and last point.

`Dimodal` adds several columns. It shifts all indices to the original data, compensating for the invalid points skipped during low-pass filtering or interval spacing. Mapping indices back to the raw data and summarizing the tests, it adds

`x`	the location of the peak in the original data
`ppeak`	the best probability of all tests run or NA if none or if not a maximum with minima to both sides
`naccept`	number of tests passing their acceptance level, 0 if none or ppeak is NA

x can be generated by calling the utility function `shiftID.place` on the find.peaks result. Dimodal also adds an attribute `"source"` with value "LP" or "Diw" depending on the filter used.

For low-pass testing the results may include

`pht`	the probability of the feature per the height model, using the larger of `lht` and `rht`
`hexcur`	the feature height used for the excursion test
`pexcur`	the probability of the peak using bootstrap sampling of the difference of the low-pass spacing

For interval spacing tests the results may include those based on runs in its signed difference, between the bounding minima

`nrun`	the number of runs
`pnrun`	the probability of nrun using the Kaplansky-Riordan test
`runlen`	the length of the longest run
`prunlen`	the probability of runlen using a Markov chain model of the entire data sample
`runht`	the feature height in the signed difference

`prunht`	the probability of the peak with height runht using a permutation test of the runs
`hexcur`	the feature height in the interval spacing between minima
`pexcur`	the probability of the peak with height hexcur using bootstrap sampling from the interval spacing of the whole data

The methods return the object, invisibly.

See Also

`find.peaks`, `Dimodal`, `Diopt`, `data.frame`, `select.peaks` `shiftID.place`

Examples

```
## We override the analysis option to avoid changepoints,
## which may not be available.
m <- Dimodal(faithful$eruptions, Diopt.local(analysis=c('diw','lp'), diw.window=16))
summary(m$lp.peaks)
m$diw.peaks
```

`Ditest` *Class Methods for Ditest Objects*

Description

Common functions to handle `"Ditest"` class results from the Dimodal feature tests.

Usage

```
## S3 method for class 'Ditest'
print(x, ...)
## S3 method for class 'Ditest'
summary(object, ...)
```

Arguments

`x`	an object of class `"Ditest"`
`object`	an object of class `"Ditest"`
`...`	extra arguments, ignored for all methods

Details

The values of the individual tests specify the contents of the object. These functions provide a common formatting of the result. The print methods show the test applied and all parameters, as well as the result. The summary methods show the tested statistic and the resulting probability.

Value

the argument `x` or `object`, invisible

See Also

Dipeak.test, Diflat.test, Dinrun.test, Dirunlen.test, Diexcurht.test, Dipermht.test

Examples

```
ptst <- Dipeak.test(1:4, 250, 0.15, 'bartlett')
ptst
## Summary results.
summary(ptst)
```

Ditrack *Class methods for Ditrack Objects*

Description

Find peaks and flats that appear in the spacing as the size of the low-pass filter or interval changes. Support methods for the class include printing and plotting the significant features.

Usage

```
Ditrack(x, smooth=c('lp', 'diw'), opt=Diopt())
## S3 method for class 'Ditrack'
print(x, ...)
## S3 method for class 'Ditrack'
summary(object, ...)
## S3 method for class 'Ditrack'
plot(x, feature=c('peaks', 'flats'), opt=Diopt(), ...)
```

Arguments

`x`	for `Ditrack`, the data to study; for the class methods, the value returned from `Ditrack`
`object`	an object of class "`Ditrack`"
`smooth`	which filtering to apply to the data, "lp" to look for features in the low-pass spacing or "diw" in the interval spacing
`opt`	local version of options passed to `Dimodal` analysis
`feature`	which feature(s) to plot, may include both
`...`	extra arguments, ignored for all methods

Details

The mode tree and SiZer visualization tools track a feature's position and some metric as the bandwidth of a kernel density estimate changes. They help find the best bandwidth for the trade-off between feature detectability and smoothing. The `Ditrack` class does the same for the spacing analysis. It varies the size of the low-pass filter kernel or interval spacing and tracks the position and probability of peaks and flats per the tests in the `Dimodal` analysis.

The `Ditrack` function varies the window size and gathers the peak and flat results into matrices.

The tracker will overwrite the options `"analysis"` and either `"lp.window"` or `"diw.window"` from `Diopt`. These should not appear in the `"lp.param"` or `"diw.param"` overrides, which have precedence. It steps these sizes from 0.01 to the option `"track.maxwindow"`.

`plot.Ditrack` graphs one or both features, with the window size on the vertical axis and the position along the horizontal. Peaks and minima are drawn as symbols, with dots for the maxima and dashes for the minima. The dots are filled in if any test passes its acceptance level. Flats are drawn with a line spanning the feature. Both are color-coded according to the overall probability, the best of all tests that were run. The coloring is fixed and does not use the `"palette"` option. There will always be a minimum between two peaks. Extrema at the edges of the data are not drawn.

`print.Ditrack` gives a table with the number of significant features at the 0.01 and 0.05 level plus the number of features significant according to the current acceptance levels, as the window size changes.

The summary method prints the ranges of window sizes for peaks and flats that give features that pass the acceptance levels of any test that has been run.

Value

The tracker returns a list assigned to class `"Ditrack"` with elements

`peaks`	a subset of the `"Dipeak"` result with columns `"pos"`, `"ismax"`, `"ppeak"`, and `"naccept"`, plus an additional column `"winpct"` giving the filter size as an integer percentage, keeping only rows with valid peaks
`flats`	a subset of the `"Diflat"` result taking columns `"stID"`, `"endID"`, `"pflat"`, and `"naccept"`, plus an additional column `"winpct"` giving the filter size
`nx`	the length of the data `x` argument
`smooth`	the argument of the same name

The `peaks` and `flats` matrices will have one row per feature, be it a minimum or maximum from `Dipeak` or a flat. There may be many rows with the same `winpct`, or none for a given value.

See Also

`Dimodal`, `Dipeak`, `Diflat`

Examples

```
## Not run because of the run time.
 trk <- Ditrack(quakes[,3], 'lp')
## Two plots side by side.
 dev.new(width=8,height=4) ; plot(trk)
 trk
```

`find.cpt` *Common changepoint detector.*

Description

Find common changepoints by majority vote of detectors available on system.

Usage

```
find.cpt(x, cptlibs, fncpt.max, timeout, qvote, sep, fsep, libsep)
```

Arguments

`x`	a vector of real or integer values
`cptlibs`	library inclusion or exclusion list
`fncpt.max`	maximum fraction of data to allow as changepoints
`timeout`	maximum time in seconds to allow for each detector method, 0 for no limit
`qvote`	pair of quantiles (ex. `c(0.1,0.9)`) for acceptable changepoint counts over all libraries
`sep`	consider changepoints within sep data points to be the same when voting
`fsep`	consider changepoints within fraction fsep of the data to be the same when voting
`libsep`	consider changepoints within libsep data points to be the same when merging library variants

Details

The changepoint detector `find.cpt` does not itself look for changes in the data. Instead, it uses whatever external libraries are available on the system and combines their individual results by majority vote into a common set of changepoints. find.cpt operates in two steps. First, it runs the detectors in each library. There may be more than one variant available. Second, the individual results are combined, first within the library, taking the union of each variant's result, and then a union over all libraries. At each of these merges close changepoints are considered to be the same, with different cut-offs for 'close'. Libraries must be consistent in the number of points they identify or they are ignored. Then for each point in the final merge, find.cpt removes those that appear in less than half the libraries. The final list is found in the `"cpt"` data frame of the `"Dicpt"` data structure.

We have not attempted to quantify the performance of the detectors; during development we have seen a large variation in their results, with none consistently matching the feature detectors. The information provided by each detector varies tremendously, as does the quality of the implementation and algorithm. Therefore, find.cpt will use any library it knows about and finds installed on the system. The `cptlibs` argument can change the selection. It is a vector of library names preceded by '+' or '-'. If any '+' libraries are specified, then these form the base set, replacing the system probe. '-' entries are deleted from the set. Matching of names is done with `pmatch` so that distinct shorter strings can be used.

Although Dimodal does not depend on any changepoint library, we do recommend using at least the **ICSS**, **jointseg**, and **changepoint.np** detectors. They complement one another, are consistent, and reflect the features found in many data sets. The **mosum** and **anomaly** libraries also seem to do well.

If there are three or more libraries in the set when done, we add the built-in level section detector, unless it has been excluded with '-lvlsec' or '-level.sections'.

find.cpt runs each detection method provided by a library. There may be more than none, using different analysis strategies, statistical tests, or evaluation functions. Because some algorithms are `O(n^2)` or worse, the `timeout` argument with a positive value can be used to limit the run time of any individual variant. Some detectors require post-processing of their results. Detectors are run with their default values, and there is no way to change them or the variants that are run.

The detectors differ greatly in the number of changepoints they identify, and find.cpt makes two checks for consistency. First, any detector that selects more than the fraction `fncpt.max` of the data is considered to be noisy and its result is ignored. Second, the `qvote` quantiles are used to discard libraries whose point count is outside the range. If there are 10 changepoint libraries, `qvote=c(0.1, 0.9)` will drop the two with the fewest and most changepoints. Use `c(0,1)` to keep all libraries in the voting. The quantile range will be relaxed so that at least five libraries remain for voting. There is a trade-off between the consistency from a tight quantile range, the number of libraries available, and having a reasonably large number left afterward for voting.

Changepoints are combined twice. First, any variants within the library are joined in a simple union, considering points separated by `libsep` indices to be the same. The separations can form chains, and the point(s) are replaced by their average. Then, after ignoring libraries for consistency, the proposed master changepoint list is a union of all library results. At all indices the algorithm counts the number of library points within the smaller of the distances from `sep` or `fsep`. Local maxima within the count form the proposed list, after a final merging of nearby points. These proposed changepoints do not chain.

The final changepoint set is the subset of the proposed list that matches at least half the library points using the same separation criterion.

The `"Dicpt"` data structure contains the individual results, for the variants and per-library roll-up, and the final changepoint list.

Using voting to combining individual classifiers is the simplest approach. It provides no estimate of the significance of the changepoints, which reflects the wide variety of data provided by the detectors: not all quantify their results, or do so implicitly as a threshold during their analysis. Since changepoints mark the change in spacing between features and not the features themselves, they do not locate peaks or the edges of flats, unless the transition between modes is very sharp or well-defined. Overall we find more changepoints than features, and they are sensitive to the inherent increase in spacing at either tail of the data.

The arguments correspond to options `"cpt.libs"`, `"cpt.fncpt.max"`, `"cpt.timeout"`, `"cpt.qvote"`, `"cpt.sep"`, `"cpt.fsep"`, and `"cpt.libsep"`, respectively. These are provided when called within Dimodal; this function does not access `Diopt` directly.

find.cpt uses `requireNamespace` to probe which libraries are available, and accesses the detectors through the namespace without loading them; this avoids adding many required dependencies to the Dimodal package. During development we have seen that some libraries do not work well with this approach, and raise "<function> not resolved from current namespace" errors. This includes the **changepoint** and **changepoint.np** packages (perhaps in combination, as both have an interior function with the same name), and **bwd**. Loading these libraries beforehand, perhaps in your .Rprofile, seems to solve the problem.

This function is not exported from the **Dimodal** package but may be useful on its own for any data. In Dimodal it is followed by a call to `shiftID.place` to move indices to the original data grid.

Value

find.cpt returns a `"Dicpt"` object. If no changepoints can be found then the result will be a dummy object with no rows in the `cpt` data frame.

The result is not stable from call to call, even if setting the random number generator seed beforehand. We cannot say which libraries are responsible.

Changepoint Libraries

These libraries are supported by find.cpt.

anomaly:
Prototype: `pass(matrix(x,ncol=1), lambda=10)`
Prototype: `capa(matrix(x,ncol=1), type="meanvar")`
We use two methods available in the library, the pass and capa detectors, testing the mean and variance for the latter. The pass method identifies segments of deviant behavior, and we take their endpoints as the changepoint list, without doing any simplification of the segments. The capa method identifies single changepoints which are added afterward. The library underwent an API change, with earlier versions requiring a transform function to standardize the data. find.cpt will handle the change. The lambda parameter is the recommended value.

astsa:
Prototype: `autoParm(x)$breakpoints`
The autoParm function is a very slow detector, and for that reason appears on the default exclusion list in Diopt. Profiling shows half the time is spent in the autoregression and half in the autoParm code.

bcp:
Prototype: `bcp(x)$posterior.prob`
We accept points with a posterior probability of 0.90 or greater. This detector seems to be noisy even with the high cut-off, often being ignored because it reaches the fncpt.max threshold. It uses random splits of the data, so the changepoints found are not stable.

breakfast:
Prototype: `breakfast(x, solution.path=<variable>)`
find.cpt runs the breakfast detector with three solution paths, the sequential IDetect (idetect_seq), Tail-Greedy Unbalanced Haar (tguh), and Wild Binary Segmentation 2 (wbs2). The detector identifies piecewise linear segments, and we take their endpoints as the changepoints. These segments cover the data and tend to be noisy; they may generate more than the fncpt.max threshold.

bwd:
Prototype: `bwd(x)$segments[[1]][,2]`
This library directly provides a changepoint list.

ccid:
Prototype: `detect.ic(matrix(x,ncol=1))$changepoints`
This library directly provides a changepoint list.

changepoint:
Prototype: `cpt.meanvar(x, method='PELT', penalty=<var>, Q=ncpt)@cpts`
We use the PELT partitioning algorithm with four penalty criteria while looking for changes in the mean and variance. These penalties, the SIC, BIC, AIC, and MBIC, may identify similar changepoints, but they often do make different selections. The point count for the `Q` argument follows from the fncpt.max argument.

changepoint.np:
Prototype: `cpt.np(x, method='PELT', penalty=<var>)@cpts`
This is a non-parametric version of the changepoint library. We again use the PELT algorithm with the four penalty criteria. This library seems less noisy than the base version.

cpm:
Prototype: `processStream(x, <var>)$changePoints`
find.cpt combines the library's two sample tests to identify changes in the mean and/or variance using Mann-Whitney (MW, mean), Mood (Md, variance), Lapage (Lp, both), Kolmogorov-Smirnov (KS, both), and Cramer-von Mises (Cvm, both) statistics.

cpss:
Prototype: `cpss.meanvar(x, <var>)@cps`
This library provides several partitioning algorithms, of which we use the segment neighborhood (SN), binary segmentation (BS), and Wild Binary Segmentation (WBS). The three approaches select different changepoints.

Dimodal:
We run `find.level.sections` with alpha=0.95 and with correction. Since the level section uses interval spacing and we assume the input to find.cpt is the spacing, we reconstruct the raw data before calling the detector.

ecp:
Prototype: `e.divisive(matrix(x,ncol=1))$estimates`
This library has not proven stable in testing. e.cp3o and ks.cp3o are left out because they segfault. kcpa is left out because it is slower than the supported e.divisive, which itself is slow. For performance reasons ecp has been put on the default exclusion list, although the points it finds are usually not unreasonable.

ICSS:
Prototype: `ICSS(x)`
This library directly provides a changepoint list. It warns if no points are found, so we suppress the warning.

jointseg:
Prototype: `jointSeg(x, method=<var>, K=ncpt)$bestBkp`
We use two methods, recursive binary segmentation (RBS) and group fused LARS (GFLars), which may produce similar changepoint lists.

mosum:
Prototype: `multiscale.bottomUp(x)$cpts`
This library directly provides a changepoint list. Its results seem to be good and we recommend the library, although it complements ICSS since both algorithms use cumulative sums of the data.

ocp:
Prototype: `onlineCPD(x)$changepoint_lists$maxDP[[1]]`
This library directly provides a changepoint list.

otsad:
Prototype: `which(CpPewma(x)$is.anomaly == 1)`
Prototype: `which(CpSdEwma(x,19)$is.anomaly == 1)`
Prototype: `which(CpTsSdEwma(x,19)$is.anomaly == 1)`
Prototype: `which(CpKnnCad(x,47,l=19,k=27)$is.anomaly == 1)`
The training/window size is set to 19 points, typical for SPC applications. The KnnCad neighbor count of 27 comes from the source paper, which has no guidance on selecting how many neighbors to track. The library seems to generate more changepoints than others and usually drops out of the final vote. The ContextualAnomalyDetector is left out because it requires Python.

Rbeast:
Prototype: `beast(x, season='none')$trend$res`
This library directly provides a changepoint list. Additional arguments to beast (quiet, print.options, print.progress) suppress its many outputs.

strucchange:
Prototype: `breakpoints(x~1, data.frame(x), h=0.05)$breakpoints`
Prototype: `breakpoints(Fstats(x~1, data.frame(x)), h=0.05)$breakpoints`
strucchange uses linear regression to identify changepoints, either modeled directly or evaluated in a two-sample F test.

Several libraries are not supported by find.cpt.

npcp: This only detects if a changepoint occurs, but not where, nor if there are multiple changepoints.

FDRSeg: This library is no longer supported. It was very slow and would segfault.

BayesProject: This is for multivariate data only.

fpop: The detector is sensitive to the lambda parameter with no guidance on how to set its value.

capushe: Documentation was insufficient to set up and use the library.

prophet: This requires a dummy time index and raised an error on its help page example.

gSeg: This runs sequentially, requiring a manual subdivision of the data without providing guidance on how to do that. The graph preparation step was also unclear.

modehunt: This has been replaced by the level section detector.

Refer to the help pages for each library to understand its algorithms. Some functions limit the number of changepoints they report; the value will come from the fncpt.max argument to find.cpt.

Error Handling

Each library call is made within a `tryCatch` with a handler for arbitrary conditions that are raised during its execution. Any output from the library that occurs outside this system is swallowed. This has the disadvantage that code errors in Dimodal during the call and its processing disappear. In the worst case the console will seem to be unresponsive. Restore it by using `closeAllConnections()`.

This approach has not been entirely robust during testing. At least one library (cpss) times out with an error that does not pass through the condition system; it dumps its output to the standard connection and raises an interrupt without any message. Time outs from other libraries (strucchange) do generate errors, presumably from the `R_ProcessEvents()` function.

See Also

`Dicpt`, `find.level.sections`, `Diopt`, `shiftID.place`

`find.flats` *Local flat detector.*

Description

Locate flat sections, specified by the ripple parameter, within data.

Usage

```
find.flats(x, fripple, minlen, fminlen, noutlier)
```

Arguments

`x`	a vector of real or integer values
`fripple`	height of flat as fraction (0 – 1 excl.) of data range
`minlen`	minimum length of flat, in data points (absolute)
`fminlen`	minimum length of flat, as fraction (0 – 1 excl.) of data length (relative)
`noutlier`	number of points outside the ripple range allowed to either side of source

Details

The detector scans every point in `x` to find the contiguous range of indices that lie within the ripple centered at the point's value. The algorithm allows noutlier points to each side of the source point (so the total can be twice this), but not as the first or last points. In other words, the endpoints of the flat will be within the ripple specification. Flats must have a length of the larger of `minlen` or the relative `fminlen` of `length(x)`, counting that portion not covered by a longer flat. Flats may therefore overlap, but only if the extension meets the length requirement. Flats are reported for their original extent and not the exposed portion.

The algorithm will ignore non-finite values in `x`, but the endpoints of the features apply to the original, unscreened data.

The arguments correspond to options `"flat.fripple"`, `"flat.minlen"`, `"flat.fminlen"`, and `"flat.noutlier"`. These are provided when called within Dimodal; this function does not access `Diopt` directly.

This function is not exported from the Dimodal package, but may be useful on its own for any signal. Within Dmodal it is followed by a call to `shiftID.place` to move indices to the original data grid.

The detector switches its scan algorithms based on the data size. The base approach, which scans outward from each source point, is O(`n Lflat`) in time and O(`n`) in memory, where Lflat is the average length of all ranges. This is usually a substantial fraction of the data, a third to half, so the algorithm is essentially O(`n^2`). For larger data sets it switches to a segtree-based search that has a high overhead. The time complexity is not known, and the memory consumption is still O(`n`). The second approach runs faster with more than 10 thousand data points, and is an order of magnitude quicker for 100 thousand points.

Value

find.flats returns a `"Diflat"` object. If there are no flats in the data then the result will be empty and have zero rows.

See Also

`Diflat`, `Diopt`, `shiftID.place`

`find.level.sections` *Detector for intervals without significant slope*

Description

An inversion of the **modehunt** package's test for sloping sections.

Usage

```
find.level.sections(x, alpha, correct)
```

Arguments

`x`	a vector of real or integer data
`alpha`	the significance level of the level test, between 0 and 1 excl.
`correct`	a boolean whether to bias the test against short sections

Details

The **modehunt** test for sloping sections sums interval spacings of different widths at each point in `x`, scaling them by a score function to get a test statistic. Comparing the test statistic to a critical value taken from 100 thousand uniform samples decides if the data has non-zero slope at the point. The algorithm combines these into sloping segments and returns the longest common subset.

This detector performs the same test to find the sloping sections, but processes them differently to identify the shortest non-sloping section beginning at each point. These sections may overlap.

Typical parameter values are 0.95 for `alpha` and TRUE for `correct`. Larger values of alpha will produce fewer sections. They will generally span all of the data.

The test uses a model of the critical value for the test statistic, to avoid the simulated draws made in **modehunt**. The model uses the length of the data and the significance level, and has been generated for lengths from 50 to 5000 and alpha from 0.00001 to 0.99999. Accuracy outside this range cannot be guaranteed.

The level detector was developed to find flats without (explicitly) filtering the data, but the results were poor. Many sections are short and concentrated near the start and end of the sorted data. The detector is sensitive to variations in the data and the division is noisy. The sections that are longer do not identify single modes, and tend to span inter-modal transitions. Any flats in the spacing are a subset of the level sections, in count and in length. There is also a bias towards sections at the start of the data rather than the end due to how the algorithm incrementally chooses the starting point of the next section. In fact, the detector places a minimum length of 5 on intervals at the end to prevent fragmenting the last. The detector cannot be used on its own to locate modes, but it can be taken into the changepoint voting where other algorithms can reduce the impact of its noisy performance.

The analysis is O(`n^2`) in time and O(`n`) in memory. If the data contains integers they are converted to reals. Other data types are not supported. The data is sorted internally.

Although not exported from the Dimodal package, this detector may be useful outside the spacing analysis for any signal.

Value

find.level.sections returns a data frame with one row per section and columns

`stID`	the index in the sorted x of the section's starting endpoint
`endID`	the index in the sorted x of the last endpoint, incl.

The indices refer to the sorted input after removing non-finite values. They must be mapped back to `x` with the mid-distribution function. There is no way to judge the significance of any section.

See Also

`Dicpt`, `modeHunting` in the **modehunt** package

find.peaks	*Local minima and maxima detector.*

Description

Locate local minima and maxima within data, removing small peaks or shallow minima.

Usage

```
find.peaks(x, fht, frelht, fhtie, fhsupp)
```

Arguments

`x`	a vector of real or integer values
`fht`	minimum absolute peak height as fraction (0 - 1 excl.) of data range
`frelht`	minimum relative peak height as fraction (0 - 1 excl.) of average value
`fhtie`	relative fraction (0 - 1 excl.) of values to consider as same
`fhsupp`	fraction (0 - 1 incl.) of max-min heights for support

Details

The detector begins by using `find.runs` to identify sequences of nearly equal values, in case there are flat tops at peaks or in valleys at minima. It treats runs as a single point when marking all triples as local minima (x[i] > x[j] < x[k]) or maxima (x[i] < x[j] > x[k]). It determines the smallest difference between a maximum and its adjacent minima, both as an absolute value and as a fraction of their average, `|x[j] - x[i or k]| / ((|x[j]| + |x[i or k]|)/2)`, for `i` and `k` the minima to the left and right of the maximum `j`. Working from the smallest absolute difference to the largest, the detector checks if either the absolute height is below the fraction `fht` of the data's range or if the relative height is below `frelht`. If so, the peak is merged into its neighbor by deleting it and the minimum and re-calculating the height of the neighbor.

The first and last data point are always extrema, a maximum if they are larger than their neighbors.

We may want to limit the extent of a peak between two minima, for example to avoid one that lies in the middle of a long flat and that makes the peak appear much wider than it is. Let the support of a peak be the points whose value drops less from the peak than the fraction `fhsupp` of the feature's height. If 1.0, the support extends to the minima. If 0 the support is only the local maximum. The Full Width at Half Maximum (FWHM) measure would use set fhsupp=0.5. a larger value, for example 0.9, will back away from a minimum in a flat without distorting the feature for testing.

Although the order of checking and merging peaks can affect the final set, and the process is deterministic, we do not document what this order is. If there are multiple local extrema with the same value, we do not guarantee which will be selected, although the result will be stable over repeated calls.

The arguments correspond to `Diopt` options `"peak.fht"`, `"peak.frelht"`, `"peak.fhtie"`, and `"peak.fhsupp"`. The caller provides them; this function does not access `Diopt` directly.

This function is not exported from the Dimodal package, but may be useful on its own for any signal. In Dimodal it is followed by a call to `shiftID.place` to move indices to the original data grid.

The detector is O(`n log n`) in time and O(`n`) in memory.

Value

find.peaks returns a `"Dipeak"` object. If the data is bad (constant or has two or fewer points or is all NA) then the result will be empty and have zero rows.

See Also

`find.runs`, `Dipeak`, `Diopt`, `shiftID.place`

`find.runs` *Fuzzy run detector*

Description

A runs length detector that handles nearly equal real values and skips invalid values.

Usage

```
find.runs(x, feps)
```

Arguments

`x`	a vector
`feps`	fractional (0 - 1 excl.) relative difference to treat values as same

Details

This runs finder looks for sequences of real values that almost match, considering them as a run of same values. Two points match if the difference between their values as a fraction of their average is less than the threshold: |x[i] - x[j]| / ((|x[i]| + |x[j]|)/2) < feps. For each point in `x` the detector scans forward until a point fails to be close, with the run taken over the matching interval. The base of the comparison is always the first point in the run, and not a chain of adjacent values. The detector treats infinities as equal to each other but not to finite values and skips over any NA or NaN values while counting them. Values within double precision tolerance of each other always match, irrespective of the relative threshold.

Note that this relative difference depends on the data not averaging close to zero, especially at zero-crossing points, and is sensitive to a constant shift. The first is not a problem for spacing, which is always positive, but may require shifting other types of data. That would also help with the second issue.

For integer, logical, and character data, values must match exactly and feps is not used.

Other data types are not supported.

Although not exported from the Dimodal package, this detector may be useful outside the spacing analysis for any signal. Within Dimodal it is called by the run count and length tests and internally within the C code.

The detector returns two vectors, each with the same length as the original data. `runs` stores the length of the run starting at the data point and `nskip` the number of skipped elements within the run. To process the runs, start at index `i=runs[1L]`, or if that is zero `i=skip[1L]`. The next run starts at `i` + `runs[i]` + `nskip[i]`, until i goes beyond the length of the data.

The detector is O(`n`) in both time and memory, albeit with several passes through the data.

Value

find.runs returns a list with elements:

`runs`	an integer vector of length `x` with the length of the run starting at each point or 0 if the point does not start a run
`nskip`	an integer vector of length `x` with the number of skipped data points within the run
`stats`	a vector of three integer values, `nrun` the total number of runs, `maxrun` the longest run, and `nx` the length of the original data and of `runs` and `stats`

Note

Use the utility function `runs.as.rle` to convert the returned value to a `"rle"` encoding.

See Also

`rle`, `runs.as.rle`

`kirkwood` *Asteroid Orbital Axis Showing Kirkwood Gaps*

Description

The distance of asteroids from the sun is multi-modal, with modes corresponding to asteroid families and anti-modes to the Kirkwood gaps.

Usage

```
data(kirkwood)
```

Format

`kirkwood` is a numeric vector of 2093 values representing the semi-major axis of an asteroid in Astronomical Units (A.U.).

Details

The axis is stored in column 18 of the original data file. We keep only those asteroids within the orbit of Jupiter, with an axis below 5 A.U. We further keep only those asteroids whose diameter is known or not blank, in column 7 of the data file.

Source

Lowell Observatory maintains the asteroid ephemeris website at https://asteroid.lowell.edu/astorb/ The kirkwood dataset is a subset of the "astorb.txt.gz" static copy of the ephemeris, downloaded on 1 May 25. The full file contains more than 1.4 million entries, and is growing daily. Lowell Observatory provides a query builder for downloading up-to-date values at https://asteroid.lowell.edu/query-builder

The ephemeris is supported by grants from NASA and the Lowell Observatory; details are found on the website. Further distribution of the data requires acknowledging these funding sources.

References

N. Moskovitz, L. Wasserman, B. Burt, R. Schottland, E. Bowell, M. Bailen, M. Granvik, The astorb database at Lowell Observatory, *Astronomy and Computing*, 41, Oct 2022, pp. 100661.

Examples

```
## Load data.
data(kirkwood, package="Dimodal")

## The Ditrack and changepoints can take a while to run, so to pass
## the CRAN time limit in examples they are marked don't run.  To see
## the complete example call example(kirkwood, run.dontrun=TRUE).

## Start from scratch.
## Normally you would not need to save the results of the call, but
## this clutters up the screen.
opt <- Diopt(NULL)

## Set up the analysis.
## This is a large data set so spacing in features is small but
## smooth, which allows tighter detector parameters.  Values depend
## on the range of the spacing and are found by trial and error.
## Leave off changepoints for repeatability in this example.
## Set RNG seeds for repeatability.
## Using bars to mark flats is easier to read with dense data.
opt1 <- Diopt(peak.fht=0.015, flat.fripple=0.0075, analysis=c("lp", "diw"),
              excur.seed=3, perm.seed=5, mark.flat="bar")

## Use Ditrack to see where passing features (filled in dots) appear.
## Interval spacing is the same cut-off.  Wrapped in \donttest because of
## the run time.
trk <- Ditrack(kirkwood, "lp")
dev.new(width=8,height=4) ; plot(trk)
opt1 <- c(opt1, Diopt(lp.window=0.05, diw.window=0.05))
```

```
## Run analysis.
m <- Dimodal(kirkwood)

## Summarize the data and spacing.
m$data

## Print features that exist in both low-pass and interval spacing.
## Allow large distance between peaks because  of data set size.
mtch <- match.features(m, near=30)

## Gap distance (in AU), dropping the extra LP peak.
## There are gaps at 2.30, 2.48, 2.86, 2.93, and 3.06 AU.
a_gap <- (select.peaks(m$lp.peaks)$x[-3L] + select.peaks(m$diw.peaks)$x) / 2
a_gap

## Orbital resonance with Jupiter, which has orbital axis of
## 5.201 AU, from Kepler's third law.  The corresponding
## resonance is 3.40 (ratio 7:2 or 10:3), 3.04 (3:1),  2.46 (5:2),
## 2.36 (7:3), and 2.22 (9:4).  Simulations confirm all but the
## first and last.
resonance <- (5.201 / a_gap) ^ (3/2)
resonance

## The flats include families of asteroids, although other
## considerations must be taken into account to find true
## family members (composition of asteroid, orbital
## eccentricity and inclination).
## The second range at 2.76 AU includes the Ceres family,
## the third at 3.13 AU the Themis.  The first range at
## 2.64 AU includes the Eunomia and Prosperina families.
m$diw.flats

## 3 graphs side-by-side with results.
dev.new(width=12, height=4) ; plot(m)

## The full test results of the low-pass peaks.
m$lp.peaks

## In the Ditrack plot there are two peaks in the very lower right
## corner.  We need smaller filter kernels/intervals to find these.
## The parameters already set will still apply.
opt3 <- Diopt(lp.window=0.015, diw.window=0.015, peak.fht=0.025, peak.frelht=0.10)
## Running Dimodal with such small windows will generate warnings
## about the model tests being pushed out of bounds.
## This adds gaps at 1.93 (2:1) and 1.74 (7:4), both expected
## from simulations.  The 2:1 gap position shifts because the
## spacing is so large that there is data point to anchor the
## value and a second, smaller increase pulls the peak to 1.93.
m2 <- Dimodal(kirkwood) ; mtch <- match.features(m2, near=30)
dev.new(width=12, height=4) ; plot(m2)

## To run changepoints, do this.
## Using all supported libraries they surround the 3:1, 2:1,
## and 7:4 gaps to the side of the peak, but not directly at
## the feature.  They also mark one side of the 5:2, 7:3, and
## 9:4 gaps.
## Not run:  mcpt <- Dimodal(kirkwood, Diopt.local(analysis="cpt")) ; mcpt
## Not run:  dev.new(width=8, height=4) ; plot(mcpt)
## Show the voting from all libraries.  This is a tall graph.
## Not run:  dev.new(height=10, width=6) ; plot(mcpt$cpt)

## Restore default.
opt <- Diopt(opt)
```

Index