Package 'IRISMustangMetrics'

July 27, 2020

Type Package

Version 2.4.1

Title Statistics and Metrics for Seismic Data

Author Jonathan Callahan [aut],
Rob Casey [aut],
Mary Templeton [aut],
Gillian Sharer [aut, cre]

Maintainer Gillian Sharer <gillian@iris.washington.edu>

Depends R (>= 3.2.0), IRISSeismic (>= 1.3.0)

Imports methods, RCurl, seismicRoll (>= 1.1.4), signal, stringr, XML, stats, pracma, dplyr (>= 0.4.3)

Collate Class-Metric.R BSSUtils.R ISPAQUtils.R basicStatsMetric.R correlationMetric.R crossCorrelationMetric.R dailyDCOffsetMetric.R DCOffsetTimesMetric.R gapsMetric.R PSDMetric.R SNRMetric.R spikesMetric.R STALTAMetric.R stateOfHealthMetric.R upDownTimesMetric.R transferFunctionMetric.R maxRangeMetric.R sampleRateChannelMetric.R sampleRateRespMetric.R

Description Classes and functions for metrics calculation as part of the 'IRIS DMC MUSTANG' project. The functionality in this package builds upon the base classes of the 'IRISSeismic' package. Metrics include basic statistics as well as higher level 'health' metrics that can help identify problematic seismometers.

License GPL (>= 2)

Repository CRAN

NeedsCompilation no

Date/Publication 2020-07-27 04:32:05 UTC

R topics documented:

`IRISMustangMetrics-package`

Utilities for calculating seismic metrics from IRIS DMC data

Description

This package provides S4 classes and functions for calculating metrics from seismological data available from the IRIS Data Management Center (DMC) (http://www.iris.edu/dms/nodes/dmc/). This package is part of the MUSTANG project and is intended for DMC internal use only.

Introduction

The **IRISMustangMetrics** package depends upon the **IRISSeismic** package which defines new S4 classes and methods for manipulating seismic data. Please see the "seismic-intro" vignette for introductory examples on using **IRISSeismic**.

History

version 2.4.1

- fix for max_range where trace length not evenly divided by window length
- max_range now rounds window*samplerate and increment*samplerate to integer values
- corrects typo error in ISPAQUtils.R

version 2.4.0

- adds new metrics sample_rate_resp, sample_rate_channel, max_range

version 2.3.0

- modifies PSD metrics for edge case involving metadata and trace start times
- adds noCorrection option to PSDMetric, if noCorrection=TRUE it only returns uncorrected PSDs and not corrected PSD or PSD-derived metrics; default noCorrection=FALSE

version 2.2.0

- removes dead_channel_exp, metric has been retired
- renames pdf_aggregator in ISPAQUtils.R to pdf

version 2.1.3

- fixed bug related to getGeneralValueMetrics,getMustangMetrics error handling
- added pdf_aggregator to ISPAQUtils.R, for multi-day pdf plotting within ISPAQ

version 2.1.2

- version 2.1.1 unintentionally removed the pct_above_nhnm metric. This version restores it.
- made sample rate sanity rate check consistent between correlationMetric and crossCorrelationMetric

version 2.1.1

- fixed bug in PSDMetrics that affected dead_channel_gsn results

version 2.1.0

- transfer_function requires sample rates to be within a factor of 10 to avoid decimation effects on amplitude
- transfer_function uses 7 order Chebyshev filter in the decimate function, to correct 1% error occurring with default 8 order Chebyshev
- fixed bug in transfer_function trace start and end time comparisons
- transfer_function when determining if sample rate < 1, round to 5 digits first

- getGeneralValueMetrics added metric_error, ts_channel_continuity, ts_channel_up_time, ts_gap_length, ts_gap_length_total, ts_max_gap, ts_max_gap_total, ts_num_gaps, ts_num_gaps_total, ts_percent_availability, ts_percent_availability metrics
- aliased the getGeneralValueMetrics function to getMustangMetrics
- dailyDCOffsetMetric now returns error when result is NaN or NA

version 2.0.9

- removed dependency on pracma package
- removed channel restrictions for pct_above_nhnm,pct_below_nlm
- cross correlation sampling rates of < 1 will round to 2 digits
- getGeneralValueMetrics better handles case of no targets found
- improved error handling in spikesMetric.R

version 2.0.8

- minor bug fix to ISPAQUtils.R, spikes=numSpikes

version 2.0.7

- fixed bug in getGeneralValueMetrics that didn't return measurements if there was more than one for any day
- crossCorrelationMetric filter now defaults to a butterworth 2 pole 0.1Hz (10 second) low pass filter

version 2.0.6

- fixed bug related to NA -> NULL replacement in Class-Metric

version 2.0.5

- fixed dplyr version dependencies

version 2.0.4

- adds additional sanity check to `getGeneralValueMetrics()`
- createBssUrl() adds "&nodata=404" to url

version 2.0.2

- updates to ISPAQUtils.R

version 2.0.1

- removed dependency on tidyr package

version 2.0.0 – GeneralValueMetrics

- `GeneralValueMetric` class introduced, `SingleValueMetric` class deprecated. All metrics that previously returned `SingleValueMetric` now return `GeneralValueMetric`
- `getGeneralValueMetrics()` function added. Retreives metrics measurements from BSS database

- crossCorrelationMetric() does not return timing_drift. The metric proved unreliable
- users can now supply instrument response information in the form of frequency, amplitude, phase to the function PSDMetric, in place of the getEvalresp webservice call

version 1.3.1 – PSDs

- getPsdMetrics reworked

version 1.3.0 – latency

- getLatencyValuesXML() removed from package.
- documentation improvements.
- additional error checking for getSingleValueMetrics().

version 1.2.7 – PSDs

- PSDMetrics() metrics percent_above_nhnm and percent_below_nlnm limited to frequencies less than nyquist/1.5.

version 1.2.6 – PSDs

- Depends on **IRISSeismic** (>= 1.3.0).
- dead_channel_exp and dead_channel_lin metrics will only return values for station channel codes matching "BH|HH".

version 1.2.5 – ISPAQUtils

- ISPAQUtils.R contains functions for use with the ISPAQ standalone metrics system.

version 1.2.4 – package version dependencies

- Depends on **IRISSeismic** (>= 1.2.3). Imports **seismicRoll** (>=1.1.2).

version 1.2.2 – correlationMetric tweak

- correlationMetric() allows trace sample lengths to differ by 2 samples without stopping.

version 1.2.1 – PSDs

- Better fix to very low powers issue in PSDMetrics() dead_channel_gsn metric.
- PSDMetrics() shifts PDF bin centers by 0.5 dB.

version 1.2.0 – PSDs

- PSDMetric() returns corrected PSD and PDF dataframes in addition to uncorrected PSDs and PSD derived metrics.
- Depends on R (>= 3.2.0) and **IRISSeismic** (>=1.1.7).
- Imports **tidyr**, **dplyr**.

version 1.1.3 – bug fix, import version increased

- Fixes typo in SNRMetric() function windowSecs argument default value.
- Imports **seismicRoll** (>=1.1.1)

version 1.1.2 – modifications

- Improves error handling messages.
- `dailyDCOffsetMetric()` removes unused selectivity argument and adds argument controlling output type.
- Fixes bug in `dailyDCOffsetMetrics()` related to outlier removal and vector length.
- Fixes bug in `PSDMetrics()` `dead_channel_gsn` metric related to very low power values.
- `PSDMetrics()` only returns metrics that generate numeric values.

version 1.1.1 – bug fix

- `crossCorrelationMetric()` exits if either input trace is flatlined (all values equal).

version 1.1.0 – updates package dependencies

- Depends on **IRISSeismic** (>= 1.1.0).

version 1.0.8 – new metric and bug fix

- Improves error handling messages.
- Adds new `dead_channel_gsn` metric to `PSDMetric()` function output.
- Fixes bug in `STALTAMetric()` involving required trace length.

version 1.0.7 – bug fix

- Fixes issue with `spikesMetric()` passing argument values to `findOutliers`.

version 1.0.6 – function argument changes

- Changes `spikesMetric()` default argument values `thresholdMin=10`,`selectivity=NA`,`fixedThreshold=TRUE`.
- `transferFunctionMetric()` now requires input of evalresp fap spectra, new arguments `evalresp1` and `evalresp2`.
- Additional sanity checks for `transferFunctionMetric()` and `PSDMetric()`.
- Depends on **IRISSeismic** (>= 1.0.10). Imports **seismicRoll** (>=1.1.0). Imports **stats**.

version 1.0.5 – new PSD metric

- Changes URL syntax for MUSTANG web services to use "format=..." instead of "output=...".
- Adds new `sample_unique` metric to `PSDMetric()` output.

version 1.0.3 – new functionality and bug fixes

- Adds new `metricList2DF()` function.
- Adds new `dead_channel_lin` metric to `PSDMetric()` output.
- Fixes typo in `Class-Metric.R` value string format.

version 1.0.0 – First Public Release

Author(s)

Jonathan Callahan <jonathan@mazamascience..com>

References

IRIS DMC web services: http://service.iris.edu/

Examples

```
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient", debug=TRUE)

# Get the seismic data
starttime <- as.POSIXct("2010-02-27 06:45:00",tz="GMT")
endtime <- as.POSIXct("2010-02-27 07:45:00",tz="GMT")
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)

# Apply a metric and show the results
metricList <- basicStatsMetric(st)
dummy <- lapply(metricList, show)
```

---

basicStatsMetric *Min, median, mean, rms variance, max, and number of unique values of a signal*

Description

The basicStatsMetric() function calculates the `min,median,mean,max,rmsVariance` and number of unique values for the input seismic signal.

Usage

```
basicStatsMetric(st)
```

Arguments

`st`	a `Stream` object containing a seismic signal

Details

This metric applies the `min,median,mean` and `max` methods of `Stream` objects to the `st` parameter to calculate the following metrics:

- `sample_min`
- `sample_median`
- `sample_mean`
- `sample_max`
- `sample_rms`

It also calculates `length(unique(stmerged@traces[[1]]@data))`, where stmerged is the st parameter after `mergeTraces` is applied to it, for the following metric:

- `sample_unique`

Any error messages generated in the process will pass through untrapped.

Value

A list of SingleValueMetric objects is returned.

Note

See the seismic package for documentation on Stream objects and the getDataselect method.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime, inclusiveEnd=FALSE)

# Calculate some metrics and show the results
metricList <- basicStatsMetric(st)
dummy <- lapply(metricList, show)

## End(Not run)
```

---

convertBssErrors *Generate Human Readable MUSTANG Errors*

Description

The MUSTANG database is in charge of storing the results of metrics calculations and is accessed through a webservice API. The convertBssErrors function extracts pertinent error information from the HTML returned by MUSTANG on error conditions.

Usage

```
convertBssErrors(err_msg)
```

Arguments

err_msg	error text received from the MUSTANG

Value

A text string with the root cause extracted from the MUSTANG HTML Java error dump.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

SingleValueMetric-class, metricList2Xml, getMetricsXml, getBssMetricList,

---

correlationMetric *Correlation between channels*

Description

The correlationMetric() function calculates the correlation between two streams of seismic data.

Usage

```
correlationMetric(st1, st2)
```

Arguments

st1	a Stream object containing a seismic signal
st2	a Stream object containing a seismic signal

Details

The correlation returned is a value in the range [0-1]. This 'pearson r' correlation is a measure of the strength and direction of the linear relationship between two variables that is defined as the (sample) covariance of the variables divided by the product of their (sample) standard deviations.

Missing values are handled by casewise deletion with the following R code:

```
cor(x,y,use="na.or.complete")
```

Value

A list with a single SingleValueMetric object is returned. The metric name is cross_talk.

Note

Seismic streams passed to correlationMetric must have the same network and station, must cover the same time range and must have the same sampling rate.

The metricList generated for this two-channel metric will have a SNCL code of the form: N.S.L1:L2.C1:C2.Q.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get seismic traces
starttime <- as.POSIXct("2013-03-01", tz="GMT")
endtime <- as.POSIXct("2013-03-02",tz="GMT")
stZ <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime,inclusiveEnd=FALSE)
st1 <- getDataselect(iris,"IU","ANMO","00","BH1",starttime,endtime,inclusiveEnd=FALSE)
st2 <- getDataselect(iris,"IU","ANMO","00","BH2",starttime,endtime,inclusiveEnd=FALSE)

# Calculate correlationMetric
correlationMetric(stZ,st1)[[1]]
correlationMetric(stZ,st2)[[1]]
correlationMetric(st1,st2)[[1]]

## End(Not run)
```

`createBssUrl` *Create URL to retrieve measurements from the MUSTANG BSS*

Description

The `createBssUrl` method of the `IrisClient` returns a URL that can be used to make a request of the MUSTANG BSS (Backend Storage System).

Usage

```
createBssUrl(obj, network, station, location, channel,
             starttime, endtime, metricName, ...)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code
`channel`	a character string with the three letter channel code
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`metricName`	a character string containing one or more comma separated metric names
`...`	optional arguments `constraint` a character string containing value constraints `url` optional url of the BSS measurements service

Details

A blank location code should be specified as `location="--"`; Using `location=""` will return all location codes.

The default MUSTANG measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

Value

A character string containing a BSS request URL

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

getSingleValueMetrics

Examples

```
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2013-06-01", tz="GMT")
endtime <- starttime + 30*24*3600
metricName <- "sample_max,sample_min,sample_mean"

# Get the measurement dataframe
url <- createBssUrl(iris,"IU","ANMO","00","BHZ",
                    starttime,endtime,metricName)

# This URL can be pasted into a web browser to see the BSS return values
```

---

crossCorrelationMetric

Correlation between channels

Description

The crossCorrelationMetric() function calculates the maximum absolute correlation (`polarity_check`) and lag at maximum correlation (`timing_drift`) associated with two streams of seismic data.

Usage

```
crossCorrelationMetric(st1, st2, maxLagSecs=10, filter)
```

Arguments

st1	a Stream object containing a seismic signal
st2	a Stream object containing a seismic signal
maxLagSecs	maximum number of seconds of lag to use
filter	a **signal** package filter to be applied before cross-correlating, optional

Details

Details of the algorithm are as follows:

- Both signals are demeaned and detrended
- If one signal has a higher sampling rate, it is decimated to the lower sampling rate using an IIR filter if it is a multiple of the lower sample rate. See (signal::decimate).
- Both signals are filtered, by default with a Butterworth 2-pole low pass filter with a 0.1 Hz (10 second) corner frequency. See (signal::filter).
- Signals are cross-correlated using the stats::ccf() function.

The maximum absolute correlation is saved as polarity_check while the lag at peak correlation is saved as timing_drift.

Note: For cross-correlation, seismic signals must not have any gaps – they must be contained in a single Trace object.

Value

A list with one GeneralValueMetric object is returned. The metric names is polarity_check.

Note

The metricList generated for this two-channel metric will have an additional sncl2 attribute identifying the SNCL in st2.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com> (R code), Mary Templeton <met@iris.washington.edu> (algorithm)

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the same signal, shifted by 3 seconds
starttime <- as.POSIXct("2013-11-12 07:09:45",tz="GMT")
endtime <- starttime + 600
st1 <- getSNCL(iris,"NM.SLM.00.BHZ",starttime,endtime)
st2 <- getSNCL(iris,"NM.SLM.00.BHZ",starttime+3,endtime+3)
```

```
# Cross-correlate
crossCorrelationMetric(st1,st2)

## End(Not run)
```

`dailyDCOffsetMetric` *DC Offset Detection*

Description

The dailyDCOffsetMetric() function identifies days with a jump in the signal mean.

Usage

```
dailyDCOffsetMetric(df,
                    offsetDays=5,
                    outlierWindow=7,
                    outlierThreshold=3.0,
                    outputType=1)
```

Arguments

`df`	a dataframe containing `sample_mean` values obtained with getSingleValueMetrics()
`offsetDays`	number of days used in calculating weighting factors
`outlierWindow`	window size passed to findOutliers() function in the **seismic** package
`outlierThreshold`	detection threshold passed to findOutliers() function in the **seismic** package
`outputType`	if 1, return last day of valid values (index= length(index)-floor(outlierWindow/2)); if 0, return all valid values (indices= max(offsetDays, floor(outlierWindow/2): length(index)-floor(outlierWindow/2))

Details

This algorithm calculates lagged differences of the daily mean timeseries over a window of `offsetDays` days. Shifts in the mean that are persistent and larger than the typical standard deviation of daily means will generate higher metric values.

Details of the algorithm are as follows

```
# data0 = download requested daily means (in the 'df' dataframe), must be greater than max(offsetDays,ou
# data1 = remove outliers using MAD outlier detection with the 'outlier' arguments specified
# data2 = replace outliers with rolling median values using a default 7 day window, remove last floor(out
# weights = calculate absolute lagged differences with 1-N day lags, default N=5
# metric0 = multiply the lagged differences together and take the N'th root
# stddev0 = calculate the rolling standard deviation of data2 with a N-day window
# METRIC = divide metric0 by the median value of stddev0
```

Value

A list is returned with a SingleValueMetric object for the last day-floor(outlierWindow/2) (default 3rd from last day) in the incoming dataframe if outputType=1 (one list element), otherwise the first+offsetDays to last day-floor(outlierWindow/2) (multiple list elements, one per day) if outputType=0.

Note

Prefer 60+ days of sample_mean values to get a good estimate of the long term sample_mean standard deviation. After initial testing on stations in the IU network, a metric value > 10 appears to be indicative of a DC offset shift (this may vary across stations or networks and larger values may be preferred as indications of a potential station issue).

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

getSingleValueMetrics

DCOffsetTimesMetric *DC Offset Detection*

Description

The DCOffsetTimesMetric() function returns times where a shift in the signal mean is detected.

Usage

```
DCOffsetTimesMetric(st, windowSecs, incrementSecs, threshold)
```

Arguments

st	a Stream object containing a seismic signal
windowSecs	chunk size (secs) used in DCOffset calculations (default=1800)
incrementSecs	increment (secs) for starttime of sequential chunks (default=windowSecs/2)
threshold	threshold used in the detection metric (default=0.9)

Details

Conceptually, this algorithm asserts: If the difference in means between sequential chunks of seismic signal is greater than the typical std dev of a chunk then this marks a DC offset shift.

Details of the algorithm are as follows

```
# Merge all traces in the time period, filling gaps with missing values
# Break up the signal into windowSecs chunks spaced incrementSecs apart
# For each chunk calculate:
#     signal mean, signal standard deviation
# Resulting mean and std dev arrays are of length 47 for 24 hours of signal
# Metric = abs(lagged difference of chunk means) / mean(chunk std devs)
# DC offset = times when Metric > threshold
```

Value

A list with a single `MultipleTimeValueMetric` object is returned.

Note

The denominator of this metric was tested with both `mean(chunk std devs)` and with `median(chunk std devs)` to identify a "typical" value for the chunk standard deviation. It was found that using `median` resulted false offset detects whenever there was a large seismic signal in an otherwise lo-noise signal.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get a signal with a DC offset problem
starttime <- as.POSIXct("2012-10-26",tz="GMT")
endtime <- starttime + 2*24*3600
st <- getDataselect(iris,"IU","TARA","00","BHZ",starttime,endtime)

# Calculate the metric
metricList <- DCOffsetTimesMetric(st)

# Extract values from the first element of the list
offsetTimes <- metricList[[1]]@values

# Plot the signal and mark locations where a DC offset was detected
plot(st)
abline(v=offsetTimes,col='red')

## End(Not run)
```

gapsMetric *Gaps and overlaps in a signal*

Description

The gapsMetric() function calculates metrics associated with gaps and overlaps in a seismic signal, *i.e.* when st consists of more than one Trace.

Usage

```
gapsMetric(st)
```

Arguments

st	a Stream object containing a seismic signal

Details

This function uses the output of the getGaps method of Stream objects to calculate the following metrics:

num_gaps: number of gaps found in st

max_gap: legnth of maximum gap (sec) found in st

num_overlaps: number of overlaps found in st

max_overlap: legnth of maximum overlap (sec) found in st

percent_availability: percentage of total requested time for which a signal is available

The requestedStarttime and requestedEndtime slots for the Stream are used to determine gaps before the start of the first or after the end of the last Trace in the Stream.

Value

A list of SingleValueMetric objects is returned.

Note

See the seismic package for documentation on Stream objects and the getDataselect method.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Calculate the gaps metrics and show the results
metricList <- gapsMetric(st)
dummy <- lapply(metricList, show)

## End(Not run)
```

GeneralValueMetric-class

Class "GeneralValueMetric"

Description

A container for metrics consisting of a vector of numeric values. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS).

Objects from the Class

Objects can be created by calls of the form:

`new("GeneralValueMetric",sncl q,starttime,endtime,metricName,elementNames,elementValues,valueStrings`

Lists of `GeneralValueMetric` objects are returned by various metrics functions in this package.

Slots

`sncl q`**:** Object of class "character": SNCLQ identifier.

`starttime`**:** Object of class "POSIXct": Start time.

`endtime`**:** Object of class "POSIXct": End time.

`metricName`**:** Object of class "character": Name of the metric.

`elementNames`**:** Object of class "character": Names of the elements storing the metric values (default="x").

`elementValues`**:** Object of class "numeric": Numeric values.

`valueStrings`**:** Object of class "character": String representations of the numeric values.

`quality_flag`**:** Object of class "numeric": Quality flag.

`quality_flagString`**:** Object of class "character": String representation of quality flag.

Methods

show `signature(object = "GeneralValueMetric")`: Prettyprints the information in the `GeneralValueMetric`

Note

The `starttime` and `endtime` slots are typically associated with the *user requested* times which may not match up with the `starttime` associated with the first `Trace` and the `endtime` associated with last `Trace` in the `Stream` object being analyzed. This ensures that metrics results for a single time period but covering many stations or channels will have the same date range and improves performance of the BSS which expects XML of the following form:

```
<measurements>
  <date start='2012-02-10T00:00:00.000' end='2012-02-10T09:20:00.000'>
    <target snclq='N.S.L.C1.Q'>
      <EXAMPLE>
        <x value="1"/>
        <x value="2"/>
        <x value="3"/>
        <x value="4"/>
      </EXAMPLE>
    </target>
  </date>
</measurements>
```

The `quality_flag` is an optional value available for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

Author(s)

Jonathan Callahan <jonathan.s.callahan@gmail.com>

`getBssMetricList` *Retrieve measurements XML from the MUSTANG BSS and convert them to a metricList*

Description

The `getBssMetricList` method makes a request of the MUSTANG BSS (Backend Storage System) and returns a list of `_Metric` objects.

Usage

```
getBssMetricList(obj, network, station, location, channel,
                 starttime, endtime, metricName, url)
```

Arguments

obj	an IrisClient object
network	a character string with the two letter seismic network code
station	a character string with the station code
location	a character string with the location code
channel	a character string with the three letter channel code
starttime	a POSIXct class specifying the starttime (GMT)
endtime	a POSIXct class specifying the endtime (GMT)
metricName	a character string identifying the name of the metric stored in the BSS
url	optional url of the BSS measurements service

Details

This method calls on getMetricsXml to communicate with the BSS and obtain an XML reponse. This response is then processed and used to create _Metric objects which are returned as a metricList.

Error returns from the BSS will stop evaluation and throw an error message.

Value

A list of _Metric objects is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

getMetricsXml

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2014-01-24", tz="GMT")
endtime <- as.POSIXct("2014-01-25", tz="GMT")

# Get the metricList
metricList <- getBssMetricList(iris,"AK","PIN","","",starttime,endtime,
                               metricName="sample_mean")
show(metricList)

## End(Not run)
```

`getGeneralValueMetrics` *Retrieve measurements from the MUSTANG BSS*

Description

The `getGeneralValueMetrics` method of the `IrisClient` makes a request of the MUSTANG database and returns a dataframe containing metrics measurments.

Usage

```
getGeneralValueMetrics(obj, network, station, location, channel,
                           starttime, endtime, metricName, ...)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code, can be "" for wildcard all
`channel`	a character string with the three letter channel code, can be "" for wildcard all
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`metricName`	a character string containing one or more comma separated metric names
`...`	optional arguments `constraint` a character string containing value constraints `url` optional url of the MUSTANG measurements service

Details

A blank location code should be specified as `location="--"`; Using `location=""` will return all location codes.

The default MUSTANG measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

Data returned from MUSTANG are converted into an R dataframe.

The optional `constraint` parameter is used to add constraints to the query as defined in the MUSTANG measurements web service documentation. Any string passed in with the `constraint` parameter will be appended to the request url following an ampersand.

Error returns from the BSS will stop evaluation and generate an error message.

Value

A dataframe with the following columns:

`~metricName~, value, additional values,  snclq, starttime, endtime, loadtime`

The `loadtime` column contains the time at which this record was loaded into the database.

The dataframe rows will be sorted by metricName and increasing starttime.

Author(s)

Jonathan Callahan <jonathan.s.callahan@gmail.com>

See Also

`createBssUrl`, `getPsdMetrics`

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2016-08-01", tz="GMT")
endtime <- starttime + 30*24*3600
metricName <- "sample_max,sample_mean,orientation_check"

# Get the measurement dataframe
juneStats <- getGeneralValueMetrics(iris,"IU","ANMO","","BH[12Z]",
                                        starttime,endtime,metricName)

print(juneStats)

## End(Not run)
```

`getMetricFunctionMetadata`

Return JSON Metadata for Metric Functions

Description

Returns a JSON formatted string with metric function metadata. This string is needed by the python-based ISPAQ command-line utility developed by IRIS DMC.

Usage

```
getMetricFunctionMetadata()
```

Value

JSON formatted string containing metric function metadata.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

`getMetricsXml` *Retrieve measurements XML from the MUSTANG BSS*

Description

The `getMetricsXml` method makes a request of the MUSTANG BSS (Backend Storage System) and returns a character string with the response XML.

Usage

```
getMetricsXml(obj, network, station, location, channel,
                   starttime, endtime, metricName, url)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code
`channel`	a character string with the three letter channel code
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`metricName`	a character string identifying the name of the metric stored in the BSS
`url`	optional url of the BSS measurements service

Details

The default BSS measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

This method returns raw XML which is not that useful by itself. Users should instead use the getBssMetricList method which calls this function and returns a list `_Metric` objects.

Error returns from the BSS will stop evaluation and throw an error message.

Value

A character string with the XML response from the BSS is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

getBssMetricList

Examples

```
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the measurement XML
xml <- getMetricsXml(iris,"AK","PIN","","BHZ",
                          starttime,endtime,metricName="sample_mean",
                          url="http://service.iris.edu/mustang/measurements/1/query?")
```

---

getMustangMetrics *Retrieve measurements from the MUSTANG BSS*

Description

The getMustangMetrics method of the IrisClient makes a request of the MUSTANG database and returns a dataframe containing metrics measurements. This function is an alias of the getGeneralValueMetrics function.

Usage

```
getMustangMetrics(obj, network, station, location, channel,
                            starttime, endtime, metricName, ...)
```

Arguments

obj	an IrisClient object
network	a character string with the two letter seismic network code
station	a character string with the station code
location	a character string with the location code, can be "" for wildcard all
channel	a character string with the three letter channel code, can be "" for wildcard all
starttime	a POSIXct class specifying the starttime (GMT)
endtime	a POSIXct class specifying the endtime (GMT)
metricName	a character string containing one or more comma separated metric names
...	optional arguments constraint a character string containing value constraints url optional url of the MUSTANG measurements service

Details

A blank location code should be specified as `location="--"`; Using `location=""` will return all location codes.

The default MUSTANG measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

Data returned from MUSTANG are converted into an R dataframe.

The optional `constraint` parameter is used to add constraints to the query as defined in the MUSTANG measurements web service documentation. Any string passed in with the `constraint` parameter will be appended to the request url following an ampersand.

Error returns from the BSS will stop evaluation and generate an error message.

Value

A dataframe with the following columns:

`~metricName~, value, additional values, snclq, starttime, endtime, loadtime`

The `loadtime` column contains the time at which this record was loaded into the database.

The dataframe rows will be sorted by metricName and increasing starttime.

Note

The database was originally populated with a version of this package that always assigned quality to be 'B'. Later versions obtained the quality from the miniSEED packet (typically 'M'). Because of this it is possible to have duplicate entries that only differ in the Q part of their snclq. To avoid double counting, when the webservice return contains two records whose only difference is the quality code portion of the of the snclq, only the record with the later loaddate will be used in the dataframe.

Author(s)

Jonathan Callahan <jonathan.s.callahan@gmail.com>

See Also

createBssUrl, getPsdMetrics

Examples

```
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2016-08-01", tz="GMT")
endtime <- starttime + 30*24*3600
metricName <- "sample_max,sample_mean,orientation_check"

# Get the measurement dataframe
juneStats <- getMustangMetrics(iris,"IU","ANMO","","BH[12Z]",
```

```
                                        starttime,endtime,metricName)

print(juneStats)
```

`getPsdMetrics` *Retrieve measurements from the MUSTANG BSS*

Description

The `getPsdMetrics` method of the `IrisClient` makes a request of the MUSTANG BSS (Backend Storage System) and returns a dataframe containing instrument corrected Power Spectral Density (PSD) measurements.

Usage

```
getPsdMetrics(obj, network, station, location, channel, starttime, endtime, url)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code
`channel`	a character string with the three letter channel code
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`url`	optional url of the BSS measurements service

Details

The default BSS measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/noise-psd/1/query?`

Data returned from the BSS are converted into an R dataframe.

Error returns from the BSS will stop evaluation and generate an error message.

Value

A dataframe with the following columns:

`target, starttime, endtime, frequency, power`

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

getBssMetricList

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the measurement XML
psdDF <- getPsdMetrics(iris,"AK","PIN","","BHZ", starttime,endtime)

## End(Not run)
```

---

getSingleValueMetrics *Retrieve measurements from the MUSTANG BSS*

Description

The getSingleValueMetrics method of the IrisClient makes a request of the MUSTANG database and returns a dataframe containing metrics that are stored as single values, e.g. sample_max, sample_min, etc..

Usage

```
getSingleValueMetrics(obj, network, station, location, channel,
                          starttime, endtime, metricName, constraint, url)
```

Arguments

obj	an IrisClient object
network	a character string with the two letter seismic network code
station	a character string with the station code
location	a character string with the location code
channel	a character string with the three letter channel code
starttime	a POSIXct class specifying the starttime (GMT)
endtime	a POSIXct class specifying the endtime (GMT)
metricName	a character string containing one or more comma separated metric names
constraint	a character string containing value constraints
url	optional url of the MUSTANG measurements service

Details

A blank location code should be specified as `location="--"`; Using `location=""` will return all location codes.

The default MUSTANG measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

Data returned from MUSTANG are converted into an R dataframe.

The optional `constraint` parameter is used to add constraints to the query as defined in the MUSTANG measurements web service documentation. Any string passed in with the `constraint` parameter will be appended to the request url following an ampersand.

Error returns from the BSS will stop evaluation and generate an error message.

Most MUSTANG metrics are single valued and can be retrieved with `getSingleValueMetrics()`. Examples of multi-valued metrics that cannot be returned with this function include "asl_coherence", "orientation_check", and "transfer_function".

Value

A dataframe with the following columns:

`~metricName~, value, snclq, starttime, endtime, loadtime`

The `loadtime` column contains the time at which this record was loaded into the database.

The dataframe rows will be sorted by increasing starttime.

The structure of this dataframe is appropriate for use with the **ggplot2** plotting package.

Note

The database was originally populated with a version of this package that always assigned quality to be 'B'. Later versions obtained the quality from the miniSEED packet (typically 'M'). Because of this it is possible to have duplicate entries that only differ in the Q part of their snclq. To avoid double counting, when the webservice return contains two records whose only difference is the quality code portion of the of the snclq, only the record with the later loaddate will be used in the dataframe.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

`createBssUrl`, `getPsdMetrics`

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2013-06-01", tz="GMT")
endtime <- starttime + 30*24*3600
metricName <- "sample_max,sample_min,sample_mean"

# Get the measurement dataframe
juneStats <- getSingleValueMetrics(iris,"IU","ANMO","00","BHZ",
                                   starttime,endtime,metricName)

head(juneStats)

# Simple ggplot2 plot
#library(ggplot2)
#p <- ggplot(juneStats, aes(x=starttime,y=value, color=as.factor(metricName))) +
#     geom_step()

#print(p)

## End(Not run)
```

`maxRangeMetric` *Maximum daily sample range calculated over a rolling window*

Description

This metric calculates the difference between the largest and smallest sample value in a 5-minute rolling window and returns the largest value encountered within a 24-hour timespan.

Usage

```
maxRangeMetric(st,
               window=300,
               increment=150)
```

Arguments

`st`	a `Stream` object containing a seismic signal
`window`	number of seconds over which to evaluate the minimum and maximum sample values
`increment`	number of seconds to advance the window for each max_range calculation

Details

For a time series passed as a `Stream` object, this function calculates differences between largest and smallest amplitudes in a series of (default) 300-second windows, incrementing the window by (default) 150 seconds for each difference calculated. It reports the largest difference as the **max_range**.

Value

The function returns a list:

- `m1` = list of `max_range` metric objects

Author(s)

Gillian Sharer <gillian@iris.washington.edu>

See Also

SingleValueMetric

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2019-08-01",tz="GMT")
endtime <- as.POSIXct("2019-08-02",tz="GMT")

# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)

# Calculate the max_range metric
tempList <- maxRangeMetric(st)

## End(Not run)
```

---

metricList2DF *Convert a MetricList into a Tidy Dataframe*

Description

The `metricList2DF` function converts a list of `SingleValueMetrics` into a "tidy" dataframe with one value per row..

Usage

```
metricList2DF(metricList)
```

Arguments

metricList	a list of SingleValueMetric objects

Details

Metrics functions return lists of SingleValueMetric objects. A long metricList may be built up by appending the results of different metrics functions or the same metrics function operating on different seismic signals. A metricList generated by any of the MUSTANG Rscripts can be stored as an .RData file and reloaded for examination.

A metricList may contain values for many different metrics. This function creates a single "tidy" dataframe with the following colulmns: metricName,value,snclq,starttime,endtime,qualityFlag.

Value

A dataframe with one row per metric measurement.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

SingleValueMetric-class, metricList2Xml

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
client <- new("IrisClient")

# Get the waveforms
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st1 <- getDataselect(client,"AK","PIN","","BHE",starttime,endtime)
st2 <- getDataselect(client,"AK","PIN","","BHN",starttime,endtime)
st3 <- getDataselect(client,"AK","PIN","","BHZ",starttime,endtime)

# Calculate metrics and append them to the metricList
metricList <- stateOfHealthMetric(st1)
metricList <- append(metricList, basicStatsMetric(st1))
metricList <- append(metricList, basicStatsMetric(st2))
metricList <- append(metricList, basicStatsMetric(st3))

# Create dataframe
metricDF <- metricList2DF(metricList)
head(metricDF)

## End(Not run)
```

metricList2DFList *Conver a metricList into a list of dataframes*

Description

The metricList2DFList function converts a list of SingleValueMetrics into a list of dataframes, one per named metric.

Usage

```
metricList2DFList(metricList)
```

Arguments

metricList	a list of SingleValueMetric objects

Details

Metrics functions return lists of SingleValueMetric objects. A long metricList may be built up by appending the results of different metrics functions or the same metrics function operating on different seismic signals. A metricList generated by any of the MUSTANG Rscripts can be stored as an .RData file and reloaded for examination.

A metricList may contain values for many different metrics. This function creates a separate dataframe for each metricName found in the metricList. As each dataframe is created, values associated with that metric are stored in a column named after the metric. Individual dataframes are stored in the returned list with their own name: metricName_DF.

Value

A character string with BSS formatted XML is returned.

Note

metricList2DFList is deprecated. Please use metricList2DF.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

SingleValueMetric-class, metricList2DF, metricList2Xml

`metricList2Xml` *Create XML for the BSS*

Description

The `metricList2Xml` function converts a list of `SingleValueMetrics` or `GeneralValueMetric` into an XML structure appropriate for submitting to the MUSTANG Backend Storage System (BSS).

Usage

```
metricList2Xml(metricList)
```

Arguments

`metricList`	a list of `SingleValueMetric` or `GeneralValueMetric` objects.

Details

Metrics functions return lists of `SingleValueMetric` or `GeneralValueMetric` objects. A long `metricList` may be built up by appending the results of different metrics functions or the same metrics function operating on different seismic signals. The list may only contain a single class (`SingleValueMetric` cannot be mixed with `GeneralValueMetric` objects). These metrics can be submitted to the BSS in a standardized XML format. (see SingleValueMetric-class)

Value

A character string with BSS formatted XML is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Apply a metric and show the results
metricList <- stateOfHealthMetric(st)
metricList <- append(metricList, basicStatsMetric(st))
bssXml <- metricList2Xml(metricList)

## End(Not run)
```

MultipleTimeValueMetric-class
Class "MultipleTimeValueMetric"

Description

A container for metrics consisting of a vector of POSIXct datetimes. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS).

Objects from the Class

Objects can be created by calls of the form:

new("MultipleTimeValueMetric",snclq,starttime,endtime,metricName,values)

Lists of MultipleTimeValueMetric objects are returned by various metrics functions in this package.

Slots

snclq: Object of class "character": SNCLQ identifier.

metricName: Object of class "character": Name of the metric.

elementName: Object of class "character": Name of the datetime element (default="t").

starttime: Object of class "POSIXct": Start time.

endtime: Object of class "POSIXct": End time.

values: Object of class "POSIXct": Datetime values.

valueStrings: Object of class "character": String representations of the datetime values.

quality_flag: Object of class "numeric": Quality flag.

quality_flagString: Object of class "character": String representation of quality flag.

Methods

show signature(object = "MultipleTimeValueMetric"): Prettyprints the information in the MultipleTimeValueMetric

Note

The starttime and endtime slots are typically associated with the *user requested* times which may not match up with the starttime associated with the first Trace and the endtime associated with last Trace in the Stream object being analyzed. This ensures that metrics results for a single time period but covering many stations or channels will have the same date range and improves performance of the BSS which expects XML of the following form:

```
<measurements>
  <date start='2012-02-10T00:00:00.000' end='2012-02-10T09:20:00.000'>
    <target snclq='N.S.L.C1.Q'>
```

```
      <up_down_times>
        <t value="2012-02-10T00:00:00.000"/>
        <t value="2012-02-10T00:01:00.000"/>
        <t value="2012-02-10T00:02:00.000"/>
        <t value="2012-02-10T00:03:00.000"/>
      </up_down_times>
    </target>
  </date>
</measurements>
```

The quality_flag is an optional value available for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

upDownTimesMetric

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Make sure we're working with a single snclq
unique_ids <- uniqueIds(st)
if (length(unique_ids) > 1) {
  stop(paste("meanMetric: Stream has",unique_ids,"unique identifiers"))
}
snclq <- unique_ids[1]

# get the upDownTimes with a minimum signal length and minimum gap (secs)
upDownTimes <- getUpDownTimes(st, min_signal=30, min_gap=60)

# Create and return a MultipleTimeValue metric from the upDownTimes
m <- new("MultipleTimeValueMetric", snclq=snclq,
         starttime=starttime, endtime=endtime,
         metricName="up_down_times", values=upDownTimes)

# Show the results
show(m)

## End(Not run)
```

PSDMetric *Power Spectral Density of a signal*

Description

The PSDMetric() function performs spectral analysis on a seismic signal and returns 'PSD' metrics with discretized spectral components as well as other metrics based on PSDs.

Usage

```
PSDMetric(st,
          linLoPeriod=4/(st@traces[[1]]@stats@sampling_rate),
          linHiPeriod=100,
          evalresp=NULL,
          noCorrection=FALSE)
```

Arguments

st	a Stream object containing a seismic signal
linLoPeriod	low end of the period band use for calculating the linear dead channel metric
linHiPeriod	high end of the period band use for calculating the linear dead channel metric
evalresp	dataframe of freq, amp, phase information matching output of getEvalresp, optional
noCorrection	boolean (default=FALSE), TRUE=only generate list of PSDs uncorrected for instrument response; FALSE=generate list of uncorrected PSDs, list of corrected PSDs, dataframe of PDF values,and PSD-derived metrics

Details

This function calculates average power spectra for a seismic signal as described in the McNamara paper. See the McNamaraPSD method of Stream objects in the **IRISSeismic** package for details.

If optional evalresp dataframe is not supplied, the code will call getEvalresp to obtain response information from webservices.

Uncorrected spectral density values are returned in spectrumMetricList in units of dB.

Instrument response corrected spectral density values are returned in correctedPsdDF in units of dB.

Probability Density Function (PDF) histogram values are returned in pdfDF.

Other metrics calculated from the PSDs are returned in svMetricList. These metrics are:

pct_above_nhnm – "percent above New High Noise Model" Percentage of PSD values that are above the New High Noise Model for their frequency. Only frequencies less than the sample_rate/3 are considered to avoid instrument response effects as you approach the nyquist frequency. This value is calculated over the entire time period.

pct_below_nlnm – "percent below New Low Noise Model" Percentage of PSD values that are below the New Low Noise Model for their frequency. Only frequencies less than the sample_rate/3 are considered to avoid instrument response effects as you approach the nyquist frequency. This value is calculated over the entire time period.

dead_channel_lin – "dead channel metric - linear fit" A "dead channel" metric is calculated from the mean of all the PSDs generated. (Typically 47 for a 24 hour period.) Values of the PSD mean line over the band (linLoPeriod:linHiPeriod) are fit to a line. The `dead_channel_lin` metric is the standard deviation of the fit residuals. Lower numbers indicate a better fit and a higher likelihood that the mean PSD is linear – an indication of a "dead channel".

Note: The dead_channel_exp metric has been removed.

Value

A list of lists is returned containing:

- `spectrumMetricList` = list of `SpectrumMetric` objects
- `correctedPsdDF` = dataframe of starttime, endtime, frequency (Hz), power (dB) values
- `pdfDF` = dataframe of frequency (Hz), power (dB), hits (count) values
- `svMetricList` = list of `SingleValueMetric` objects:
  - `pct_above_nhnm`
  - `pct_below_nlnm`
  - `dead_channel_lin`

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

Observations and Modeling of Seismic Background Noise (Peterson 1993).

See Also

`SpectrumMetric` `SingleValueMetric`

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# NOTE:  The following trace has 1.728 million points.
# NOTE:  Downloading and calculating PSD may take a few seconds.
starttime <- as.POSIXct("2010-02-27",tz="GMT")
endtime <- as.POSIXct("2010-02-28",tz="GMT")
```

```
# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)

# Calculate the PSD metric and show the SingleValueMetric results
listOfLists <- PSDMetric(st)
svMetricList <- listOfLists[['svMetricList']]

dummy <- lapply(svMetricList, show)

## End(Not run)
```

`sampleRateChannelMetric` *Sample rate consistency between miniSEED and metadata*

Description

The sampleRateChannelMetric() function compares the miniSEED sample rate with the sample rate stored in the metadata channel.

Usage

```
sampleRateChannelMetric(st,
                channel_pct=1,
                chan_rate=NULL)
```

Arguments

`st`	a `Stream` object containing a seismic signal
`channel_pct`	percentage by which the miniSEED and channel sample rates must agree to be considered a match
`chan_rate`	metadata channel sample rate from miniSEED blockette 52, stationXML, or other metadata representation <Channel:SampleRate> element, optional

Details

This function retrieves the sample rate of the first trace from a `Stream` object and compares it to the metadata channel sample rate passed as `chan_rate` to see whether both sample rates agree within `channel_pct` percent. If chan_rate is not provided, the code will retrieve a sample rate from IRIS web services.

The sampleRateChannelMetric function calculates and returns the following metrics:

sample_rate_chan – "agreement between daily miniSEED and metadata channel sample rates"
A boolean measurement that returns 0 if miniSEED and Channel sample rates agree within 1%, or 1 if they disagree.

Value

A list of lists is returned containing:

- m1 = list of sample_rate_channel metric objects

Author(s)

Mary Templeton <met@iris.washington.edu>

See Also

SingleValueMetric

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2019-08-01",tz="GMT")
endtime <- as.POSIXct("2019-08-02",tz="GMT")

# Get channel-level metadata, sample rate and normalizaton frequency
meta <- IRISSeismic::getChannel(iris, "IU","ANMO","00","BHZ",starttime,endtime)
chan_rate <- meta$samplerate

# Get the waveform
st <- IRISSeismic::getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)

# Calculate the sample rate metrics
list1 <- sampleRateChannelMetric(st,channel_pct=1,chan_rate)

## End(Not run)
```

---

sampleRateRespMetric *Sample rate consistency between miniSEED and metadata*

Description

The sampleRateRespMetric() function compares the miniSEED sample rate with the sample rate derived from the high-frequency corner of the channel's amplitude response.

Usage

```
sampleRateRespMetric(st,
                     resp_pct=15,
             norm_freq=NULL,
                     evalresp=NULL)
```

Arguments

st	a Stream object containing a seismic signal
resp_pct	percentage by which the miniSEED and response-derived sample rates must agree to be considered a match
norm_freq	the normalization frequency at which the stationXML InstrumentSensitivity or dataless Stage 0 Sensitivity is valid, optional
evalresp	dataframe of freq, amp, phase information matching output of getEvalresp, optional

Details

Next the function retrieves the instrument response that corresponds with the start of the miniSEED time series, from frequencies one decade below the norm_freq through one decade above the miniSEED sampling frequency. The difference of the amplitude values,normalized for frequency spacing, are then scanned to find the first steep rolloff. The frequency associated with the maximum difference in the rolloff is stored as the empirical Nyquist frequency and multiplied by two to give the empirical response-derived sample rate. The function then compares this sample rate with the miniSEED sample rate to see whether both rates agree within resp_pct percent. The default percentage of 15 there is significant variations across instruments. If norm_freq or evalresp values are not provided, the code will retrieve values from IRIS web services.

The sampleRateMetric function calculates and returns the following metrics:

sample_rate_resp – "agreement between daily miniSEED and response-derived sample rates"
A boolean measurement that returns 0 if miniSEED and Response-derived sample rates agree within 15%, or 1 if they disagree. Response-derived sample rates assume that the high-frequency amplitude rolloff is ~85% of the Nyquist frequency.

Value

A list of lists is returned containing:

- m1 = list of sample_rate_resp metric objects

Author(s)

Mary Templeton <met@iris.washington.edu>

See Also

SingleValueMetric

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2019-08-01",tz="GMT")
endtime <- as.POSIXct("2019-08-02",tz="GMT")
```

```
# Get channel-level metadata, sample rate and normalizaton frequency
meta <- IRISSeismic::getChannel(iris, "IU","ANMO","00","BHZ",starttime,endtime)
norm_freq <- meta$scalefreq

# Get the waveform
st <- IRISSeismic::getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)

# Calculate the sample rate metrics
list1 <- sampleRateRespMetric(st,resp_pct=15,norm_freq)

## End(Not run)
```

`saveMetricList` *Save a MetricList as RData or XML*

Description

The saveMetricList() function allows metrcis to be saved as either .RData files or as XML. The XML format is the same as that used by the IRIS DMC MUSTANG database for metric submission.

Usage

```
saveMetricList(metricList, id=Sys.getpid(), rdata=FALSE)
```

Arguments

`metricList`	list of SingleValueMetric objects
`id`	ID to be used when generating output files
`rdata`	optional flag to save the incoming `metricList` as a .RData file

Details

The `saveMetricList` function saves a list of `SingleValueMetrics` as a .RData binary file or converts the list into the XML format expected by the MUSTANG database submission process. This XML format is human readable and can be used to spot check results of metrics calculations.

Value

The automatically generated filename is returned invisibly.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

`SingleValueMetric-class`, `metricList2Xml`, `getMetricsXml`, `getBssMetricList`,

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Apply a metric and show the results
metricList <- stateOfHealthMetric(st)
metricList <- append(metricList, basicStatsMetric(st))
saveMetricList(metricList,id='AK.PIN..BHZ')

## End(Not run)
```

SingleValueMetric-class

Class "SingleValueMetric"

Description

A container for metrics results and associated metadata. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS). This has been superceded by `GeneralValueMetric` and is no longer in use.

Objects from the Class

Objects can be created by calls of the form:

`new("SingleValueMetric",snclq,starttime,endtime,metricName,value)`

Lists of `SingleValueMetric` objects are returned by various metrics functions in this package.

Slots

`snclq`: Object of class "`character`": SNCLQ identifier.

`metricName`: Object of class "`character`": Name of the metric.

`starttime`: Object of class "`POSIXct`": Start time.

`endtime`: Object of class "`POSIXct`": End time.

`valueName`: Object of class "`character`": Name of the XML value identifier (default="`value`").

`value`: Object of class "`numeric`": Metric value.

`valueString`: Object of class "`character`": String representation of the metric value.

`quality_flag`: Object of class "`numeric`": Quality flag.

`quality_flagString`: Object of class "`character`": String representation of quality flag.

`attributeName`: Object of class "`character`": Name of one or more optional attributes.

`attributeValueString`: Object of class "`character`": String representation of one or more attribute values.

Methods

show `signature(object = "SingleValueMetric")`: Prettyprints the information in the `SingleValueMetric`

Note

The `starttime` and `endtime` slots are typically associated with the *user requested* times which may not match up with the `starttime` associated with the first `Trace` and the `endtime` associated with last `Trace` in the `Stream` object being analyzed. This ensures that metrics results for a single time period but covering many stations or channels will have the same date range and improves performance of the BSS which expects XML of the following form:

```
<measurements>
  <date start='2012-02-10T00:00:00.000' end='2012-02-10T09:20:00.000'>
    <target snclq='N.S.L.C1.Q'>
      <example value='1.0'/>
    </target>
    <target snclq='N.S.L.C2.Q'>
      <example value='2.0'/>
    </target>
    <target snclq='N.S.L.C3.Q'>
      <example value='3.0'/>
    </target>
  </date>
  </date>
</measurements>
```

The `quality_flag` is an optional value available for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

For an IRIS/DMC specific example, the `station_completeness` metric obtains a list of available channels for a station from the availability web service and compares this list with the list of `percent_availability` metrics for this station stored in the MUSTANG BSS. In the case of the `station_completeness` metric, the `quality_flag` is set to the number of channels that should be available but for whom no `percent_availability` measure is obtained form the BSS.

The `attributeName` and `attributeValueString` slots can be used to store additional attributes associated with a metric values. For example, the `max_stalta` value for a seismic trace can be calculated and a metric can be created that contains this value and another attribute with a string representation of the time at which this maximum occurred.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")
```

```
# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Apply a metric and show the results
metricList <- basicStatsMetric(st)
show(metricList[[1]])

## End(Not run)
```

SNRMetric *Signal to Noise Ratio*

Description

The SNRMetric() function calculates the Signal-to-Noise Ratio of a seismic trace by one of several named algorithms.

Usage

```
SNRMetric(st, algorithm, windowSecs)
```

Arguments

st	a Stream object containing a seismic signal
algorithm	a named algorithm to use for calculating SNR (default="splitWindow")
windowSecs	width (seconds) of the full window used in SNR calculations (default=60)

Details

Seismic signals in the Stream must be without gaps, *i.e.* contained within a single Trace.

algorithm="splitWindow"

This algorithm uses the midpoint of the seismic signal as the border between noise to the left of the midpoint and signal to the right. The value for signal-to-noise is just the rmsVariance calculated for windowSecs/2 seconds of data to the right of the midpoint divided by the rmsVariance for windowSecs/2 seconds of data to the left of the midpoint.

No other algorithms have been vetted at this point.

Value

A list with a single SingleValueMetric object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
 ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get an hour long waveform centered on a big quake
starttime <- as.POSIXct("2010-02-27 06:16:15",tz="GMT")
endtime <- as.POSIXct("2010-02-27 07:16:15",tz="GMT")
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
tr <- st@traces[[1]]

# Calculate the SNR metric and show the results
metricList <- SNRMetric(st)
dummy <- lapply(metricList, show)

## End(Not run)
```

SpectrumMetric-class *Class* "SpectrumMetric"

Description

A container for metrics consisting of discrete spectra. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS).

Objects from the Class

Objects can be created by calls of the form:

`new("SpectrumMetric",snclq,starttime,endtime,metricName,freqs,amps,phases)`

Slots

`snclq`**:** Object of class `"character"`: SNCLQ identifier.

`metricName`**:** Object of class `"character"`: Name of the metric.

`elementName`**:** Object of class `"character"`: Name of the datetime element (default=`"t"`).

`starttime`**:** Object of class `"POSIXct"`: Start time.

`endtime`**:** Object of class `"POSIXct"`: End time.

`freqs`**:** Object of class `"numeric"`: Frequency values.

`freqStrings`**:** Object of class `"character"`: String representations of the frequency values.

`amps`**:** Object of class `"numeric"`: Amplitude values.

`ampStrings`**:** Object of class `"character"`: String representations of the amplitude values.

`phases`**:** Object of class `"numeric"`: Phase values.

`phaseStrings`**:** Object of class `"character"`: String representations of the phase values.

`quality_flag`**:** Object of class `"numeric"`: Quality flag.

`quality_flagString`**:** Object of class `"character"`: String representation of quality flag.

Methods

show `signature(object = "SpectrumMetric")`: Prettyprints the information in the `SpectrumMetric`

Note

The `quality_flag` is an optional value available for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

`spectrumMetric2Xml` *Convert a SpectrumMetric into XML for the BSS*

Description

The `spectrumMetric2Xml` function converts a `SpectrumMetric` into an XML structure appropriate for submitting to the MUSTANG Backend Storage System (BSS).

Usage

```
spectrumMetric2Xml(metricList)
```

Arguments

`metricList`	a list of `SpectrumMetric` objects

Value

A character string with BSS formatted XML is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# NOTE:  The following trace has 1.728 million points.
# NOTE:  Downloading and calculating PSD may take a while.
starttime <- as.POSIXct("2010-02-27",tz="GMT")
endtime <- as.POSIXct("2010-02-28",tz="GMT")

# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
```

```
# Make sure we're working with a single snclq
unique_ids <- uniqueIds(st)
if (length(unique_ids) > 1) {
  stop(paste("PSDMetric: Stream has",unique_ids,"unique identifiers"))
}
snclq <- unique_ids[1]

# Calculate and plot the Power Spectral Density
psdList <- psdList(st)

# Create a Spectrum metric list
spectrumMetricList <- list()
index <- 1
for (psd in psdList) {
  spectrumMetricList[[index]] <- new("SpectrumMetric", snclq=snclq,
                                     starttime=psd$starttime, endtime=psd$endtime,
                                     metricName="psd", freqs=psd$freq,
                                     amps=psd$spec, phases=psd$freq*0)
  index <- index + 1
}

# Show the XML version of the metric
bssXml <- spectrumMetric2Xml(spectrumMetricList)
cat(bssXml)

## End(Not run)
```

`spikesMetric` *Find spikes using a rolling Hampel filter*

Description

The spikesMetric() function determines the number of spikes in a seismic `Stream`.

Usage

```
spikesMetric(st, windowSize=41, thresholdMin=10, selectivity=NA, fixedThreshold=TRUE)
```

Arguments

`st`	a `Stream` object containing a seismic signal
`windowSize`	The window size to roll over (default=`41`)
`thresholdMin`	Initial value for outlier detection (default=`10.0`)
`selectivity`	Numeric factor [0-1] used in determining outliers, or NA if fixedThreshold=TRUE (default=`NA`)
`fixedThreshold`	TRUE or FALSE, set the threshold=thresholdMin and ignore selectivity (default=`TRUE`)

Details

This function uses the output of the `findOutliers()` function in the **seismicRoll** package to calculate the number of 'spikes' containing outliers.

The `thresholdMin` level is similar to a sigma value for normally distributed data. Hampel filter values above 6.0 indicate a data value that is extremely unlikely to be part of a normal distribution (~ 1/500 million) and therefore very likely to be an outlier. By choosing a relatively large value for `thresholdMin` we make it less likely that we will generate false positives. False positives can include high frequency environmental noise.

The `selectivity` is a value between 0 and 1 and is used to generate an appropriate threshold for outlier detection based on the statistics of the incoming data. A lower value for `selectivity` will result in more outliers while a value closer to 1.0 will result in fewer. The code ignores selectivity if `fixedThreshold=TRUE`.

The `fixedThreshold` is a logical `TRUE` or `FALSE`. If `TRUE`, then the threshold is set to `thresholdMin`. If `FALSE`, then the threshold is set to maximum value of the `roll_hample()` function output multiplied by the `selectivity`.

The total count of spikes reflects the number of outlier data points that are separated by at least one non-outlier data point. Each individual spike may contain more than one data point.

Value

A list of `SingleValueMetric` objects is returned.

Note

The `thresholdMin` parameter is sensitive to the data sampling rate. The default value of 10 seems to work well with sampling rates of 10 Hz or higher ('B..' or 'H..' channels). For 'L..' channels with a sampling rate of 1 Hz `thresholdMin=12.0` or larger may be more appropriate.

More testing of spiky signals at different resolutions is needed.

See the **seismicRoll** package for documentation on the findOutliers() function.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2013-01-03 15:00:00", tz="GMT")
endtime <- starttime + 3600 * 3
st <- getDataselect(iris,"IU","RAO","10","BHZ",starttime,endtime)

# Calculate the gaps metrics and show the results
metricList <- spikesMetric(st)
dummy <- show(metricList)
```

```
## End(Not run)
```

STALTAMetric *Maximum STA/LTA of a signal*

Description

The STALTAMetric() function calculates the maximum of STA/LTA over the incoming seismic signal.

Usage

```
STALTAMetric(st, staSecs, ltaSecs, increment, algorithm)
```

Arguments

st	a Stream object containing a seismic signal
staSecs	length of the **s**hort term averaging window in seconds (default=3)
ltaSecs	length of the **l**ong term averaging window in seconds (default=30)
algorithm	algorithm to be used (default="classic_LR")
increment	increment used when sliding the averaging windows to the next location (default=1)

Details

Currently supported algorithms include:

- "classic_RR"
- "classic_LR"
- "EarleAndShearer_envelope"

This metric applies the STALTA method of Trace objects to every Trace in st with the following parameter settings:

- demean=TRUE
- detrend=TRUE
- taper=0.0

The final metric value is the maximum STALTA value found in any Trace in this Stream.

Further details are given in the documentation for STALTA.Trace().

Value

A list with a single SingleValueMetric object is returned. The metric name is max_stalta.

Note

The STALTA method of Trace objects returns a numeric vector of STA/LTA values that has the same length as the signal data. This is a moderately time consuming operation. By comparison, finding the maximum value of this vector of STA/LTA values is very fast.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

First break picking (Wikipedia)

Automatic time-picking of first arrivals on noisy microseismic data (Wong et. al. 2009)

Automatic first-breaks picking: New strategies and algorithms (Sabbione and Velis 2010))

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-02-12",tz="GMT")
endtime <- as.POSIXct("2012-02-13",tz="GMT")
st <- getDataselect(iris,"AK","GHO","","BHN",starttime,endtime)

# Calculate the STA/LTA metric and show the results
metricList <- STALTAMetric(st)
dummy <- lapply(metricList, show)

## End(Not run)
```

---

stateOfHealthMetric *State of Health metrics*

Description

The stateOfHealthMetric function extracts accumulated miniSEED quality flags and a measure of timing quality associated with the incoming seismic signal.

Usage

```
stateOfHealthMetric(st)
```

Arguments

st	a Stream object containing a seismic signal

Details

The miniSEED flags and timing_qual values are described in the SEED manual (http://www.fdsn.org/seed_manual/SEEDManual_V2.4.pdf).

Each Stream object contains "accumulators" with counts of the number of times each bit flag was set during the parsing of a miniSEED file. Metrics are reported for a subset of these flags as show in the code snippet below:

```
# act_flags
calibration_signal <- st@act_flags[1]
timing_correction <- st@act_flags[2]
event_begin <- st@act_flags[3]
event_end <- st@act_flags[4]
event_in_progress <- st@act_flags[7]

# io_flags
clock_locked <- st@io_flags[6]

# dq_flags
amplifier_saturation <- st@dq_flags[1]
digitizer_clipping <- st@dq_flags[2]
spikes <- st@dq_flags[3]
glitches <- st@dq_flags[4]
missing_padded_data <- st@dq_flags[5]
telemetry_sync_error <- st@dq_flags[6]
digital_filter_charging <- st@dq_flags[7]
```

An additional "timing quality" metric gives the average value for the timing_qual value associated with each block of miniSEED data.

Value

A list of SingleValueMetric objects is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Generate State of Health metrics and show the results
```

```
metricList <- stateOfHealthMetric(st)
dummy <- lapply(metricList, show)

## End(Not run)
```

`timesMetric2Xml` *Create XML for the BSS*

Description

The `timesMetric2Xml` function converts a `MultipleTimeValueMetric` into an XML structure appropriate for submitting to the MUSTANG Backend Storage System (BSS).

Usage

```
timesMetric2Xml(metric)
```

Arguments

`metric`	a `MultipleTimeValueMetric` object

Value

A character string with BSS formatted XML is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Make sure we're working with a single snclq
unique_ids <- uniqueIds(st)
if (length(unique_ids) > 1) {
  stop(paste("meanMetric: Stream has",unique_ids,"unique identifiers"))
}
snclq <- unique_ids[1]

# get the upDownTimes with a minimum signal length and minimum gap (secs)
upDownTimes <- getUpDownTimes(st, min_signal=30, min_gap=60)
```

```
# Create and return a MultipleTimeValue metric from the upDownTimes
m <- new("MultipleTimeValueMetric", snclq=snclq, starttime=starttime,
         endtime=endtime, metricName="up_down_times", values=upDownTimes)

# Show the XML version of the metric
bssXml <- timesMetric2Xml(m)
cat(bssXml)

## End(Not run)
```

`transferFunctionMetric` *Cross-spectral comparison*

Description

The transferFunctionMetric() function calculates metrics that assess the relationship between two SNCLs with the same network, station and channel but separate locations. When seismometers are working properly, the transfer function amplitude and phase will match similar values calculated from the instrument responses.

This function calculates the transfer function from data in the incoming streams. Response information is then obtained from the evalresp web service.

Usage

```
transferFunctionMetric(st1, st2, evalresp1, evalresp2)
```

Arguments

`st1`	a `Stream` object containing a seismic signal
`st2`	a `Stream` object containing a seismic signal
`evalresp1`	a `data frame` containing an amplitude and phase spectrum
`evalresp2`	a `data frame` containing an amplitude and phase spectrum

Details

Details of the algorithm are as follows

```
# compute complex cross-spectrum of traces x and y ==> Pxx, Pxy, Pyy
# calculate transfer function values:
#     Txy(f) = Pxy(f) / Pxx(f)
#     dataGain <- Mod(Txy)
#     dataPhase <- Arg(Txy)
#
# calculate avgDataGain and avgDataPhase values for periods of 5-7s
#
# calculate the corresponding response amplitude ratio and phase difference:
```

```
#     request responses for x and y
#     respGain = respGainy(f) / respGainx(f)
#     respPhase = respPhasey(f) - respPhasex(f)
#
# calculate avgRespGain and avgRespPhase values for periods of 5-7s
#
# calculate metrics:
#     gain_ratio = avgDataGain / avgRespGain
#     hase_diff = avgDataPhase - avgRespPhase
#     ms_coherence = |Pxy|^2 / (Pxx*Pyy)
```

Value

A list with a single `SingleValueMetric` object is returned. The metric name is `transfer_function` and it has three attributes:

- `gain_ratio` – reasonableness of cross-spectral amplitude between `st1` and `st2`
- `phase_diff` – reasonableness of cross-spectral phase between `st1` and `st2`
- `ms_coherence` – mean square coherence between `st1` and `st2`

These values can be interpreted as follows:

Whenever `ms_coherence` `~=` `1.0`, properly functioning seismometers should have:

- `gain_raio` `~=` `1.0`
- `phase_diff` < `10.0` (degrees)

Note

Seismic streams passed to transferFunctionMetric() must have the same network, station and channel and must cover the same time range. The two channels should also have values of azimuth and dip within five degrees of each other. If sampling rates differ and one is a multiple of the other, the stream with the higher sampling rate will be decimated to match the lower sampling rate.

The metricList generated for these two-channel metrics will have a SNCL code of the form: `N.S.L1:L2.C.Q`.

Author(s)

Jonathan Callahan <`jonathan@mazamascience.com`> (R code), Mary Templeton <`met@iris.washington.edu`> (algorithm)

Examples

```
  ## Not run:
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# Get seismic data
starttime <- as.POSIXct("2011-05-01", tz="GMT")
endtime <- starttime + 3600
# These values are specific to two 40 sps channels
minfreq <- 0.005255603
```

```
maxfreq <- 19.7403
nfreq <- 96
units <- 'def'
output <- 'fap'

st1 <- getDataselect(iris,"CI","PASC","00","BHZ",starttime,endtime)
st2 <- getDataselect(iris,"CI","PASC","10","BHZ",starttime,endtime)
evalresp1 <- getEvalresp(iris, "CI", "PASC", "00", "BHZ", starttime,
                         minfreq, maxfreq, nfreq, units, output)
evalresp2 <- getEvalresp(iris, "CI", "PASC", "10", "BHZ", starttime,
                         minfreq, maxfreq, nfreq, units, output)

# Calculate metrics
metricList <- transferFunctionMetric(st1,st2,evalresp1,evalresp2)
print(metricList)

## End(Not run)
```

`upDownTimesMetric` *Up/down times for a channel*

Description

The upDownTimesMetric() function determines the times at which data collection starts and stops within a seismic `Stream`.

Usage

```
upDownTimesMetric(st, min_signal, min_gap)
```

Arguments

`st`	a `Stream` object containing a seismic signal
`min_signal`	minimum duration of a `Trace` in seconds (default=`30`)
`min_gap`	minimum gap in seconds (default=`60`)

Details

This function uses the output of the `getUpDownTimes` method of `Stream` objects.

Value

A list with a single `MultipleTimeValueMetric` object is returned.

Note

See the `seismic` package for documentation on `Stream` objects and the `getDataselect` method.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

getUpDownTimes

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Create the upDownTimesMetric, ignoring Traces < 3 minutes and gaps of < 5 minutes
metricList <- upDownTimesMetric(st, min_signal=180, min_gap=300)

## End(Not run)
```

Index