Package 'gmsp'

November 11, 2026

Type Package

Title Ground Motion Signal Processing

Version 1.1.1

Description Implements short-time Fourier transform (STFT) based processing of strong-motion time series: time-grid regularisation, STFT-window and anti-alias-resampling strategy selection, edge tapering, and frequency-domain integration and differentiation, mapping a single input (acceleration, velocity, or displacement) to a consistent triplet under a chosen analysis bandwidth. Also provides intrinsic-mode-function decomposition via empirical mode decomposition (EMD), ensemble EMD (EEMD), and variational mode decomposition (VMD) with optional band-rule filtering; elastic single-degree-of-freedom (SDOF) domain-normalised acceleration, velocity, and displacement response spectra from their corresponding time-series domains by exact state-space integration; intensity measures including peak, root-mean-square (RMS), Arias intensity, significant-duration, cumulative absolute velocity, mean period, and the derived indices earthquake destructiveness potential (EPI) and power-of-input (PDI); and D50 and D100 horizontal response spectra. Methods: Huang et al. (1998) <doi:10.1098/rspa.1998.0193>, Wu and Huang (2009) <doi:10.1142/S1793536909000047>, Dragomiretskiy and Zosso (2014) <doi:10.1109/TSP.2013.2288675>, Boore (2010) <doi:10.1785/0120090400>. An optional indexing layer parses provider files in formats including 'PEER' 'NGA-West2' 'AT2', 'CESMD' 'V2'/'V2c', 'NWZ' 'V2A', Geological Survey of Canada 'TR', 'IGP'/'UCR' 'AC' variants, and generic two-column ASCII text, normalises components, writes per-record CSV (comma-separated values) and JSON (JavaScript Object Notation) pairs, and computes per-record intensity tables.

License AGPL (>= 3)

URL https://kanameishi.github.io/gmsp/

Encoding UTF-8

Depends R (>= 4.1.0)

Imports data.table, digest, expm, jsonlite, Matrix, osqp, parallel, pracma, seewave, signal, stats, tools, utils, xemd

Config/roxygen2/version 8.1.1

Suggests bit64, knitr, rmarkdown

VignetteBuilder knitr

SystemRequirements tar (for archived record metadata)

NeedsCompilation no

Author Alejandro Daniel Verri [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-8535-1170>)

Maintainer Alejandro Daniel Verri <averri@fi.uba.ar>

Repository CRAN

Date/Publication 2026-11-11 22:40:02 UTC

Contents

accumulateIntensityTSL . . . 3
alignComponents . . . 4
archiveRawOwner . . . 5
assessCompatibility . . . 5
assessDistances . . . 7
assessParsers . . . 8
assessSite . . . 9
buildRawFileTable . . . 10
buildRawIntensityTable . . . 11
buildRawRecordTable . . . 12
buildTSL . . . 13
computeSpectraTSL . . . 16
decomposeRecords . . . 17
describeModule . . . 18
exportProject . . . 19
extractRecord . . . 20
fitModalFactor . . . 21
fitRecordFactor . . . 23
lengthenPSW . . . 25
lengthenTSW . . . 26
mapComponents . . . 27
matchProject . . . 28
measureIntensitiesTSL . . . 30
normalizeTSL . . . 32
parseRecord . . . 33
processProject . . . 34
rankProject . . . 35
readProviderFile . . . 38
readRecordTSL . . . 39
reconstructRecords . . . 40
selectProject . . . 41
selectRecords . . . 43
widenIML . . . 44
widenPSL . . . 45
widenTSL . . . 46
writeSelection . . . 47

Index **49**

`accumulateIntensityTSL` *Accumulate normalized intensity histories*

Description

Computes cumulative prefixes from acceleration rows and divides each record/component history by its own maximum.

Usage

```
accumulateIntensityTSL(TSL, by, ID, unitsSource, unitsTarget)
```

Arguments

`TSL`	A table with columns `OCID,ID,t,s` and the metadata selected by `by`. Rows with ID equal to AT supply acceleration samples; other signal domains are excluded from the result. Time t is in seconds and s is numeric acceleration in the source length basis per second squared.
`by`	Character vector of record metadata columns; `character()` identifies one record without metadata. A retained `Units` column declares the target length basis and must agree with `unitsTarget`.
`ID`	Cumulative quantity: exactly `"AI"`, `"CAV"` or `"CAV5"`.
`unitsSource, unitsTarget`	Required source and target length bases: `"mm"`, `"cm"` or `"m"`, matched case-insensitively. Acceleration units g and gal, and forms such as mm/s, are rejected.

Details

At least one AT history is required. Each history requires at least two finite numeric samples and unique times; rows are sorted by t before calculation. Missing columns, duplicate keys and invalid units cause errors.

Prefix sums use the mean time step and include the first sample. AI sums squared acceleration with the Arias scale; CAV sums absolute acceleration. CAV5 includes absolute acceleration at least 0.05 times standard gravity. This CAV5 cutoff is the inherited gmsp definition.

The input is copied. The function reads or writes no files, uses no random numbers and changes no session options.

Value

A new data.table with columns `by,OCID,ID,t,s`, ordered by metadata, component and time. ID contains the requested quantity and s contains dimensionless ratios. A zero or nonfinite cumulative maximum produces NA_real_ throughout that history, including arithmetic overflow from finite samples.

See Also

measureIntensitiesTSL()

Examples

```
TSL <- data.table::data.table(
  RecordID = "R1", OCID = "H1", ID = "AT",
  t = seq(0, 1, by = 0.25), s = c(0, 400, 500, 400, 0))
accumulateIntensityTSL(TSL, by = "RecordID", ID = "CAV5",
                       unitsSource = "mm", unitsTarget = "mm")
```

---

alignComponents *Equalize component sample counts*

Description

Pads shorter components with trailing zeros for max alignment, or truncates longer components for min alignment. Alignment operates by row count without interpolating time.

Usage

```
alignComponents(data, align = "max")
```

Arguments

data	A long data.table with t, OCID, s, DIR for one record.
align	Either max or min.

Details

Padding uses the interval between the first two samples of the first OCID. Empty input, unknown alignment and a nonfinite or nonpositive interval needed for padding raise errors.

Value

A list with DT, the aligned table, and NP, the final sample count per component. The input table remains unchanged.

Examples

```
Record <- data.table::data.table(
  t = c(0, 0.01, 0.02, 0, 0.01), OCID = c("H1", "H1", "H1", "H2", "H2"),
  DIR = c("H1", "H1", "H1", "H2", "H2"), s = c(1, 2, 3, 4, 5))
Padded <- alignComponents(Record, align = "max")
Padded$NP
print(Padded$DT)
alignComponents(Record, align = "min")$NP
```

archiveRawOwner *Archive a station's provider files*

Description

Creates raw.owner.tar.gz from raw.owner and removes the source directory after confirming that the archive exists and has a nonempty listing.

Usage

```
archiveRawOwner(path)
```

Arguments

path	Station folder containing raw.owner.

Details

The working directory is restored on exit. The archive check is a listing check and does not compare extracted file bytes. A failed creation or listing leaves the source directory and removes the archive.

Value

An invisible logical value: TRUE after archiving, FALSE when the source directory is absent or the destination exists, and NA for a treated archive creation or listing failure.

Examples

```
Station <- tempfile("archiveRawOwner")
dir.create(file.path(Station, "raw.owner"), recursive = TRUE)
writeLines(c("header", "Event, date, station, H1", "units",
             "NPTS= 4, DT= 0.01 SEC", "1.0 2.0 3.0 4.0"),
           file.path(Station, "raw.owner", "H1.AT2"))
print(archiveRawOwner(Station))
list.files(Station)
utils::untar(file.path(Station, "raw.owner.tar.gz"), list = TRUE)
unlink(Station, recursive = TRUE)
```

assessCompatibility *Assess compatibility of a scaled suite*

Description

Calculates record diagnostics and a suite summary against a target band, preserving the input tables and performing no file access.

Usage

```
assessCompatibility(PSW, target, scale, OCID, factorMin, factorMax)
```

Arguments

PSW	A data.table of scaled spectra with RecordID, Tn in seconds and PSA.<OCID> in acceleration units. Available OwnerID, EventID, StationID and ProcessID keys are retained.
target	A table with finite, unique, non-negative Tn and finite positive Sa.low, Sa.mean, Sa.high in the same acceleration units as PSW, ordered from low through mean to high. Exactly one row satisfies abs(Tn) <= 1e-12, the PGA anchor.
scale	A data.table with unique RecordID and numeric scaleFactor; every evaluated record requires a finite factor.
OCID	Component identifier: "H1", "H2" or "UP".
factorMin	Finite non-negative lower record-factor bound.
factorMax	Upper record-factor bound, at least factorMin; Inf permits an unbounded upper factor.

Details

Duplicate RecordID/Tn keys in PSW are errors. Calculations use matching periods without interpolation; an empty intersection is an error, and nTn reports its size for each record. When a record lacks the PGA period, its PGA value, ratios and status remain NA. Band containment uses 1e-6 * pmax(1, abs(Sa.high)); relative violations use the band limits without that tolerance. Factor-bound flags use an absolute difference below 1e-8.

Value

A list of data.tables: record contains record keys, nTn, PGA values and target ratios, relative PSA RMSE, band containment and violations, scaleFactor, scaleAtMin, scaleAtMax and PGA status, ordered by decreasing PGA.final. summary contains record counts by PGA status and factor bounds, and median or extreme values of record RMSE, containment and violation.

See Also

fitRecordFactor()

Examples

```
Target <- data.table::data.table(Tn = c(0, 1), Sa.low = 0.50 * 9806.65,
  Sa.mean = 0.84 * 9806.65, Sa.high = 0.95 * 9806.65)
PSW <- data.table::data.table(RecordID = rep(c("R1", "R2"), each = 2),
  Tn = c(0, 1, 0, 1), PSA.H1 = c(7000, 8500, 4000, 9800))
Scale <- data.table::data.table(RecordID = c("R1", "R2"), scaleFactor = c(1, 1.6))
Compatibility <- assessCompatibility(PSW, target = Target, scale = Scale,
  OCID = "H1", factorMin = 0.5, factorMax = 1.6)
print(Compatibility$record[, c("RecordID", "PGA.final", "PSA.inside",
                               "scaleAtMax", "status")])
```

```
print(Compatibility$summary[, c("records", "nPGA.inside", "nPGA.below",
                                "nScaleAtMax")])
```

`assessDistances` *Assess coordinate and distance metadata*

Description

Returns flagged rows with the first applicable reason, preserving the input.

Usage

```
assessDistances(data, repiOutlier)
```

Arguments

`data`	Table with numeric EventLatitude, EventLongitude, StationLatitude, StationLongitude, EventDepth, Repi and Rhyp. Repi and Rhyp use kilometres. Additional columns are retained.
`repiOutlier`	Finite scalar cutoff in kilometres for Repi.

Details

Reasons follow this precedence: latNA, lonNA, eventLatOOR, stationLatOOR, eventLonOOR, stationLonOOR, depthNeg, repiOut, geomBad. Absolute coordinate limits are 90 for latitude and 180 for longitude. geomBad uses Rhyp less than Repi minus 0.01 kilometres; this is the diagnostic rule. Missing coordinates are flagged, while missing depth or distances do not by themselves produce a reason. Missing required columns, nonnumeric columns and an invalid cutoff are input errors. The operation performs no file access.

Value

A data.table containing flagged rows in input order, with Reason.

See Also

`assessSite()`

Examples

```
Records <- data.frame(RecordID = c("R1", "R2", "R3", "R4"),
  EventLatitude = c(10, NA, 10, 10), EventLongitude = 20,
  StationLatitude = 10.4, StationLongitude = 20.1, EventDepth = 30,
  Repi = c(45, 45, 450, 45), Rhyp = c(54, 54, 451, 40))
assessDistances(Records, repiOutlier = 300)
```

assessParsers *Assess provider parsing by station*

Description

Reads each station of an owner through parseRecord and reports its status.

Usage

```
assessParsers(records, owner, path)
```

Arguments

records	Record metadata with OwnerID, EventID, StationID and the FileID rows required by parseRecord.
owner	One nonempty OwnerID string.
path	Nonempty path to the records root, passed to parseRecord.

Details

Status is OK for a returned parser result, with its row and column counts; FAIL for an error; WARN for a warning. The latter statuses retain conditionMessage as reason. Missing grouping columns or invalid owner/path arguments are input errors. The operation writes no extracted records.

Value

A data.table with OwnerID, EventID, StationID, status and reason, in first-appearance station order. An owner without rows returns an empty table with these columns.

See Also

parseRecord()

Examples

```
Root <- tempfile("assessParsers")
Station <- file.path(Root, "NGAW", "E1", "S1", "raw.owner")
dir.create(Station, recursive = TRUE)
writeLines(c("header", "Event, date, station, H1", "units",
             "NPTS= 4, DT= 0.01 SEC", "1.0 2.0 3.0 4.0"),
           file.path(Station, "H1.AT2"))
Records <- data.table::data.table(OwnerID = "NGAW", EventID = "E1",
  StationID = c("S1", "S2"), FileID = "H1.AT2")
Status <- assessParsers(Records, owner = "NGAW", path = Root)
print(Status[, c("StationID", "status")])
unlink(Root, recursive = TRUE)
```

`assessSite` *Assess station velocity metadata*

Description

Returns flagged rows with the first applicable station-velocity reason, preserving the input.

Usage

```
assessSite(data, vs30Low, vs30High)
```

Arguments

`data`	Table with numeric StationVs30 in metres per second. Additional columns are retained.
`vs30Low`, `vs30High`	Finite scalar cuts, ordered from low to high, in metres per second.

Details

Missing StationVs30 gives vs30NA; values below vs30Low give vs30Low, and values above vs30High give vs30High, in that precedence. Values equal to a cut are not flagged. A missing or nonnumeric StationVs30 column and invalid cuts are input errors. The operation performs no file access.

Value

A data.table containing flagged rows in input order, with Reason.

See Also

`assessDistances()`

Examples

```
Stations <- data.frame(StationID = c("S1", "S2", "S3", "S4"),
                       StationVs30 = c(NA, 150, 400, 2000))
assessSite(Stations, vs30Low = 180, vs30High = 1500)
```

`buildRawFileTable` *Build provider file indexes*

Description

Reads record.json from each station's raw.owner directory or its raw.owner.tar.gz archive and writes one RawFileTable CSV per owner.

Usage

```
buildRawFileTable(pathRecords, pathIndex, owners = NULL)
```

Arguments

`pathRecords`	Records root containing owner, event and station folders.
`pathIndex`	Existing destination folder for `RawFileTable.<OwnerID>.csv`.
`owners`	Owner identifiers; NULL scans immediate directories of pathRecords.

Details

Columns are OwnerID, EventID, StationID, ComponentID, FileID, NP, dt, Fs, Units, HP, LP and isArray, ordered by event, station and component. Missing metadata become typed NA and extra fields are dropped. isArray indicates more than three component rows in the provider metadata.

A station failure retains its cause and path. Valid rows are published; an error of class modulePartialPublication then carries failures, the partial result and destination path. Counts are not returned as complete after such a failure. Archive metadata are read without extracting files.

Value

An invisible named list of row counts for the owners written.

Examples

```
Root <- tempfile("buildRawFileTable")
Station <- file.path(Root, "records", "NGAW", "E1", "S1", "raw.owner")
Index <- file.path(Root, "index")
dir.create(Station, recursive = TRUE)
dir.create(Index)
Metadata <- list(Event = list(EventID = "E1"), Station = list(StationID = "S1"),
  Record = data.frame(ComponentID = c("HNE", "HNN", "HNZ"),
                      FileID = c("HNE.AT2", "HNN.AT2", "HNZ.AT2"),
                      NP = 4L, dt = 0.01, Fs = 100, Units = "g"))
jsonlite::write_json(Metadata, file.path(Station, "record.json"), auto_unbox = TRUE)
Counts <- buildRawFileTable(file.path(Root, "records"), pathIndex = Index)
str(Counts)
data.table::fread(file.path(Index, "RawFileTable.NGAW.csv"))
unlink(Root, recursive = TRUE)
```

`buildRawIntensityTable` *Build raw acceleration intensity indexes*

Description

Measures raw AT records and writes `RawIntensityTable.<OwnerID>.csv`, with one row per RecordID and DIR and the provider OCID retained.

Usage

```
buildRawIntensityTable(
  pathRecords,
  pathIndex,
  owners = NULL,
  incremental = TRUE,
  force = FALSE
)
```

Arguments

`pathRecords`	Records root containing owner, event and station folders.
`pathIndex`	Existing destination folder for the index files.
`owners`	Owner identifiers; NULL scans immediate directories of pathRecords.
`incremental`	Reuses records already present in the index when TRUE.
`force`	Ignores the incremental cache when TRUE.

Details

The index contains RecordID, DIR, OCID, AI, AId, AIu, ARMS, ATn, ATo, AZC, CAV, CAV5, D0575, D0595, D2080, Dmax, EPI, Fs, NP, PDI, PGA, TmA and dt, ordered by RecordID and DIR. Values use a millimetre length basis. Duration and period fields use one decimal place and dt uses four; the remaining measure fields are rounded to integers.

VT and DT stations are omitted with a warning naming the station and quantity. A station failure retains its cause and path. Valid rows are published; an error of class modulePartialPublication then carries failures, the partial result and destination path. Existing files are replaced when the owner has new intensity rows.

Value

An invisible named list of row counts for the owners written.

Examples

```
Root <- tempfile("buildRawIntensityTable")
Raw <- file.path(Root, "records", "NGAW", "E1", "S1", "raw")
Index <- file.path(Root, "index")
dir.create(Raw, recursive = TRUE)
dir.create(Index)
Time <- seq(0, 5, by = 0.01)
Envelope <- sin(pi * Time / max(Time))^2
data.table::fwrite(data.table::data.table(
  HNE = 1200 * Envelope * sin(2 * pi * 1.5 * Time),
  HNN = 800 * Envelope * sin(2 * pi * 2.5 * Time),
  HNZ = 400 * Envelope * sin(2 * pi * 4 * Time)), file.path(Raw, "AT.R1.csv"))
jsonlite::write_json(list(RecordID = "R1", DIR = c("H1", "H2", "UP"),
  OCID = c("HNE", "HNN", "HNZ"), dt = 0.01), file.path(Raw, "AT.R1.json"),
  auto_unbox = TRUE)
Counts <- buildRawIntensityTable(file.path(Root, "records"), pathIndex = Index)
str(Counts)
Intensity <- data.table::fread(file.path(Index, "RawIntensityTable.NGAW.csv"))
print(Intensity[, c("RecordID", "DIR", "OCID", "PGA", "AI", "D0595")])
unlink(Root, recursive = TRUE)
```

---

buildRawRecordTable *Build raw record indexes*

Description

Reads AT, VT and DT raw JSON sidecars and writes `RawRecordTable.<OwnerID>.csv` for each owner with records.

Usage

```
buildRawRecordTable(pathRecords, pathIndex, owners = NULL)
```

Arguments

`pathRecords`	Records root containing owner, event and station folders.
`pathIndex`	Existing destination folder for the index files.
`owners`	Owner identifiers; NULL scans immediate directories of pathRecords.

Details

Columns are RecordID, EventID, StationID, OwnerID, NP, Fs and pad, ordered by event and station. NP is the maximum pre-alignment count from the sidecar, and pad is the difference between maximum and minimum counts. Under truncation, NP does not describe the CSV row count.

A station failure retains its cause and path. Valid rows are published; an error of class modulePartialPublication then carries failures, the partial result and destination path.

Value

An invisible named list of row counts for the owners written.

Examples

```
Root <- tempfile("buildRawRecordTable")
Raw <- file.path(Root, "records", "NGAW", "E1", "S1", "raw")
Index <- file.path(Root, "index")
dir.create(Raw, recursive = TRUE)
dir.create(Index)
jsonlite::write_json(list(RecordID = "R1", OwnerID = "NGAW", EventID = "E1",
  StationID = "S1", NP = c(4, 4, 3), Fs = 100), file.path(Raw, "AT.R1.json"),
  auto_unbox = TRUE)
Counts <- buildRawRecordTable(file.path(Root, "records"), pathIndex = Index)
str(Counts)
data.table::fread(file.path(Index, "RawRecordTable.NGAW.csv"))
unlink(Root, recursive = TRUE)
```

---

buildTSL *Build acceleration, velocity, and displacement histories*

Description

`buildTSL()` processes one measured quantity into acceleration (`AT`), velocity (`VT`), and displacement (`DT`) on a common time grid per record. Records are identified by the explicit metadata columns in `by` and their channels by `OCID`.

Usage

```
buildTSL(
  TSL,
  by,
  unitsSource,
  unitsTarget,
  fmax,
  windowLength,
  overlap,
  astop0,
  apass0,
  astopLP,
  apassLP,
  kNyq = NULL,
  isRaw = TRUE,
  flatZeros = FALSE,
  trimZeros = FALSE,
  removeMean = FALSE,
  forceRegularization = FALSE,
  includeWarnings = TRUE,
```

```
  derivate = "freq",
  lowPass = TRUE
)
```

Arguments

TSL	A data.table with columns by, OCID, ID, t, and s. The call contains one ID: "AT", "VT", or "DT". Times t are seconds and samples s are finite numeric values. Channels of each record have identical time grids and unique by,OCID,ID,t keys.
by	Character vector of metadata columns identifying a record. character() declares one record without metadata. Only these metadata columns are retained. A retained Units column contains the target length basis; a conflicting value is an error.
unitsSource	Input length basis when isRaw = TRUE: "mm", "cm", or "m"; AT also accepts "g" and "gal". Matching is case-insensitive. The quantity determines whether the units are length/s^2, length/s, or length. g uses 9806.650 mm/s^2 and gal means cm/s^2. With isRaw = FALSE, this argument is not evaluated.
unitsTarget	Required target length basis: "mm", "cm", or "m", matched case-insensitively, including with isRaw = FALSE.
fmax	Required finite positive frequency of interest in Hz.
windowLength	Required positive integer nominal STFT window length in samples. The strategy can adjust the length; the initial request is capped for AT and DT inputs according to record length. The effective STFT window must be even.
overlap	Required overlap percentage in [0,100).
astop0, apass0	Required finite positive normalized amplitude thresholds for the channel edge tapers.
astopLP, apassLP	Required finite positive magnitudes for the resampling antialias and derivative low-pass filters. The controls require astopLP < apassLP < 1.
kNyq	NULL selects the automatic sampling-frequency grid. A finite positive scalar requests round(kNyq * fmax) Hz, which must be positive.
isRaw	Logical scalar. Raw input is converted from unitsSource, regularized, and tapered. Processed input is already in unitsTarget.
flatZeros	Logical scalar enabling edge tapers for processed input. For AT input it also applies an exact amplitude-support mask.
trimZeros	Logical scalar selecting final nonzero window support. AT retains the union of channel supports; VT and DT retain their intersection.
removeMean	Logical scalar enabling removal of channel means at the processing stages of the measured quantity.
forceRegularization	Logical scalar forcing interpolation of processed input. Irregular or non-rational grids are always regularized.

`includeWarnings`
: Logical scalar enabling strategy warnings. Diagnostics do not alter the computed amplitudes.

`derivate`
: Derivative method for `VT` and `DT` inputs: `"freq"` or `"time"`, matched case-insensitively. It is not evaluated for `AT`.

`lowPass`
: Logical scalar enabling the derivative low-pass with `derivate = "freq"`. It is not evaluated for `AT`.

Details

Each channel requires at least three samples; derivatives require more than four samples on the grid used by the derivative. Frequency-domain integration and differentiation set the zero-frequency term to zero. Tapering, resampling, and removal of means affect the processed histories. The returned quantities are not exact algebraic inverses of the input.

Invalid keys, non-finite samples, incompatible channel grids, unsupported units, and invalid controls cause an error. Errors during record processing carry the record context. The function reads or writes no files, uses no random numbers, and changes no working directory, options, or environment.

Value

A new data.table with columns `by,OCID,ID,t,s`, ordered by record, channel, quantity (`AT`, `VT`, `DT`), and time. The common grid starts at zero for each record. Amplitudes use `unitsTarget`/s^2, `unitsTarget`/s, and `unitsTarget`, respectively. The input is unchanged.

See Also

`normalizeTSL()`, `widenTSL()`, `lengthenTSW()`

Examples

```
Time <- seq(0, 5, by = 0.01)
TSL <- data.table::data.table(
  OCID = "H1", ID = "AT", t = Time,
  s = 120 * sin(pi * Time / max(Time))^2 * sin(2 * pi * 1.5 * Time)
)
Histories <- buildTSL(
  TSL, by = character(), unitsSource = "mm", unitsTarget = "mm", fmax = 5,
  windowLength = 128, overlap = 75,
  astop0 = 1e-3, apass0 = 1e-3,
  astopLP = 1e-3, apassLP = 0.98,
  includeWarnings = FALSE
)
unique(Histories$ID)
```

`computeSpectraTSL` *Compute response spectra from canonical time series*

Description

Computes PSA, PSV and SD from AT, VT and DT as separate excitations. Each ordinate retains the units of its source domain. The zero-period anchor is the peak absolute value of the regularized source signal.

Usage

```
computeSpectraTSL(
  TSL,
  by,
  xi,
  Tn,
  includeD50 = FALSE,
  includeD100 = FALSE,
  nTheta
)
```

Arguments

`TSL`	A table with `OCID`, `ID`, `t`, `s` and the metadata selected by `by`. Each record/component requires AT, VT and DT, with more than two finite samples per domain and unique times. Time is in seconds.
`by`	Character vector of record metadata columns, or `character()` for one record without metadata.
`xi`	Nonempty numeric vector of finite damping ratios in `[0,1]`.
`Tn`	Nonempty numeric vector of finite positive periods in seconds. Zero is reserved for the internally added peak-value anchor.
`includeD50`	Logical scalar selecting the median rotated component.
`includeD100`	Logical scalar selecting the maximum rotated component.
`nTheta`	Number of angles in [0,pi), required when a rotated component is selected; an integer of at least two.

Details

Each domain is regularized independently to the minimum rationalized time step and a zero time origin. The oscillator starts from zero state and uses the final sample of each interval as input. PSV and SD use VT and DT excitations, respectively.

Rotations require H1 and H2 on identical input time grids in each domain. D50 uses the median with quantile type 7; D100 uses the maximum across the selected angles, including the peak-value anchor. Original components remain in the result. Vector xi rejects an input column named xi. Duplicate sample keys and invalid controls are errors. The input is copied; the operation uses no random numbers or files and does not change session options. Units are preserved without conversion.

Value

A new data.table with `by,OCID,Tn,ID,S`, ordered by metadata, component, spectral ID and period. Vector `xi` adds an xi column after the metadata and preserves the requested damping order within each record. Repeated periods and damping ratios remain repeated.

See Also

`widenPSL()`, `lengthenPSW()`

Examples

```
t <- seq(0, 1, by = 0.01)
TSL <- data.table::rbindlist(list(
  data.table::data.table(t = t, s = sin(2 * pi * t), ID = "AT", OCID = "H1"),
  data.table::data.table(t = t, s = cos(2 * pi * t), ID = "VT", OCID = "H1"),
  data.table::data.table(t = t, s = sin(pi * t), ID = "DT", OCID = "H1")))
computeSpectraTSL(TSL, by = character(), xi = 0.05, Tn = c(0.1, 0.2))
```

`decomposeRecords` *Decompose acceleration records into modes*

Description

Regularizes acceleration records and composes the native xemd decomposition tables.

Usage

```
decomposeRecords(TSL, method, control, by)
```

Arguments

`TSL`	A data.table containing t in seconds, finite real samples s, character ID and OCID, and record metadata. Only ID AT is decomposed. Present components form a nonempty subset of H1, H2, and UP. Each record has at least three unique times on a common component grid; other metadata are constant within each component.
`method`	Historical name vmd or eemd, or a name returned by xemd::listDecompositions().
`control`	Named list without duplicate names. Historical vmd requires k, alpha, tau, DC, init, tolerance and maxIterations; init must be zero. Historical eemd requires k, noise.amp, trials and an explicit sifting list. Native controls follow the selected xemd method; signal, samplingInterval, noise and noiseChannels are derived and cannot be supplied.
`by`	Distinct metadata column names identifying a record; character() declares one record.

Details

Regularization uses the smallest rationalized time difference, monoH.FC interpolation and zero origin before translating controls. Historical vmd uses zero initial centers and converts alpha by samplingInterval^2 / (8 * pi^2); its relative tolerance is explicit, and tol is not accepted. Historical eemd converts absolute noise.amp to noise.amp / sd(signal) for each component, requiring positive finite sd; it uses divisor realizations and maxModes equal to k. Conflicting maxModes is an error. sifting follows its explicit xemd stopRule and its documented defaults; wave and type5 are not translated. Historical vmd calls ascend by effective interval, then record and component; historical eemd calls ascend by record and component. Native methods follow decomposeTSL ordering. Tables are ordered by by, OCID, ID, mode or boundary, and t when present. Units are transported without conversion. Stochastic methods consume the caller's random stream. The function sets no seed, restores no RNG state, starts no parallel workers and writes no files. Inputs are preserved. Invalid data or controls cause errors; xemd failures retain their classes and group, with the moduleFailure family and operation. Native diagnostics and termination retain non-convergence and iteration limits.

Value

A named list of data.tables with signal and modes and the fields residue, fitResidual, meanNoise, diagnostics, termination, centers, maskFrequencies or boundaries returned by the method. signal contains the effective input. Mode indices and optional fields retain their native meaning.

See Also

`reconstructRecords()`, `xemd::decomposeTSL()`

Examples

```
Time <- seq(0, 9, length.out = 1000)
TSL <- data.table::data.table(RecordID = "R1", OCID = "H1", ID = "AT", t = Time,
  s = sin(pi * Time) + sin(2 * pi * Time) + sin(6 * pi * Time) + 0.5 * Time,
  Units = "mm")
Control <- list(k = 3, alpha = 2000, tau = 0.5, DC = FALSE, init = 0,
                tolerance = 1e-6, maxIterations = 98)
Decomposition <- decomposeRecords(TSL, method = "vmd", control = Control,
                                  by = "RecordID")
names(Decomposition)
print(Decomposition$centers)
head(Decomposition$modes)
```

---

describeModule *Describe the Installed Module*

Description

Return the installed identity of this module.

Usage

```
describeModule()
```

Value

A list containing package and version.

`exportProject` *Export project products*

Description

Writes selected time series and product tables, and copies available metadata and HTML files to a delivery directory.

Usage

```
exportProject(file = file.path("gmsp", "export.json"), root = getwd())
```

Arguments

`file`	Path to the JSON export contract, resolved against `root`.
`root`	Existing project directory.

Details

The contract requires `path.out` and selects `path.source` or database paths with an explicit selection. A source directory requires `TSW.csv` and `IMW.csv`; `PSW.csv` is optional. `ID` defaults to `AT` and `DIR` to `H1`. A raw database supports `AT`. An explicit `path.index` identifies a MasterIndex CSV file. HTML copies preserve subdirectories and reject duplicate relative paths within the plot or audit inputs. A nonempty destination requires `override=true`, which preserves unrelated files. The working directory is restored after completion or failure.

Value

Invisible integer zero on completion; usage, execution or partial publication failures signal conditions.

Examples

```
Root <- tempfile("exportProject")
dir.create(file.path(Root, "source"), recursive = TRUE)
dir.create(file.path(Root, "gmsp"))
data.table::fwrite(data.table::data.table(RecordID = "001", t = c(0, 0.1, 0.2),
  AT.H1 = c(0, 1, -1), AT.H2 = c(1, 0, -1)), file.path(Root, "source", "TSW.csv"))
data.table::fwrite(data.table::data.table(RecordID = "001", OCID = c("H1", "H2"),
  PGA = c(100, 200)), file.path(Root, "source", "IMW.csv"))
jsonlite::write_json(list(path = list(source = "source", out = "delivery"),
  DIR = c("H1", "H2")), file.path(Root, "gmsp", "export.json"), auto_unbox = TRUE)
exportProject(root = Root)
list.files(file.path(Root, "delivery"), recursive = TRUE)
data.table::fread(file.path(Root, "delivery", "TSW.001.AT.H1.csv"))
unlink(Root, recursive = TRUE)
```

`extractRecord` *Extract a provider record to raw files*

Description

Reads and classifies one provider record without rotation, converts its amplitudes to a millimetre length basis and writes a CSV and JSON pair.

Usage

```
extractRecord(record, path, align = "max", kind = NULL)
```

Arguments

`record`	Table for one owner, event and station, with FileID and Units; NetworkID is retained in the sidecar when supplied.
`path`	Records root containing OwnerID/EventID/StationID/raw.owner.
`align`	Alignment by sample count: "max" pads with trailing zeros; "min" truncates. Neither choice interpolates time.
`kind`	Quantity override AT, VT or DT; NULL derives the quantity from Units. An override still requires recognisable Units.

Details

The CSV retains provider OCID columns and omits time, and its lines end with LF on every system. RecordID is the first 16 characters of the MD5 digest of the CSV bytes, so it does not depend on the system. The JSON stores DIR, OCID and pre-alignment NP arrays in H1, H2, UP order; peaks are measured after alignment. Its dt and Fs use nine and six decimal places, respectively, and Units is "mm".

Existing raw contents are removed after classification and unit resolution, before alignment and writing. A later failure can leave partial products; extraction is not atomic. The supplied table is not modified.

Value

Absolute path to the written CSV, or NULL when classification or unit interpretation cannot produce a record.

Examples

```
Root <- tempfile("extractRecord")
Station <- file.path(Root, "NGAW", "E1", "S1", "raw.owner")
dir.create(Station, recursive = TRUE)
Samples <- c(HNE = "0.01 0.03 -0.02 0.01", HNN = "0.02 -0.01 0.01 0",
             HNZ = "0 0.01 -0.01 0")
for (Channel in names(Samples)) {
  writeLines(c("header", paste0("Event, date, station, ", Channel), "units",
```

```
              "NPTS= 4, DT= 0.01 SEC", Samples[[Channel]]),
            file.path(Station, paste0(Channel, ".AT2")))
}
Record <- data.table::data.table(OwnerID = "NGAW", EventID = "E1",
  StationID = "S1", FileID = paste0(names(Samples), ".AT2"), Units = "g")
CSV <- extractRecord(Record, path = Root)
basename(CSV)
data.table::fread(CSV)
Sidecar <- jsonlite::read_json(sub("[.]csv$", ".json", CSV), simplifyVector = TRUE)
str(Sidecar[c("DIR", "OCID", "NP", "PGA", "dt", "Units")])
unlink(Root, recursive = TRUE)
```

`fitModalFactor` *Fit modal factors against a response-spectrum band*

Description

Fits the retained modes of one component of one record, then restores signal energy in the final reconstruction.

Usage

```
fitModalFactor(
  IMF,
  target,
  source,
  key,
  OCID,
  xi,
  fmax,
  kNyq,
  signalControl,
  factorMin,
  factorMax,
  criterion,
  unitsSource,
  unitsTarget,
  includeResidue = TRUE
)
```

Arguments

`IMF`	Table for one record with OCID, t, signal, optional `IMF<k>` mode columns and optional residue. Time is in seconds and amplitudes use unitsSource. A mode entirely NA within a component is absent; partial NA or non-finite samples are errors.

`target` Table with unique nonnegative Tn in seconds and finite positive Sa.low, Sa.mean and Sa.high, in that order at each period. The band includes Tn = 0 and positive periods; amplitudes use unitsTarget/s^2.

`source` Wide time-series table for the same record, with its key columns, t and `AT.<OCID>`, `VT.<OCID>`, `DT.<OCID>`, in unitsSource.

`key` One-row table with RecordID, OwnerID, EventID and StationID, plus ProcessID when present in the context. Missing IMF key columns are filled from this row; conflicting values are errors.

`OCID` Component to fit: "H1", "H2" or "UP".

`xi` Positive finite damping ratio accepted by computeSpectraTSL.

`fmax` Positive finite frequency of interest in Hz.

`kNyq` Positive finite sampling-frequency multiplier.

`signalControl` Named list with windowLength, overlap, astop0, apass0, astopLP and apassLP, as in buildTSL. Optional flatZeros, trimZeros, removeMean, forceRegularization and includeWarnings retain their buildTSL defaults; duplicate or other fields are errors.

`factorMin` Finite lower modal-factor bound in (0,1].

`factorMax` Upper modal-factor bound at least one; Inf is admitted.

`criterion` Explicit xemd selection list: rule "band" with fMin and fMax in Hz, or rule "energyFraction" with minimum, and action "keep" or "remove". Band limits are finite with 0 <= fMin <= fMax; minimum is finite and nonnegative. At least one mode must remain; selection by mode name or index is not admitted.

`unitsSource`, `unitsTarget`
Source and destination length bases: "mm", "cm" or "m", matched case-insensitively. Source histories and IMF amplitudes are converted to the destination basis.

`includeResidue` Logical scalar, TRUE by default, retaining the residue when present with a fixed coefficient of one, outside the modal optimization and its factor bounds. Final energy restoration applies to the complete reconstruction, including the residue.

Details

Modes are measured and selected through xemd. The objective uses relative spectral RMSE, squared relative band violations and a 0.01 weight on mean squared log-factor movement. L-BFGS-B starts with factors equal to one and maxit = 1. Energy restoration applies after optimization.

In metrics, status is done and convergence and message retain the stats::optim result: convergence 0 is successful completion, 1 means that the iteration limit maxit was reached, and 51 and 52 are a warning and an error of L-BFGS-B explained by message. message is the optimizer text, such as NEW_X, or NA when the optimizer returns none. With maxit = 1, code 1 is the expected result and does not indicate a failure. The convergence code does not change status.

Invalid keys, controls, samples or target bands cause an error with record context. Zero reconstruction energy is an error. Dependency conditions retain their classes. Inputs are preserved; the operation reads or writes no files and uses no random numbers.

Value

A list with TSW and PSW for the fitted component; PSA with RecordID, kind (source/start/final), Tn and Sa; coeff with the record keys, IMF, b0, b, ratio, energyFactor and stageFactor; metrics with optimizer status and fit diagnostics; and eval in objective-call order. Time histories and spectra are ordered by record and time or period; coefficients follow the retained modal order, then the residue. Its row has b0, b and ratio equal to one, and stageFactor equal to energyFactor.

See Also

`fitRecordFactor()`, `assessCompatibility()`, `buildTSL()`, `xemd::selectModesTSL()`

Examples

```
Key <- data.table::data.table(RecordID = "R1", OwnerID = "A", EventID = "E1",
                              StationID = "S1")
Time <- seq(0, 4, by = 0.01)
Mode <- 1000 * sin(2 * pi * Time)
Residue <- 100 * sin(pi * Time)
IMF <- data.table::data.table(OCID = "H1", t = Time, signal = Mode + Residue,
                              IMF1 = Mode, residue = Residue)
Source <- data.table::data.table(Key, t = Time, AT.H1 = Mode + Residue,
                                 VT.H1 = 0, DT.H1 = 0)
Target <- data.table::data.table(Tn = c(0, 1, 2), Sa.low = 1000,
                                 Sa.mean = 2000, Sa.high = 3000)
Control <- list(windowLength = 128, overlap = 75, astop0 = 0.001, apass0 = 0.001,
                astopLP = 0.001, apassLP = 0.98, includeWarnings = FALSE)
Fit <- fitModalFactor(IMF, target = Target, source = Source, key = Key,
  OCID = "H1", xi = 0.05, fmax = 25, kNyq = 4, signalControl = Control,
  factorMin = 0.001, factorMax = Inf,
  criterion = list(rule = "energyFraction", minimum = 0, action = "keep"),
  unitsSource = "mm", unitsTarget = "mm")
print(Fit$coeff[, c("IMF", "b", "energyFactor", "stageFactor")])
print(Fit$PSA[Fit$PSA$kind == "final"])
print(Fit$metrics[, c("status", "convergence", "RMSE.start", "RMSE.final")])
```

---

fitRecordFactor *Fit suite record-scaling factors*

Description

Fits one bounded scale factor per record with a quadratic program against a target response-spectrum band, without modifying the input tables.

Usage

```
fitRecordFactor(
  PSA,
  target,
```

```
  records,
  factorMin,
  factorMax,
  bandWeight,
  deficitWeight
)
```

Arguments

PSA	A data.table with character RecordID and kind, numeric Tn in seconds and numeric Sa in acceleration units. The kind == "final" curves enter the fit.
target	A table with Tn, Sa.low, Sa.mean and Sa.high, in the same acceleration units as PSA. Periods are finite, non-negative and unique; ordinates are finite, positive and ordered from low through mean to high.
records	Character vector of distinct completed record identities, in the requested output order.
factorMin	Finite non-negative lower factor bound.
factorMax	Upper factor bound, at least factorMin; Inf permits an unbounded upper factor.
bandWeight	Finite non-negative weight of the record band-violation term.
deficitWeight	Finite positive weight of the suite-mean deficit term.

Details

Duplicate RecordID/Tn/kind keys are errors. Alignment retains target periods present in the final curves and requires at least two records with a finite spectral matrix. The solver uses absolute and relative tolerances of 1e-8, a maximum of 200000 iterations and polishing; the returned status is solved or solved inaccurate, and other statuses are errors. The stage RMSE is relative to Sa.mean; PGA ratio is NA when the aligned periods lack zero, and rcond is NA when its matrix calculation fails. The function calculates in memory and performs no file access.

Value

A list with scale and stage data.tables. scale contains RecordID and scaleFactor in the requested record order after alignment. stage contains one row labelled B, the osqp method and solver status, objective weights, relative RMSE, PGA ratio, band violation and scale statistics, and matrix diagnostics equations, records, rankSVD, cond2 and rcond.

See Also

fitModalFactor(), assessCompatibility()

Examples

```
Target <- data.table::data.table(Tn = c(0, 1), Sa.low = 0.50 * 9806.65,
  Sa.mean = 0.84 * 9806.65, Sa.high = 0.95 * 9806.65)
PSA <- data.table::data.table(RecordID = rep(c("R1", "R2"), each = 2),
  kind = "final", Tn = c(0, 1, 0, 1), Sa = c(7000, 8500, 4000, 6000))
Fit <- fitRecordFactor(PSA, target = Target, records = c("R1", "R2"),
```

```
  factorMin = 0.5, factorMax = 1.6, bandWeight = 2, deficitWeight = 1)
print(Fit$scale)
print(Fit$stage[, c("status", "RMSE", "pgaRatio", "insideFraction", "nAtMax")])
```

`lengthenPSW` *Lengthen wide response spectra*

Description

Converts `<ID>.<OCID>` columns to canonical long spectra without changing values, units or the supplied table.

Usage

```
lengthenPSW(PSW, by, ids = c("PSA", "PSV", "SD"))
```

Arguments

`PSW`	Wide table with `Tn`, selected metadata and spectra columns named `<ID>.<OCID>`.
`by`	Character vector of metadata columns to retain, or `character()` for one record without metadata. Remaining columns other than Tn are spectra columns.
`ids`	Preferred order of spectral IDs. Other IDs remain after these; the argument orders rows without filtering quantities.

Details

The first dot separates ID from OCID; additional dots remain part of OCID. Duplicate metadata/period keys are errors. A correctly shaped empty input retains the key columns and has no spectral rows. The projection uses no random numbers, files or session-option changes.

Value

A new data.table with `by,OCID,Tn,ID,S`, ordered by metadata, component, preferred ID and period. NA values remain explicit rows.

See Also

`widenPSL()`, `computeSpectraTSL()`

Examples

```
PSW <- data.table::data.table(RecordID = "R1", Tn = 0.1, PSA.H1 = 1)
lengthenPSW(PSW, by = "RecordID")
```

`lengthenTSW` *Lengthen wide time-series columns*

Description

Converts columns named `<ID>.<OCID>` into canonical samples without signal processing, unit conversion, or mutation of the supplied table.

Usage

```
lengthenTSW(TSW, by, ids = c("AT", "VT", "DT"))
```

Arguments

`TSW`	A table with `t` (or `ts` when t is absent), the selected metadata and one or more `<ID>.<OCID>` columns.
`by`	Character vector of metadata columns to retain; use `character()` for one record without metadata. All remaining non-time columns must be signal columns.
`ids`	Preferred quantity order; quantities not listed are retained.

Details

The first dot separates ID and OCID; additional dots in OCID remain intact. Duplicate metadata/time keys are errors. Tables carrying both t and ts are rejected. Sparse wide cells become explicit NA rows.

Value

A new data.table with columns `by`, `OCID`, `ID`, `t`, `s`, ordered by metadata, component, preferred quantity and time. NA values remain NA. Metadata and amplitudes retain their supplied units.

See Also

`widenTSL()`

Examples

```
TSW <- data.table::data.table(RecordID = "R1", t = c(0, 0.01),
                              AT.H1 = c(1, 2), VT.H1 = c(0.1, 0.2))
lengthenTSW(TSW, by = "RecordID")
```

mapComponents *Map provider channels to canonical directions*

Description

Classifies one three-component record as H1, H2 and UP while preserving provider channel identities in OCID. Optional rotation aligns the horizontal signals with their principal axes.

Usage

```
mapComponents(data, rotate, output = "long")
```

Arguments

data	A long data.table with t, OCID, s, or a wide data.table with t and three provider signal columns.
rotate	Logical scalar selecting horizontal rotation.
output	Either long or wide.

Details

Classification requires three OCID values and one recognized vertical channel. Finite horizontal peaks provide the classification fallback when covariance is unavailable. Rotation requires at least two finite horizontal pairs. The input table remains unchanged.

Value

A long data.table with t, OCID, DIR, s, or a wide data.table with t, H1, H2, UP. Both carry componentMap and rotate attributes; rotated results also carry theta in radians. Returns NULL when the record cannot be classified or rotated.

Examples

```
Time <- seq(0, 1, by = 0.01)
Record <- data.table::data.table(
  t = rep(Time, 3), OCID = rep(c("HNE", "HNN", "HNZ"), each = length(Time)),
  s = c(sin(2 * pi * 2 * Time), 0.5 * sin(2 * pi * 2 * Time + 0.3),
        0.2 * sin(2 * pi * 5 * Time)))
Wide <- mapComponents(Record, rotate = FALSE, output = "wide")
head(Wide)
attr(Wide, "componentMap")
Rotated <- mapComponents(Record, rotate = TRUE)
attr(Rotated, "theta")
head(Rotated)
```

`matchProject` *Match selected records and publish project products*

Description

Composes modal shaping, record scaling and population summaries.

Usage

```
matchProject(file = file.path("gmsp", "match.json"), root = getwd())
```

Arguments

`file`	Path to the JSON project contract, relative to root or absolute.
`root`	Existing project directory used to resolve relative paths.

Details

The contract declares id, a source folder or processed base with an explicit selection, target, shape, units, xi, fmax, kNyq, nTheta, signalControl and imf.criterion. A database selection uses path.selection or an explicit candidate.ownerID. imf.criterion accepts band or energyFraction, or exactly list(rule = "target", action = "remove"). The target rule retains mode centers in `[1/max(Tn), fmax]` Hz using positive periods of the actual target after the Tn restriction and fmax from the contract. fmax must exceed 1/max(Tn). metadata/criterion.json records the requested rule/action, effective fMin/fMax in Hz, TnMax in seconds (null for explicit rules) and origin target/explicit. Energy-fraction rules have null frequency limits. imf.includeResidue defaults to TRUE: a present residue has coefficient one, outside the modal optimization; final energy restoration includes it. signalControl supplies the buildTSL controls required by fitModalFactor. Upper modal and record-factor bounds accept the exact JSON string "Inf"; other numeric controls require numbers. units.psa must match the acceleration unit of units.target. Products use units.target. summary.probabilities is optional; when absent mean.csv keeps Tn, Sa and recordCount, the arithmetic mean that gmsp 0.8.0 wrote, and intensitySummary.csv keeps OCID, IM, mean and recordCount. When supplied, probabilities are finite, unique and within `[0,1]`; mean.csv adds q probability columns and intensitySummary.csv adds p and value. Summaries use quantile type 7 without removing missing values; duplicate population keys and nonfinite values are errors. The keys plot.audit and plot.standard are ignored since gmsp 1.0.0 with a warning, because figures are produced in the reports scaffold; no plot or audit folder is written. override defaults to FALSE; TRUE replaces known files while preserving foreign files. Failed records or writes preserve their causes and signal incomplete publication. The working directory is unchanged.

parallel is TRUE or FALSE and defaults to FALSE. A declared workers value is one positive integer, even when parallel is FALSE; other values of either key are usage errors. When workers is absent, it is the number of logical cores that parallel::detectCores reports. With parallel TRUE and more than one worker, records are fitted in up to workers concurrent R processes, never more than the selected records; otherwise they are fitted one after another in the calling R process.

In match/metrics.csv, status done marks a fitted record and status error a failed one, whose error column keeps its condition message and whose convergence and message are NA. For a fitted record,

convergence and message retain the stats::optim result of the L-BFGS-B modal fit: convergence 0 is successful completion, 1 means that the iteration limit maxit was reached, and 51 and 52 are a warning and an error of L-BFGS-B explained by message. message is the optimizer text, such as NEW_X, or NA when the optimizer returns none. The modal fit is limited to one iteration (maxit = 1), so 1 is the expected code and does not indicate a failure. The convergence code does not change status.

Value

Invisible integer zero after publication of data, metadata and match products under path.out.

Matching with the band

Modal shaping uses squared relative error to Sa.mean and squared relative violations below Sa.low or above Sa.high, together with the modal-factor movement penalty. During record scaling, shape.bandWeight weights individual record band violations, while shape.deficitWeight weights the suite-mean deficit below Sa.mean. Both stages use the target band including zero (PGA). match/target.csv publishes Tn, Sa.low, Sa.mean and Sa.high in units.psa; metadata/runMatch.json copies the contract, including target.envelope.

Target envelope

target.envelope is a required JSON object with lower and upper fields, for example `{"lower": "0.50", "upper": "0.95"}`. Each bound and target.p select an exact p label in the target table: `"mean"` or a decimal of the form `0.<digits>` strictly between 0 and 1, such as `"0.50"`. Percentages (`"50"`, `"50%"`) and p prefixes (`"p50"`) are rejected. Numeric scalars are converted to text and must satisfy the same decimal format and literal table match; `"0.5"` and `"0.50"` are different labels. The lower, central and upper labels must be distinct; duplicate object keys and target.envelope.ID are rejected. `"mean"` is valid for either bound or target.p when its rows exist and satisfy the band checks. lower, target.p and upper become Sa.low, Sa.mean and Sa.high respectively; Sa.mean denotes the selected central curve even when target.p is a quantile. The curves require identical Tn meshes containing zero (PGA) and positive periods, with finite ordinates satisfying `0 < Sa.low <= Sa.mean <= Sa.high` at every retained period. Tn is in seconds. Ordinates are read in target.units (default g) and converted to units.psa; conversions between g and mm/s2 (or mm/s^2) are supported, and identical units are retained.

See Also

`fitModalFactor()`, `fitRecordFactor()`, `assessCompatibility()`

Examples

```
Root <- tempfile("matchProject")
for (DIR in c("data", "source/data", "source/metadata")) {
  dir.create(file.path(Root, DIR), recursive = TRUE)
}
Target <- data.table::CJ(ID = "MCE", siteID = "SITE", Vs30 = 800,
  Tn = c(0, 0.01, 0.04, 1, 5, 10), p = c("0.50", "0.84", "0.95"))
Target$SaF <- as.numeric(Target$p)
saveRDS(Target, file.path(Root, "data", "MCETable.Rds"))
```

```
Keys <- data.table::data.table(RecordID = c("R1", "R2"), OwnerID = "A",
  EventID = c("E1", "E2"), StationID = "S1")
Time <- seq(0, 4, by = 0.01)
Mode <- 3000 * sin(2 * pi * Time)
Residue <- 300 * sin(pi * Time)
Signal <- Mode + Residue
Velocity <- cumsum(Signal) * 0.01
Displacement <- cumsum(Velocity) * 0.01
# R2 repeats the histories of R1 at 60 percent of their amplitude.
Rows <- rep(seq_len(nrow(Keys)), each = length(Time))
Amplitude <- c(1, 0.6)[Rows]
data.table::fwrite(data.table::data.table(Keys[Rows], t = Time,
  AT.H1 = Amplitude * Signal, AT.H2 = 0.5 * Amplitude * Signal,
  AT.UP = 0.3 * Amplitude * Signal, VT.H1 = Amplitude * Velocity,
  VT.H2 = 0.5 * Amplitude * Velocity, VT.UP = 0.3 * Amplitude * Velocity,
  DT.H1 = Amplitude * Displacement, DT.H2 = 0.5 * Amplitude * Displacement,
  DT.UP = 0.3 * Amplitude * Displacement),
  file.path(Root, "source", "data", "TSW.csv"))
data.table::fwrite(data.table::data.table(Keys[Rows], OCID = "H1", t = Time,
  signal = Amplitude * Signal, IMF1 = Amplitude * Mode,
  residue = Amplitude * Residue), file.path(Root, "source", "data", "IMF.csv"))
data.table::fwrite(data.table::data.table(Keys, status = "done"),
  file.path(Root, "source", "data", "manifest.csv"))
data.table::fwrite(Keys, file.path(Root, "source", "metadata", "MasterIndex.csv"))
data.table::fwrite(data.table::data.table(Keys, OCID = "H1"),
  file.path(Root, "source", "metadata", "OCIDMAP.csv"))
jsonlite::write_json(list(id = "demo",
  path = list(source = "source", out = "out", data = "data"),
  target = list(demand = "MCE", siteID = "SITE", Vs30 = 800, p = "0.84",
    value = "SaF", units = "g", envelope = list(lower = "0.50", upper = "0.95")),
  Tn = list(min = 0.04, max = 5),
  shape = list(modalFactor = list(min = 0.001, max = "Inf"),
    recordFactor = list(min = 0.5, max = 1.6), bandWeight = 2, deficitWeight = 1),
  units = list(source = "mm", target = "mm", psa = "mm/s2"),
  imf = list(criterion = list(rule = "target", action = "remove")),
  signalControl = list(windowLength = 128, overlap = 75, astop0 = 0.001,
    apass0 = 0.001, astopLP = 0.001, apassLP = 0.98, includeWarnings = FALSE),
  xi = 0.05, fmax = 25, kNyq = 4, nTheta = 180, parallel = FALSE),
  file.path(Root, "match.json"), auto_unbox = TRUE)
matchProject(file = "match.json", root = Root)
list.files(file.path(Root, "out", "match"))
data.table::fread(file.path(Root, "out", "match", "scale.csv"))
unlink(Root, recursive = TRUE)
```

`measureIntensitiesTSL` *Measure intensities from time histories*

Description

Computes scalar intensity measures for each record, component and signal domain after conversion to the requested length basis.

Usage

```
measureIntensitiesTSL(TSL, by, unitsSource, unitsTarget)
```

Arguments

TSL	A table with columns OCID,ID,t,s and the metadata selected by by. Time t is in seconds and s contains finite numeric samples. Each record and component contains AT alone or the complete AT, VT and DT set.
by	Character vector of record metadata columns; character() identifies one record without metadata. A retained Units column declares the target length basis and must agree with unitsTarget.
unitsSource, unitsTarget	Required source and target length bases: "mm", "cm" or "m", matched case-insensitively. Acceleration units g and gal, and forms such as mm/s, are rejected.

Details

Each series is ordered by t and requires at least two finite numeric samples with unique times. Duplicate by,OCID,ID,t keys, missing AT, incomplete domain sets and unsupported units are errors.

Arias intensity and cumulative absolute velocity use mean-step rectangular sums including the first sample. CAV5 includes samples with absolute acceleration at least 0.05 times standard gravity. Durations use the first crossings without interpolation: D0595 uses 0.05 and 0.95, D2080 uses 0.20 and 0.80, and D0575 uses 0.05 and 0.75. Zero signals return zero duration. Zero-crossing counts include adjacent zero samples when the series has more than two samples. Dmax is the last time, including a nonzero time origin. The first FFT coefficient is assigned one frequency step, and mean periods use the 0.1–25 Hz band; zero signals return zero mean period. PDI retains Inf or NaN when its zero-crossing denominator is zero.

The input is copied. The function reads or writes no files, uses no random numbers and changes no session options.

Value

A new data.table with columns by,OCID,ID,IM,value,units. Rows follow quantity order AT, VT, DT and the measure order below; record/component groups retain their first-occurrence order within each measure. Dimensional values use unitsTarget; the units column identifies the units of each measure.

Measures

AT returns AI, AIu, AId, PGA, ARMS, AZC, ATo, ATn, D0595, D2080, D0575, TmA, CAV, CAV5, NP, dt, Fs, Dmax, EPI and PDI. VT returns PGV, VRMS, VZC, VTo, VTn and TmV. DT returns PGD, DRMS, DZC, DTo, DTn and TmD.

See Also

accumulateIntensityTSL(), widenIML()

Examples

```
t <- seq(0, 1, by = 0.01)
TSL <- data.table::rbindlist(list(
  data.table::data.table(t = t, s = sin(2 * pi * t), ID = "AT", OCID = "H1"),
  data.table::data.table(t = t, s = cos(2 * pi * t), ID = "VT", OCID = "H1"),
  data.table::data.table(t = t, s = sin(pi * t), ID = "DT", OCID = "H1")))
measureIntensitiesTSL(TSL, by = character(),
                      unitsSource = "mm", unitsTarget = "mm")
```

---

normalizeTSL *Normalize all quantities by a reference peak*

Description

Applies one factor per record and component, based on the peak of AT, VT or DT. The input table is preserved, including its row order.

Usage

```
normalizeTSL(TSL, by, reference)
```

Arguments

TSL	Canonical long table with OCID, ID, t, s and metadata.
by	Character vector of record metadata columns, or character() for a single record without metadata.
reference	Reference measure: exactly "PGA", "PGV" or "PGD", corresponding to AT, VT and DT.

Details

A missing reference, nonfinite reference sample or nonpositive reference peak is an error. Samples are not omitted to compute a peak. All quantities in the group are scaled, including additional IDs.

Value

A new data.table with the same columns and rows, whose s values are divided by the reference peak in each by, OCID group. No metadata unit is reinterpreted. Other quantities use the same factor; their own peaks need not equal one.

Examples

```
t <- seq(0, 1, by = 0.01)
TSL <- data.table::rbindlist(list(
  data.table::data.table(t = t, s = 2 * sin(2 * pi * t), ID = "AT", OCID = "H1"),
  data.table::data.table(t = t, s = cos(2 * pi * t), ID = "VT", OCID = "H1"),
  data.table::data.table(t = t, s = sin(pi * t), ID = "DT", OCID = "H1")))
normalizeTSL(TSL, by = character(), reference = "PGA")
```

parseRecord *Parse provider files for one record*

Description

Reads unique FileID entries under path/OwnerID/EventID/StationID/raw.owner/ using the owner's format.

Usage

```
parseRecord(record, path)
```

Arguments

record	A data.table with OwnerID, EventID, StationID and FileID. OwnerID, EventID and StationID identify one station record.
path	Path to the records root.

Details

Missing keys, NA keys, multiple records and owners without a format raise errors before file reading. Duplicate FileID entries are read once. This operation does not add ID, convert amplitudes or write files.

Value

A long data.table with t, OCID, s in native units.

Examples

```
Root <- tempfile("parseRecord")
Station <- file.path(Root, "NGAW", "E1", "S1", "raw.owner")
dir.create(Station, recursive = TRUE)
Samples <- c(HNE = "0.01 0.03 -0.02 0.01", HNN = "0.02 -0.01 0.01 0",
             HNZ = "0 0.01 -0.01 0")
for (Channel in names(Samples)) {
  writeLines(c("header", paste0("Event, date, station, ", Channel), "units",
               "NPTS= 4, DT= 0.01 SEC", Samples[[Channel]]),
             file.path(Station, paste0(Channel, ".AT2")))
}
Record <- data.table::data.table(OwnerID = "NGAW", EventID = "E1",
  StationID = "S1", FileID = paste0(names(Samples), ".AT2"))
parseRecord(Record, path = Root)
unlink(Root, recursive = TRUE)
```

`processProject` *Process selected project records*

Description

Reads a project JSON contract and publishes record products, process tables, a signed processing recipe and execution evidence.

Usage

```
processProject(file = file.path("gmsp", "process.json"), root = getwd())
```

Arguments

`file`	Path to the JSON contract, relative to root or absolute. The default is gmsp/process.json.
`root`	Existing project directory used to resolve relative paths. The default is the caller's working directory.

Details

The contract requires processID, ordered stages, path.selection, path.stage for execution evidence, and roots for records, index and process. path.database can supply these roots; explicit paths take precedence. Scientific controls are explicit in each stage section.

The stages are gmsp, trim, imf, imfTS and psa. Raw processing produces TSW and IMW; trim replaces these products within the same run, and psa writes PSW. imf writes IMF and IMFF through decomposeRecords and the xemd mode metrics. imfTS reads IMF, selects by its explicit criterion, reintegrates through reconstructRecords and writes TSW and IMW.

imf.method accepts vmd, eemd or a native xemd decomposition name. vmd requires k, alpha, tau, DC, init, tolerance and maxIterations in imf; eemd requires k, noise.amp, trials and sifting. Native methods take their controls in imf.control. Numeric JSON arrays become numeric vectors. imfTS requires criterion, fmax, kNyq, unitsTarget, windowLength, overlap, astop0, apass0, astopLP and apassLP. criterion uses band or energyFraction with an explicit action; rule target is resolved only by match and is rejected here. Removal by mode name is rejected. includeResidue defaults to TRUE. Optional signal controls follow reconstructRecords.

Stochastic imf requires imf.seed, parallel FALSE and workers 1 explicitly. The process sorts the record keys and calls set.seed once for a nonempty batch of new records. The caller's resulting random state is retained. A modal JSON file beside the execution evidence records the seed, RNGkind, states before and after processing, executed and skipped records, effective controls and xemd version. Reproducibility is limited to the same environment, seed and executed order.

Append compares the recipe identity before processing records and skips complete records already committed under that identity. Writer locks and metadata transactions protect publication. An incomplete rollback retains its recovery transaction and record products.

Input errors, processing failures and partial publication raise conditions instead of returning 0. Failures retain their causes; partial publication identifies the products or transaction retained for recovery. The operation preserves the caller's working directory and session options.

Value

Invisible integer 0 after complete publication.

Examples

```
Root <- tempfile("processProject")
Raw <- file.path(Root, "database", "records", "A", "E1", "S1", "raw")
dir.create(Raw, recursive = TRUE)
Time <- seq(0, 10, by = 0.01)
Envelope <- sin(pi * Time / max(Time))^2
data.table::fwrite(data.table::data.table(
  HNE = 1200 * Envelope * sin(2 * pi * 1.5 * Time),
  HNN = 800 * Envelope * sin(2 * pi * 2.5 * Time + 0.4),
  HNZ = 400 * Envelope * sin(2 * pi * 4 * Time)), file.path(Raw, "AT.R1.csv"))
jsonlite::write_json(list(RecordID = "R1", OwnerID = "A", EventID = "E1",
  StationID = "S1", dt = 0.01), file.path(Raw, "AT.R1.json"), auto_unbox = TRUE)
data.table::fwrite(data.table::data.table(ProcessID = "raw", RecordID = "R1",
  OwnerID = "A", EventID = "E1", StationID = "S1"), file.path(Root, "selection.csv"))
# The gmsp stage spells its edge stop amplitude Astop0.
jsonlite::write_json(list(processID = "TEST", stages = "gmsp",
  parallel = FALSE, workers = 1,
  path = list(database = "database", selection = "selection.csv", stage = "stage"),
  gmsp = list(fmax = 25, kNyq = 4, flatZeros = FALSE, trimZeros = FALSE,
    Astop0 = 0.001, unitsSource = "mm", unitsTarget = "mm", windowLength = 128,
    overlap = 75, astopLP = 0.001, apass0 = 0.001, apassLP = 0.98)),
  file.path(Root, "process.json"), auto_unbox = TRUE)
processProject(file = "process.json", root = Root)
list.files(file.path(Root, "database", "process", "TEST"))
Intensity <- data.table::fread(file.path(Root, "database", "process", "TEST",
                                        "IntensityTable.csv"))
print(Intensity[, c("RecordID", "OCID", "PGA", "AI")])
unlink(Root, recursive = TRUE)
```

---

rankProject *Rank a processed candidate pool*

Description

Scores candidate spectra against a target band and writes a selection.

Usage

```
rankProject(file = file.path("gmsp", "match.json"), root = getwd())
```

Arguments

file	Path to the JSON project contract, relative to root or absolute.
root	Existing project directory used to resolve relative paths.

Details

stage0.N is an integer of at least two. The contract declares a processed base, target, xi, units.psa and candidate.ownerID or a candidate.pool with complete record keys. An explicit owner selector may be filtered by a pool. candidate.oneRecordPerEvent defaults to FALSE; when TRUE, the best-ranked record of each earthquake is kept, with the event identity of `selectProject()` read from the MasterIndex. Declared PSW is preferred; otherwise spectra are calculated from TSW. Checkpoint reuse requires identical resolved inputs and recipe, including oneRecordPerEvent, and an unchanged checkpoint file. Missing or mismatched identity causes a resumption error. parallel and workers are not read: ranking runs in the calling R process, and both keys enter the checkpoint identity with the rest of the contract. Invalid inputs and partial publication cause errors. The working directory is unchanged.

Value

Invisible integer zero after publication of stage0.psa.csv, stage0.identity.json, stage0.spectral.csv and the file at path.selection.

Target envelope

target.envelope is a required JSON object with lower and upper fields, for example `{"lower": "0.50", "upper": "0.95"}`. Each bound and target.p select an exact p label in the target table: `"mean"` or a decimal of the form `0.<digits>` strictly between 0 and 1, such as `"0.50"`. Percentages (`"50"`, `"50%"`) and p prefixes (`"p50"`) are rejected. Numeric scalars are converted to text and must satisfy the same decimal format and literal table match; `"0.5"` and `"0.50"` are different labels. The lower, central and upper labels must be distinct; duplicate object keys and target.envelope.ID are rejected. `"mean"` is valid for either bound or target.p when its rows exist and satisfy the band checks. lower, target.p and upper become Sa.low, Sa.mean and Sa.high respectively; Sa.mean denotes the selected central curve even when target.p is a quantile. The curves require identical Tn meshes containing zero (PGA) and positive periods, with finite ordinates satisfying `0 < Sa.low <= Sa.mean <= Sa.high` at every retained period. Tn is in seconds. Ordinates are read in target.units (default g) and converted to units.psa; conversions between g and mm/s2 (or mm/s^2) are supported, and identical units are retained.

Ranking with the band

At positive target periods, Sa.low and Sa.high define insideFraction and relative band violations. Candidates are ordered by bandRMSE, q90Violation and relaxedRMSE, with RecordID breaking ties. relaxedRMSE compares with Sa.mean and downweights values inside the band. stage0.spectral.csv records these scores before publication of the selection at path.selection.

Event magnitude

Before candidate.filter is applied, the EventMagnitude of the MasterIndex is transformed as in `selectProject()`: an Mw published by the index is kept, otherwise one published by the owner is taken; otherwise an Ms in any letter case becomes `0.67 Ms + 2.13`, the ISC-GEM catalogue relation of Storchak et al. (2013) as reproduced by Villalón-Semanat and Palau-Clares (2018, equation 10), published for `Ms <= 6.47` and extrapolated to every larger Ms. An mb in any letter case without a published Mw becomes `0.85 mb + 1.03` for `3.5 <= mb & mb <= 6.2`, the global empirical relation of Scordilis (2006, equation 22); outside that interval `Mw = mb`. An ML without a published Mw is

taken as Mw by identity. Magnitude types are compared in any letter case. Md and other or empty types keep their magnitude. A candidate.filter range on EventMagnitude and the Mw column of stage0.spectral.csv use the resulting magnitude. The MasterIndex file is unchanged and no column is added.

References

Villalón-Semanat, M. and Palau-Clares, R. (2018). Relaciones empíricas entre las magnitudes mb/Ms, Ms/Mw y mb/Mc para el área de Cuba, Jamaica y La Española. Minería y Geología, 34(2), 167-176.

Scordilis, E. M. (2006). Empirical Global Relations Converting MS and mb to Moment Magnitude. Journal of Seismology, 10(2), 225-236. doi:10.1007/s1095000690124.

See Also

selectProject(), matchProject()

Examples

```
Root <- tempfile("rankProject")
Process <- file.path(Root, "database", "process", "TEST")
dir.create(Process, recursive = TRUE)
dir.create(file.path(Root, "data"))
Keys <- data.table::data.table(ProcessID = "TEST", RecordID = c("R1", "R2", "R3"),
  OwnerID = c("A", "B", "A"), EventID = c("E1", "E2", "E3"), StationID = "S1")
data.table::fwrite(data.table::data.table(Keys, EventName = "First",
  EventMagnitude = 6, Repi = 10), file.path(Process, "MasterIndex.csv"))
data.table::fwrite(data.table::data.table(RecordID = Keys$RecordID, OCID = "H1",
  PGA = 100, AI = 10), file.path(Process, "IntensityTable.csv"))
data.table::fwrite(data.table::data.table(Keys[rep(seq_len(nrow(Keys)), each = 2)],
  Tn = rep(c(0.5, 2), nrow(Keys)),
  PSA.H1 = rep(c(0.84, 0.99, 0.40), each = 2) * 9806.65),
  file.path(Process, "PSW.csv"))
jsonlite::write_json(list(processID = "TEST", output = list(record = "PSW",
  table = c("MasterIndex", "IntensityTable", "PSW"))),
  file.path(Process, "process.json"), auto_unbox = TRUE)
Target <- data.table::CJ(ID = "MCE", siteID = "SITE", Vs30 = 800, Tn = c(0, 1),
                         p = c("0.50", "0.84", "0.95"))
Target$SaF <- as.numeric(Target$p)
saveRDS(Target, file.path(Root, "data", "MCETable.Rds"))
jsonlite::write_json(list(processID = "TEST", xi = 0.05, units = list(psa = "mm/s2"),
  stage0 = list(N = 2),
  path = list(database = "database", data = "data", selection = "rank/selection.csv"),
  candidate = list(ownerID = c("A", "B")),
  target = list(demand = "MCE", siteID = "SITE", Vs30 = 800, p = "0.84",
    value = "SaF", units = "g", envelope = list(lower = "0.50", upper = "0.95"))),
  file.path(Root, "match.json"), auto_unbox = TRUE)
rankProject(file = "match.json", root = Root)
Scores <- data.table::fread(file.path(Root, "rank", "stage0.spectral.csv"))
print(Scores[, c("RecordID", "rank.band", "insideFraction", "bandRMSE")])
data.table::fread(file.path(Root, "rank", "selection.csv"))
unlink(Root, recursive = TRUE)
```

`readProviderFile` *Read a provider signal file*

Description

Returns provider channels and native amplitudes without unit conversion. Time is in seconds; TWO_COL preserves its time column and the other formats construct time from zero using the header sampling interval.

Usage

```
readProviderFile(file, format)
```

Arguments

`file`	Path to one provider file.
`format`	One of AT2, TWO_COL, TRA, TRZ, TRB, TRC, ACA, ACB, LIS, V2, V2A or ISEE. V2 and ISEE variants are recognized within the format.

Details

Unknown formats, unreadable files and invalid required headers raise errors with file and format context. Files are read without writes.

Value

A data.table with columns `t`, `OCID`, `s`. An NPTS block with insufficient samples or NA within its declared samples returns NULL; multi-block readers omit such blocks. ISEE returns T, V and L velocities in mm/s.

Examples

```
Root <- tempfile("readProviderFile")
dir.create(Root)
File <- file.path(Root, "record.AT2")
writeLines(c("header", "Event, date, station, H1", "units",
             "NPTS= 4, DT= 0.01 SEC", "1.0 2.0 3.0 4.0"), File)
readProviderFile(File, format = "AT2")
unlink(Root, recursive = TRUE)
```

`readRecordTSL` *Read raw records as long time histories*

Description

Reads the requested raw CSV and JSON pairs and reconstructs time from zero using the sidecar sampling interval, without converting amplitudes.

Usage

```
readRecordTSL(records, path, ID)
```

Arguments

`records`	Table with RecordID, OwnerID, EventID and StationID keys.
`path`	Records root containing the per-station raw folders.
`ID`	Required quantity AT, VT or DT, selecting the file prefix.

Details

A missing requested pair, a nonpositive or nonfinite sampling interval, or a sidecar key contradicting the selection causes an error with the record and path. The supplied table is not modified.

Value

A new data.table with columns RecordID, OwnerID, EventID, StationID, OCID, ID, t and s, ordered by those keys and t. OCID retains the provider channel names. Time t is in seconds.

Examples

```
Root <- tempfile("readRecordTSL")
Raw <- file.path(Root, "AAA", "E1", "S1", "raw")
dir.create(Raw, recursive = TRUE)
data.table::fwrite(data.table::data.table(H1 = c(1, 2), H2 = c(0, 1)),
                   file.path(Raw, "AT.R1.csv"))
jsonlite::write_json(list(dt = 0.01), file.path(Raw, "AT.R1.json"), auto_unbox = TRUE)
Keys <- data.table::data.table(RecordID = "R1", OwnerID = "AAA",
                               EventID = "E1", StationID = "S1")
readRecordTSL(Keys, path = Root, ID = "AT")
unlink(Root, recursive = TRUE)
```

`reconstructRecords` *Reconstruct acceleration records from selected modes*

Description

Measures and selects modes through xemd, reconstructs acceleration, and reintegrates it with buildTSL using isRaw = FALSE.

Usage

```
reconstructRecords(
  decomposition,
  criterion,
  by,
  signalControl,
  includeResidue = TRUE
)
```

Arguments

`decomposition`	Native list returned by decomposeRecords, containing signal and modes tables with ID AT, OCID, t in seconds, s and metadata. Samples use the length basis declared by signalControl$unitsTarget. An optional residue or fitResidual supplies the remainder; both together are an error.
`criterion`	Named list with exactly rule, action and the rule fields. Rule "band" requires finite fMin and fMax in Hz with 0 <= fMin <= fMax; "energyFraction" requires finite minimum >= 0. Action is "keep" or "remove"; selection by mode index is not admitted.
`by`	Explicit character vector of metadata columns identifying a record; character() declares one record without grouping metadata. Metadata must be constant within each record/component/quantity series. Only the metadata selected by by is retained in the returned TSL.
`signalControl`	Named list requiring unitsTarget, fmax, windowLength, overlap, astop0, apass0, astopLP and apassLP, with the units and conditions of buildTSL. Optional kNyq, flatZeros, trimZeros, removeMean, forceRegularization and includeWarnings retain their buildTSL defaults. Duplicate or other fields are errors; Units metadata, when present, must agree with unitsTarget.
`includeResidue`	Logical scalar, TRUE by default, including residue or fitResidual when present; an absent remainder contributes nothing.

Details

Selection applies separately by OCID and retains native mode identities; the closed band compares frequency without bandwidth. A zero-energy mode has no frequency and is outside a frequency

band. Reconstruction sums retained modes and the requested remainder before reintegration; meanNoise is not added again. Zero reconstructed acceleration energy is an error before buildTSL. Reintegration can taper, resample or remove means according to the supplied signal controls, so its histories need not equal the modal sum.

Invalid data or controls cause errors; dependency conditions retain their original classes, cause and group when supplied. Inputs are preserved; the operation reads or writes no files and uses no random numbers.

Value

A list with TSL, the reintegrated AT/VT/DT table; metrics, the freshly measured native mode metrics; and selection, the native decision table for that decomposition.

See Also

`decomposeRecords()`, `buildTSL()`, `xemd::selectModesTSL()`, `xemd::reconstructTSL()`

Examples

```
Time <- seq(0, 9, length.out = 1000)
TSL <- data.table::data.table(RecordID = "R1", OCID = "H1", ID = "AT", t = Time,
  s = sin(pi * Time) + sin(2 * pi * Time) + sin(6 * pi * Time) + 0.5 * Time,
  Units = "mm")
Decomposition <- decomposeRecords(TSL, method = "vmd",
  control = list(k = 3, alpha = 2000, tau = 0.5, DC = FALSE, init = 0,
                 tolerance = 1e-6, maxIterations = 98),
  by = "RecordID")
Control <- list(unitsTarget = "mm", fmax = 25, kNyq = 4, windowLength = 128,
                overlap = 75, astop0 = 1e-3, apass0 = 1e-3, astopLP = 1e-3,
                apassLP = 0.98, includeWarnings = FALSE)
Reconstruction <- reconstructRecords(Decomposition,
  criterion = list(rule = "band", fMin = 2, fMax = 15, action = "keep"),
  by = "RecordID", signalControl = Control)
print(Reconstruction$selection)
head(Reconstruction$TSL)
unique(Reconstruction$TSL$ID)
```

---

selectProject *Select records from a project contract*

Description

Filters index metadata and writes the selected records and candidate pool.

Usage

```
selectProject(file = file.path("gmsp", "select.json"), root = getwd())
```

Arguments

`file`	Path to the JSON project contract, relative to root or absolute.
`root`	Existing project directory used to resolve relative paths.

Details

owners, path.selection and index paths are required. processID defaults to raw, requireIMF to FALSE and oneRecordPerEvent to TRUE. PGA criteria require units.PGA. A target PGA can be derived from path.matchConfig and explicit positive record-factor bounds. A selection directory containing foreign entries is rejected before replacement. Missing contracts, invalid criteria and partial publication cause errors. The working directory is unchanged.

Value

Invisible integer zero after publication of selection.csv, candidates.csv, selection.json and summary.json under path.selection.

One record per event

Records belong to the same earthquake when their EventID.USGS is equal; legacy processed indexes name that column usgsEventId. A record without it is identified by its EventID. With oneRecordPerEvent TRUE, selection.csv keeps one record per earthquake, the one with the largest PGA; ties go to the larger StationID and then to the smaller RecordID. candidates.csv keeps every record that meets a criterion. The events count in summary.json uses the same identity.

Event magnitude

Before the criteria are applied, EventMagnitude takes a moment magnitude where one is published or an Ms or mb is converted. When EventMagnitudeType is mw, mwc, mww, mwr or mwb in any letter case, the magnitude and its type are kept. Otherwise, when EventMagnitudeType.owner is one of those types, EventMagnitude.owner is taken and the type becomes Mw. Otherwise an Ms in any letter case becomes `0.67 Ms + 2.13` and its type becomes Mw. This is the ISC-GEM catalogue relation of Storchak et al. (2013) as reproduced by Villalón-Semanat and Palau-Clares (2018, equation 10); it is published for `Ms <= 6.47`, and the same line is extrapolated to every larger Ms. An mb in any letter case without a published Mw becomes `0.85 mb + 1.03` for `3.5 <= mb & mb <= 6.2`, the global empirical relation of Scordilis (2006, equation 22). Outside that interval its value is kept, so that `Mw = mb`; the type becomes Mw in both cases. An ML without a published Mw keeps its value and takes the type Mw, an identity meant for the moderate range. Md and other or empty types keep their magnitude and type. The EventMagnitude and EventMagnitudeType criteria and the published tables use the resulting values. The source values remain in the .owner and .USGS columns of the index (and .ISC where present).

References

Villalón-Semanat, M. and Palau-Clares, R. (2018). Relaciones empíricas entre las magnitudes mb/Ms, Ms/Mw y mb/Mc para el área de Cuba, Jamaica y La Española. Minería y Geología, 34(2), 167-176.

Scordilis, E. M. (2006). Empirical Global Relations Converting MS and mb to Moment Magnitude. Journal of Seismology, 10(2), 225-236. doi:10.1007/s1095000690124.

See Also

rankProject(), matchProject()

Examples

```
Root <- tempfile("selectProject")
dir.create(file.path(Root, "database", "index"), recursive = TRUE)
dir.create(file.path(Root, "database", "process", "TEST"), recursive = TRUE)
Index <- data.table::data.table(ProcessID = "TEST", RecordID = c("001", "002", "003"),
  OwnerID = "A", EventID = c("2020A", "2020A", "2021B"),
  StationID = c("01", "02", "03"), DIR = "H1", OCID = "H1",
  PGA = c(4903.325, 4903.325, 7354.9875))
data.table::fwrite(Index, file.path(Root, "database", "process", "TEST",
                                    "MasterIndex.csv"))
jsonlite::write_json(list(processID = "TEST", owners = "A", oneRecordPerEvent = TRUE,
  path = list(index = "database/index", selection = "selection")),
  file.path(Root, "select.json"), auto_unbox = TRUE)
selectProject(file = "select.json", root = Root)
list.files(file.path(Root, "selection"))
data.table::fread(file.path(Root, "selection", "selection.csv"),
                  colClasses = "character")
unlink(Root, recursive = TRUE)
```

---

selectRecords *Select record identities*

Description

Intersects identity filters and returns unique record keys without modifying the supplied table.

Usage

```
selectRecords(
  data,
  recordID = NULL,
  eventID = NULL,
  stationID = NULL,
  ownerID = NULL
)
```

Arguments

data	Table containing RecordID, OwnerID, EventID and StationID.
recordID, eventID, stationID, ownerID	Identity vectors for the corresponding columns. NULL leaves a dimension unrestricted; an empty vector selects zero rows.

Details

Duplicate key rows are collapsed. With every filter NULL, the result contains the unique keys of the supplied table. Missing required columns are input errors. The operation performs no file access.

Value

A data.table with RecordID, OwnerID, EventID and StationID, in that column order, sorted by OwnerID, EventID and StationID.

See Also

`readRecordTSL()`, `writeSelection()`

Examples

```
Records <- data.table::data.table(RecordID = c("R1", "R2", "R3"), OwnerID = "AAA",
  EventID = c("E1", "E1", "E2"), StationID = c("S1", "S2", "S1"))
selectRecords(Records, eventID = "E1")
selectRecords(Records, stationID = "S1", ownerID = "AAA")
```

`widenIML` *Widen long intensity measures*

Description

Projects intensity measures to one row per record and component, with one column for each measure. Values and units are preserved without conversion.

Usage

```
widenIML(IML, by)
```

Arguments

`IML`	A table with columns `OCID`, `IM`, `value`, optional `ID` and `units`, and the metadata selected by `by`.
`by`	Character vector of record metadata columns; `character()` retains OCID as the row key. Data columns cannot be metadata keys.

Details

Keys `by`, `OCID`, `IM` must be unique; duplicate or contradictory measure rows cause an error without aggregation. Absent ID or units fields are omitted from the attribute. Metadata excluded by by are dropped. CSV serialization contains the value columns; the measureMetadata attribute remains an in-memory representation of domains and units.

The input is copied. The function reads or writes no files, uses no random numbers and changes no session options.

Value

A new data.table with columns by,OCID,<IM>, ordered by metadata and component. Measure columns follow their first occurrence in IML; missing combinations and supplied NA values remain missing. The tabular attribute measureMetadata retains by,OCID,IM and the available ID,units fields. A correctly shaped empty input returns the row-key columns with zero rows and no measure columns.

See Also

measureIntensitiesTSL()

Examples

```
IML <- data.table::data.table(
  RecordID = "R1", OCID = "H1", ID = "AT", IM = c("PGA", "CAV"),
  value = c(500, 120), units = c("mm /s2", "mm /s"))
IMW <- widenIML(IML, by = "RecordID")
attr(IMW, "measureMetadata")
```

---

widenPSL *Widen canonical response spectra*

Description

Projects spectra to columns named <ID>.<OCID> without changing values or units. The input table is preserved.

Usage

```
widenPSL(PSL, by)
```

Arguments

PSL	Long table with OCID,Tn,ID,S and selected metadata.
by	Character vector of metadata columns forming row keys, or character() for one record without metadata. A damping column xi is retained when included in by.

Details

Duplicate by,Tn,ID,OCID keys are errors; values are not aggregated. Missing combinations become NA. A correctly shaped empty input returns its key columns with no spectra columns. The projection uses no random numbers, files or session-option changes.

Value

A new data.table ordered by by,Tn. These key columns precede the spectra columns, ordered by component with H1, H2, UP, D50 and D100 first, and by ID with PSA, PSV and SD first within each component. Other components and IDs remain after the preferred values.

See Also

lengthenPSW(), computeSpectraTSL()

Examples

```
PSL <- data.table::data.table(RecordID = "R1", OCID = "H1",
                              Tn = 0.1, ID = "PSA", S = 1)
widenPSL(PSL, by = "RecordID")
```

---

widenTSL *Widen a canonical long time-series table*

Description

Projects samples to columns named <ID>.<OCID>. It performs no signal processing or unit conversion and does not modify its input.

Usage

```
widenTSL(TSL, by, ids = c("AT", "VT", "DT"))
```

Arguments

TSL	A table with OCID, ID, t and s, and the metadata columns selected by by. Time is in seconds.
by	Character vector of metadata columns identifying a record. Use character() for one record without metadata. Names must be unique, present, and distinct from the data columns.
ids	Preferred order of quantity identifiers. Other identifiers remain after these; this argument never filters quantities.

Details

Duplicate by,t,ID,OCID keys are errors. A dense input with nonempty identifiers and no dots in ID supports a lossless value round trip. The first dot separates ID from OCID; further dots belong to OCID. Sparse input acquires explicit NA rows on conversion back.

Value

A new data.table ordered by by, t, with those columns first and one column per quantity/component. Missing combinations become NA. Values and unit-bearing metadata selected in by are preserved.

See Also

lengthenTSW()

Examples

```
TSL <- data.table::data.table(
  RecordID = "R1", OCID = "H1", ID = rep(c("AT", "VT"), each = 2),
  t = rep(c(0, 0.01), 2), s = c(1, 2, 0.1, 0.2))
widenTSL(TSL, by = "RecordID")
```

---

writeSelection *Write a station-folder selection*

Description

Writes unique station-folder keys to CSV and a JSON summary.

Usage

```
writeSelection(data, name, path)
```

Arguments

data	Table containing OwnerID, EventID and StationID.
name	Nonempty selection name used as the output filename stem.
path	Nonempty destination directory, created recursively if needed.

Details

The files are named from name with .csv and .json suffixes. Existing files at these paths are overwritten. The JSON fields are name, dateUTC, nHits and hitsPerOwner; dateUTC uses UTC to the second. An empty selection has nHits zero. RecordID is not part of these keys. A JSON failure after CSV publication raises modulePartialPublication with the original condition, partial result and affected paths; the CSV remains. Completion signals a moduleNote condition. Missing required columns and invalid name/path arguments are input errors.

Value

Invisibly, a data.table with OwnerID, EventID and StationID, sorted by these keys and without duplicate rows.

See Also

selectRecords()

Examples

```
Root <- tempfile("writeSelection")
Records <- data.table::data.table(OwnerID = "AAA", EventID = c("E1", "E1", "E2"),
                                  StationID = c("S1", "S2", "S1"))
Selection <- writeSelection(Records, name = "demo", path = Root)
print(Selection)
list.files(Root)
str(jsonlite::read_json(file.path(Root, "demo.json")))
unlink(Root, recursive = TRUE)
```

Index

accumulateIntensityTSL, 3
accumulateIntensityTSL(), *31*
alignComponents, 4
archiveRawOwner, 5
assessCompatibility, 5
assessCompatibility(), *23, 24*, *29*
assessDistances, 7
assessDistances(), *9*
assessParsers, 8
assessSite, 9
assessSite(), *7*

buildRawFileTable, 10
buildRawIntensityTable, 11
buildRawRecordTable, 12
buildTSL, 13
buildTSL(), *23*, *41*

computeSpectraTSL, 16
computeSpectraTSL(), *25*, *46*

decomposeRecords, 17
decomposeRecords(), *41*
describeModule, 18

exportProject, 19
extractRecord, 20

fitModalFactor, 21
fitModalFactor(), *24*, *29*
fitRecordFactor, 23
fitRecordFactor(), *6*, *23*, *29*

lengthenPSW, 25
lengthenPSW(), *17*, *46*
lengthenTSW, 26
lengthenTSW(), *15*, *46*

mapComponents, 27
matchProject, 28
matchProject(), *37*, *43*
measureIntensitiesTSL, 30
measureIntensitiesTSL(), *4*, *45*

normalizeTSL, 32
normalizeTSL(), *15*

parseRecord, 33
parseRecord(), *8*
processProject, 34

rankProject, 35
rankProject(), *43*
readProviderFile, 38
readRecordTSL, 39
readRecordTSL(), *44*
reconstructRecords, 40
reconstructRecords(), *18*

selectProject, 41
selectProject(), *36, 37*
selectRecords, 43
selectRecords(), *47*

widenIML, 44
widenIML(), *31*
widenPSL, 45
widenPSL(), *17*, *25*
widenTSL, 46
widenTSL(), *15*, *26*
writeSelection, 47
writeSelection(), *44*

xemd::decomposeTSL(), *18*
xemd::reconstructTSL(), *41*
xemd::selectModesTSL(), *23*, *41*