

# Package: movementsync (via r-universe)

September 5, 2024

**Type** Package

**Title** Analysis and Visualisation of Musical Audio and Video Movement Synchrony Data

**Version** 0.1.4

**Description** Analysis and visualisation of synchrony, interaction, and joint movements from audio and video movement data of a group of music performers. The demo is data described in Clayton, Leante, and Tarsitani (2021) <[doi:10.17605/OSF.IO/KS325](https://doi.org/10.17605/OSF.IO/KS325)>, while example analyses can be found in Clayton, Jakubowski, and Eerola (2019) <[doi:10.1177/1029864919844809](https://doi.org/10.1177/1029864919844809)>. Additionally, wavelet analysis techniques have been applied to examine movement-related musical interactions, as shown in Eerola et al. (2018) <[doi:10.1098/rsos.171520](https://doi.org/10.1098/rsos.171520)>.

**License** MIT + file LICENSE

**Depends** R (>= 2.10)

**Imports** circular, dplyr, ggplot2, gridExtra, hms, igraph, lmtest, methods, osfr, rlang, scales, signal, tidyr, WaveletComp, zoo

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**URL** <https://tuomaseerola.github.io/movementsync/>

**BugReports** <https://github.com/tuomaseerola/movementsync/issues>

**Encoding** UTF-8

**LazyData** false

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Repository** <https://tuomaseerola.r-universe.dev>

**RemoteUrl** <https://github.com/tuomaseerola/movementsync>

**RemoteRef** HEAD

**RemoteSha** 8c71e20abf7cc64d83184e9eaf6fca022aec0f1

## Contents

analyze_coherency . . . . .	4
analyze_wavelet . . . . .	5
apply_column_spliceview . . . . .	7
apply_filter . . . . .	8
apply_filter_sgolay . . . . .	9
apply_segment_spliceview . . . . .	10
autolayer . . . . .	11
autoplot . . . . .	13
autoplot.GrangerTime . . . . .	14
autoplot.SpectralDensityView . . . . .	15
ave_cross_power_over_splices . . . . .	16
ave_cross_power_spliceview . . . . .	17
ave_power_over_splices . . . . .	18
ave_power_spliceview . . . . .	20
calculate_ave_cross_power1 . . . . .	21
calculate_ave_power1 . . . . .	22
clip_splice . . . . .	23
compare_ave_cross_power1 . . . . .	24
compare_ave_power1 . . . . .	25
compare_avg_cross_power2 . . . . .	26
compare_avg_power2 . . . . .	28
difference_onsets . . . . .	29
distribution_dp . . . . .	30
get_data_points . . . . .	31
get_duration_annotation_data . . . . .	31
get_feature_data . . . . .	32
get_filtered_views . . . . .	33
get_granger_interactions . . . . .	34
get_joined_view . . . . .	35
get_local_max_average_power . . . . .	36
get_metre_data . . . . .	37
get_onsets_selected_data . . . . .	38
get_osf_recordings . . . . .	39
get_processed_view . . . . .	39
get_processed_views . . . . .	40
get_raw_optflow_view . . . . .	41
get_raw_view . . . . .	42
get_raw_views . . . . .	43
get_recording . . . . .	44
get_sample_recording . . . . .	45
get_spliced_view . . . . .	45
granger_test . . . . .	46
is_splice_overlapping . . . . .	47
list_osf_recordings . . . . .	48
map_to_granger_test . . . . .	48
merge_splice . . . . .	49

motion_gram . . . . .	50
ms_condgrangertest . . . . .	51
ms_grangertest1 . . . . .	52
ms_grangertest2 . . . . .	53
NIR_ABh_Puriya_Annotation . . . . .	54
NIR_ABh_Puriya_Annotation_Influence . . . . .	54
NIR_ABh_Puriya_Central_Feature_Sitar . . . . .	55
NIR_ABh_Puriya_Central_Pose_Sitar . . . . .	56
NIR_ABh_Puriya_Central_Pose_Tabla . . . . .	57
NIR_ABh_Puriya_Metre_DrutTeental . . . . .	58
NIR_ABh_Puriya_Metre_VilambitTeental . . . . .	59
NIR_ABh_Puriya_Onsets_Selected_DrutTeental . . . . .	59
NIR_ABh_Puriya_Onsets_Selected_VilambitTeental . . . . .	60
NIR_ABh_Puriya_OptFlow_Central_Sitar . . . . .	61
open_movementsync_data . . . . .	62
plot.Duration . . . . .	62
plot.GrangerInteraction . . . . .	63
plot.Metre . . . . .	64
plot.OnsetsSelected . . . . .	65
plot.View . . . . .	65
plot_average_coherency . . . . .	66
plot_average_power . . . . .	67
plot_cross_spectrum . . . . .	68
plot_cwt_energy . . . . .	69
plot_history_xy . . . . .	69
plot_influence_diagram . . . . .	70
plot_phase_difference . . . . .	71
plot_power_spectrum . . . . .	72
plot_roll_resultant_length . . . . .	73
plot_sel_phases . . . . .	74
plot_wt_energy . . . . .	75
pull_segment_spliceview . . . . .	76
sample_gap_splice . . . . .	77
sample_offset_splice . . . . .	78
sample_time_spliced_views . . . . .	79
specgram_plot . . . . .	80
spectral_density . . . . .	80
splice_time . . . . .	81
splice_time.Duration . . . . .	82
splice_time.list . . . . .	83
splice_time.Metre . . . . .	83
splice_time.OnsetsDifference . . . . .	85
splice_time.View . . . . .	86
split.SplicedView . . . . .	87
subset.View . . . . .	88
summary.analyze.wavelet . . . . .	88
summary.Duration . . . . .	89
summary.Metre . . . . .	90

summary.OnsetsSelected . . . . .	90
summary.Recording . . . . .	91
summary.sel.phases . . . . .	92
summary.View . . . . .	92
summary_onsets . . . . .	93
velocity_dp . . . . .	94
visualise_sample_splices . . . . .	95
xlim_duration . . . . .	96

<b>Index</b>	<b>97</b>
--------------	-----------

---

analyze_coherency	<i>Analyze Coherency from View object</i>
-------------------	---

---

## Description

Analyze Coherency from View object

## Usage

```
analyze_coherency(
  obj,
  columns,
  loess.span = 0,
  dj = 1/50,
  lowerPeriod = 2/obj$recording$fps,
  upperPeriod = 5,
  window.type.t = 1,
  window.type.s = 1,
  window.size.t = 5,
  window.size.s = 1/4,
  make.pval = TRUE,
  method = "white.noise",
  params = NULL,
  n.sim = 1,
  date.format = NULL,
  date.tz = NULL,
  verbose = FALSE
)
```

## Arguments

obj	View object.
columns	Two column names.
loess.span	parameter alpha in loess controlling the degree of time series smoothing, if the time series is to be detrended; no detrending if loess.span = 0. Default: 0.
dj	frequency resolution. Default 1/20.

lowerPeriod	in seconds
upperPeriod	in seconds
window.type.t	see <a href="#">WaveletComp::analyze.coherency()</a> .
window.type.s	see <a href="#">WaveletComp::analyze.coherency()</a> .
window.size.t	see <a href="#">WaveletComp::analyze.coherency()</a> .
window.size.s	see <a href="#">WaveletComp::analyze.coherency()</a> .
make.pval	see <a href="#">WaveletComp::analyze.coherency()</a> .
method	see <a href="#">WaveletComp::analyze.coherency()</a> .
params	see <a href="#">WaveletComp::analyze.coherency()</a> .
n.sim	number of simulations (default 1).
date.format	see <a href="#">WaveletComp::analyze.coherency()</a> .
date.tz	see <a href="#">WaveletComp::analyze.coherency()</a> .
verbose	see <a href="#">WaveletComp::analyze.coherency()</a> .

**Value**

an analyze\_coherency object.

**See Also**

Other wavelet functions: [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
co <- analyze_coherency(pv, c("Nose_x", "Nose_y"))
```

---

analyze\_wavelet

*Analyze Wavelet from View object*


---

**Description**

Analyze Wavelet from View object

**Usage**

```
analyze_wavelet(
  obj,
  column,
  loess.span = 0,
  dj = 1/20,
  lowerPeriod = 2/obj$recording$fps,
  upperPeriod = 5,
  make.pval = TRUE,
  method = "white.noise",
  params = NULL,
  n.sim = 1,
  date.format = NULL,
  date.tz = NULL,
  verbose = TRUE
)
```

**Arguments**

obj	View object.
column	Column in view to analyse.
loess.span	parameter alpha in loess controlling the degree of time series smoothing, if the time series is to be detrended; no detrending if loess.span = 0. Default: 0.
dj	frequency resolution. Default 1/20.
lowerPeriod	lower Fourier period in seconds. Defaults to 2/fps.
upperPeriod	upper Fourier period in seconds. Defaults to 5s.
make.pval	see <a href="#">WaveletComp::analyze.wavelet()</a> .
method	see <a href="#">WaveletComp::analyze.wavelet()</a> .
params	see <a href="#">WaveletComp::analyze.wavelet()</a> .
n.sim	number of simulations (default 1).
date.format	see <a href="#">WaveletComp::analyze.wavelet()</a> .
date.tz	see <a href="#">WaveletComp::analyze.wavelet()</a> .
verbose	see <a href="#">WaveletComp::analyze.wavelet()</a> .

**Value**

an analyze.wavelet object.

**See Also**

Other wavelet functions: [analyze\\_coherency\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

## Examples

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
w <- analyze_wavelet(pv, "Nose_y")
```

---

apply\_column\_spliceview

*Apply summary function to the columns in each segment of a Splice-View object*

---

## Description

Apply summary function to each data point column in a SplicedView and return list of output data.

## Usage

```
apply_column_spliceview(sv, FUN, simplify = FALSE, USE.NAMES = FALSE, ...)
```

```
sapply_column_spliceview(sv, FUN, simplify = TRUE, USE.NAMES = TRUE, ...)
```

## Arguments

sv	SplicedView object.
FUN	function to apply.
simplify	see <a href="#">sapply()</a> .
USE.NAMES	see <a href="#">sapply()</a> .
...	passed to FUN.

## Value

see [sapply\(\)](#).

## See Also

Other statistical and analysis functions: [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```

r <- get_sample_recording()
d1 <- get_duration_annotation_data(r)
# only one relevant section for sample data
splicing_smile_df <- splice_time(d1, tier = 'INTERACTION',
  comments = 'Mutual look and smile')

fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)
sv_duration_smile <- get_spliced_view(jv, splicing_df = splicing_smile_df)
mean_mat <- apply_column_spliceview(sv_duration_smile, mean, na.rm=TRUE)

```

---

`apply_filter`*Apply a filter to a View*

---

**Description**

Apply a filter to a View

**Usage**

```

apply_filter(
  view,
  data_points,
  sig_filter,
  param_str = "",
  folder_out = "Filtered",
  save_output = FALSE
)

```

**Arguments**

<code>view</code>	ProcessedView object.
<code>data_points</code>	body parts e.g. 'Nose'.
<code>sig_filter</code>	S3 filter object from signals package.
<code>param_str</code>	string of parameter values to add to output file if desired.
<code>folder_out</code>	output folder relative to recording home (default is 'Filtered').
<code>save_output</code>	save the output?

**Value**

a filtered object.



---

apply\_filter\_sgolay    *Apply a Savitzky-Golay filter to a view*

---

### Description

Apply a Savitzky-Golay filter to a view

### Usage

```
apply_filter_sgolay(  
  view,  
  data_points,  
  n,  
  p,  
  folder_out = "Filtered",  
  save_output = FALSE  
)
```

### Arguments

view	View object.
data_points	body parts e.g. 'Nose'.
n	window size.
p	poly order.
folder_out	output folder relative to recording home (default is 'Filtered').
save_output	save the output?

### Value

a FilteredView object.

### See Also

Other data functions: [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

### Examples

```
r <- get_sample_recording()  
rv <- get_raw_view(r, "Central", "", "Sitar")  
pv <- get_processed_view(rv)  
  
set.seed(1)  
fv1 <- apply_filter_sgolay(pv, c("Nose", "RWrist", "LWrist"), n = 19, p = 4)  
fv2 <- apply_filter_sgolay(pv, c("Nose", "RWrist", "LWrist"), n = 41, p = 3)
```

```
set.seed(1) # to reproduce with S3 filter object
fv3 <- apply_filter(pv, c("Nose", "RWrist", "LWrist"), signal::sgolay(4, 19))
```

---

```
apply_segment_spliceview
```

*Apply complex function to each segment in a SpliceView object*

---

## Description

Apply complex function to each segment in a SpliceView object

## Usage

```
apply_segment_spliceview(sv, FUN, ...)
```

## Arguments

sv	SplicedView object.
FUN	function to apply.
...	passed to FUN.

## Value

list of two elements: 'output' containing results of apply FUN to 'input'

## See Also

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

## Examples

```
r <- get_sample_recording()
d1 <- get_duration_annotation_data(r)
# only one relevant section for sample data
splicing_smile_df <- splice_time(d1, tier = 'INTERACTION',
  comments = 'Mutual look and smile')

fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)
sv_duration_smile <- get_spliced_view(jv, splicing_df = splicing_smile_df)
wavelet_smile_list <- apply_segment_spliceview(sv_duration_smile, analyze_wavelet,
  column = "Nose_x_Central_Sitar")
names(wavelet_smile_list)
```

---

`autolayer`*Autolayer methods*

---

**Description**

Layers of annotation data to add to ggplots in 'movementsync'.

**Usage**

```
## S3 method for class 'OnsetsSelected'
autolayer(
  object,
  time_limits = c(-Inf, Inf),
  colour = "Inst.Name",
  fill = "Metre",
  alpha = 0.4,
  instrument_cols = NULL,
  ...
)

## S3 method for class 'Metre'
autolayer(
  object,
  time_limits = c(-Inf, Inf),
  colour = "hotpink",
  alpha = 0.5,
  tempo = FALSE,
  view = NULL,
  columns = NULL,
  ...
)

## S3 method for class 'Duration'
autolayer(
  object,
  time_limits = c(-Inf, Inf),
  expr = .data$Tier == "FORM",
  fill_column = "Comments",
  geom = "rect",
  vline_column = "In",
  ...
)

## S3 method for class 'Splice'
autolayer(object, geom = "rect", vline_column = "Start", ...)
```

**Arguments**

object	S3 object
time_limits	tuple of time limits.
colour	name of column for colouring.
fill	name of column for filling.
alpha	aesthetic
instrument_cols	instrument column names.
...	passed to geom.
tempo	do we plot tempo with a Metre layer? (Default is FALSE).
view	view object for a tempo Metre layer (Default is NULL).
columns	columns for view for a tempo Metre layer (Default is NULL).
expr	unquoted R expression for filtering data (default is Tier == 'FORM').
fill_column	data column used for fill.
geom	'rect' or 'vline'.
vline_column	column name for position of vertical lines.

**Value**

ggplot geom object

**Examples**

```

r<-get_recording("NIR_ABh_Puriya", fps=25)
o <- get_onsets_selected_data(r)
v <- get_raw_view(r, "Central", "", "Sitar")
autoplot(v, columns = c("LEar_x", "LEar_y"), maxpts=5000) + autolayer(o)

m <- get_metre_data(r)
autoplot(v, columns = c("LEar_x", "LEar_y"), time_limits = c(1000, 2000)) +
  autolayer(m, time_limits = c(1000, 2000))
autoplot(v, columns = c("LEar_x", "LEar_y"), time_limits = c(1000, 2000)) +
  autolayer(m, tempo = TRUE, time_limits = c(1000, 2000), view = v,
            columns = c("LEar_x", "LEar_y"))

d <- get_duration_annotation_data(r)
autoplot(m)
autoplot(m) + autolayer(d)
autoplot(m) + autolayer(d, fill_col = "Tier")

v <- get_raw_view(r, "Central", "", "Sitar")
autoplot(v, columns = c("LEar_x", "LEar_y")) +
  autolayer(d)
autoplot(v, columns = c("LEar_x", "LEar_y")) +
  autolayer(d, expr = Tier == "FORM" & substr(Comments, 1, 1) == "J")
autoplot(v, columns = c("LEar_x", "LEar_y")) +
  autolayer(d, geom = "vline", nudge_x = -60, size = 3, colour = "blue")

```

---

autoplot	<i>Diagnostic plots</i>
----------	-------------------------

---

## Description

Autoplot methods for S3 objects in the movementsync package.

## Usage

```
## S3 method for class 'Duration'
autoplot(object, horizontal = FALSE, ...)

## S3 method for class 'OnsetsSelected'
autoplot(object, instrument = "Inst", tactus = "Matra", ...)

## S3 method for class 'Metre'
autoplot(object, ...)

## S3 method for class 'View'
autoplot(
  object,
  columns = NULL,
  maxpts = 1000,
  time_limits = c(-Inf, Inf),
  time_breaks = NULL,
  expr = NULL,
  ...
)

## S3 method for class 'SplicedView'
autoplot(
  object,
  columns = NULL,
  segments = NULL,
  time_breaks = NULL,
  time_limits = c(-Inf, Inf),
  maxpts = 1000,
  ...
)
```

## Arguments

object	S3 object
horizontal	make the barchart horizontal? (Default is FALSE).
...	passed to <code>zoo::plot.zoo()</code> .
instrument	instrument column name.

tactus	beat column name.
columns	names of columns in input data.
maxpts	maximum number of points to plot
time_limits	tuple to restrict the timeline or a duration object.
time_breaks	suggests the number of major time tick marks (Default is NULL).
expr	an R expression that sets the time scale using a duration object (Default is NULL).
segments	only include these segments in a SplicedView plot.

**Value**

a ggplot object.

**Examples**

```
r <- get_sample_recording()
d <- get_duration_annotation_data(r)
autoplot(d)
o <- get_onsets_selected_data(r)
autoplot(o)
m <- get_metre_data(r)
autoplot(m)
v <- get_raw_view(r, "Central", "", "Sitar")
autoplot(v, columns = c("LEar_x", "LEar_y"), time_limits = c(20, 40))
l <- list(a = c(0, 10), b = c(20, 30), c = c(30, 60))
splicing_df <- splice_time(l)
sv <- get_spliced_view(v, splicing_df)
autoplot(sv, columns = c("LEar_x", "LEar_y", "Nose_x", "Nose_y"), time_breaks = 4, maxpts = 1000)
```

---

autoplot.GrangerTime *Plot a Granger S3 object*

---

**Description**

Plot a Granger S3 object

**Usage**

```
## S3 method for class 'GrangerTime'
autoplot(object, splicing_df, lev_sig = 0.05, ...)
```

**Arguments**

object	S3 object.
splicing_df	Splicing data.frame object.
lev_sig	significance level.
...	ignored.

**Value**

a ggplot object.

**See Also**

Other Granger Causality: [get\\_granger\\_interactions\(\)](#), [granger\\_test\(\)](#), [map\\_to\\_granger\\_test\(\)](#), [ms\\_condgrangertest\(\)](#), [ms\\_grangertest1\(\)](#), [ms\\_grangertest2\(\)](#), [plot.GrangerInteraction\(\)](#), [plot\\_influence\\_diagram\(\)](#)

**Examples**

```
r1 <- get_sample_recording()
fv_list <- get_filtered_views(r1, data_points = "Nose", n = 41, p = 3)
jv_sub <- get_joined_view(fv_list)
splicing_df <- splice_time(jv_sub, win_size = 3, step_size = 0.5)
sv <- get_spliced_view(jv_sub, splicing_df)
g <- granger_test(sv, "Nose_x_Central_Sitar", "Nose_x_Central_Tabla", lag = 3/25)
autoplot(g, splicing_df)
```

---

autoplot.SpectralDensityView

*Autoplot a SpectralDensityView S3 object*

---

**Description**

Autoplot a SpectralDensityView S3 object

**Usage**

```
## S3 method for class 'SpectralDensityView'
autoplot(object, period_range = c(0, 10), colour = "blue", ...)
```

**Arguments**

object	SpectralDensityView object.
period_range	tuple for limiting range of periods.
colour	name of line colour.
...	ignored.

**Value**

a ggplot object.

**Examples**

```

r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
sd1 <- spectral_density(pv, columns = c("LElbow_x", "LEye_x"), spans = 5)
autoplot(sd1)

fv <- apply_filter_sgolay(pv, data_points = c("LElbow", "LEye"), n = 19, p = 4)
sd2 <- spectral_density(fv, data_points = c("LElbow", "LEye"), spans = c(3, 3))
autoplot(sd2)

```

---

```
ave_cross_power_over_splices
```

*Calculate mean average cross power over splices using a splicing table*

---

**Description**

Randomly generates splices from a splicing table and calculates average cross power for each segment and splice. Calculates the mean average cross power over the random splices for each segment and period. Compares with the average cross power for the original splice.

**Usage**

```

ave_cross_power_over_splices(
  jv,
  splicing_df,
  num_splices,
  columns,
  sampling_type = "offset",
  rejection_list = list(),
  include_original = TRUE,
  show_plot = TRUE
)

```

**Arguments**

<code>jv</code>	JoinedView object.
<code>splicing_df</code>	Splice object.
<code>num_splices</code>	number of randomly chosen splices.
<code>columns</code>	name of data columns on which to calculate average cross power.
<code>sampling_type</code>	either 'offset' or 'gap'.
<code>rejection_list</code>	list of splice objects that random splices must not overlap.
<code>include_original</code>	include the original splice in output? (Default is TRUE).
<code>show_plot</code>	show a plot? (Default is TRUE).



**Value**

data.frame of splice segments and their average cross power.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)

d <- get_duration_annotation_data(r)
splicing_tabla_solo_df <- splice_time(d,
  expr = "Tier == 'INTERACTION' & Comments == 'Mutual look and smile'")

# Only do the first splice for sample data
mean_ave_cross_power_df <- ave_cross_power_over_splices(jv,
  splicing_tabla_solo_df[1,], num_splices = 10,
  columns = c('Nose_x_Central_Sitar', 'Nose_y_Central_Sitar'), show_plot = TRUE)
```

---

ave\_cross\_power\_spliceview

*Get the average cross power on each segment in a SplicedView*

---

**Description**

Get the average cross power on each segment in a SplicedView

**Usage**

```
ave_cross_power_spliceview(
  sv,
  columns,
  colour = "blue",
  segments = NULL,
  show_plot = FALSE,
  ...
)
```

**Arguments**

sv	SplicedView object
columns	column names in the data of each SplicedView object.
colour	name of colour on plots (default is 'blue').
segments	indices of segments to plot e.g. 1:10 (default plots up to first 10).
show_plot	show a plot (default is FALSE).
...	passed to <a href="#">analyze_coherency()</a> .

**Value**

data.frame with columns containing Average Cross Power for each segment.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
d1 <- get_duration_annotation_data(r)
# only one relevant section for sample data
splicing_smile_df <- splice_time(d1, tier = 'INTERACTION',
  comments = 'Mutual look and smile')

fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)
sv_duration_smile <- get_spliced_view(jv, splicing_df = splicing_smile_df)
ave_cross_power_smile <- ave_cross_power_spliceview(
  sv_duration_smile, columns = c("Nose_x_Central_Sitar", "Nose_y_Central_Sitar"), show_plot = TRUE)
head(ave_cross_power_smile)
```

---

ave\_power\_over\_splices

*Calculate mean average power over splices using a splicing table*

---

**Description**

Randomly generates splices from a splicing table and calculates average power for each segment and splice. Calculates the mean average power over the random splices for each segment and period. Compares with the average power for the original splice.

**Usage**

```
ave_power_over_splices(
  jv,
  splicing_df,
  num_splices,
  column,
  sampling_type = "offset",
  rejection_list = list(),
  include_original = TRUE,
  show_plot = TRUE
)
```

**Arguments**

jv	JoinedView object.
splicing_df	Splice object.
num_splices	number of randomly chosen splices.
column	name of data column on which to calculate average power.
sampling_type	either 'offset' or 'gap'.
rejection_list	list of splice objects that random splices must not overlap.
include_original	include the original splice in output? (Default is TRUE).
show_plot	show a plot? (Default is TRUE).

**Value**

data.frame of splice segments and their average power.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)

d <- get_duration_annotation_data(r)
splicing_tabla_solo_df <- splice_time(d,
  expr = "Tier == 'INTERACTION' & Comments == 'Mutual look and smile'")
```

```
# Only do the first splice for sample data
mean_ave_power_df <- ave_power_over_splices(jv, splicing_tabla_solo_df[1,], num_splices = 10,
column = 'Nose_x_Central_Sitar', show_plot = TRUE)
```

---

ave\_power\_spliceview *Get the average power on each segment in a SplicedView*

---

### Description

Get the average power on each segment in a SplicedView

### Usage

```
ave_power_spliceview(
  sv,
  column,
  colour = "blue",
  segments = NULL,
  show_plot = FALSE,
  ...
)
```

### Arguments

sv	SplicedView object
column	name of data column on which to calculate average power.
colour	name of colour on plots (default is 'blue').
segments	indices of segments to plot e.g. 1:10 (default plots up to first 10).
show_plot	show a plot? (Default is FALSE).
...	passed to <a href="#">analyze_wavelet()</a> .

### Value

data.frame with columns containing Average Power for each segment.

### See Also

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```

r <- get_sample_recording()
d1 <- get_duration_annotation_data(r)
# only one relevant section for sample data
splicing_smile_df <- splice_time(d1, tier = 'INTERACTION',
  comments = 'Mutual look and smile')

fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)
sv_duration_smile <- get_spliced_view(jv, splicing_df = splicing_smile_df)
ave_power_smile <- ave_power_spliceview(sv_duration_smile,
  column = "Nose_x_Central_Sitar", show_plot=TRUE)
head(ave_power_smile)

```

---

```
calculate_ave_cross_power1
```

*Calculate average cross power distribution using a splicing table*

---

**Description**

Calculate average cross power distribution using a splicing table

**Usage**

```

calculate_ave_cross_power1(
  jv,
  splicing_df,
  splice_name,
  num_segment_samples,
  columns,
  show_plot = TRUE
)

```

**Arguments**

jv	JoinedView object.
splicing_df	Splice object.
splice_name	Name to give randomly spliced segments.
num_segment_samples	number of segments to randomly sample.
columns	name of data columns on which to calculate average cross power.
show_plot	show the plot? (Default is TRUE).

**Value**

a data frame: containing average cross power on the spliced JoinedView.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = 'Nose', n = 41, p = 3)
jv <- get_joined_view(fv_list)
splicing_df <- splice_time(list(a = c(0, 5), b = c(10, 15)))
output_dfr <- calculate_ave_cross_power1(jv, splicing_df, 'Splice', 10,
  c('Nose_x_Central_Tabla', 'Nose_y_Central_Tabla'))
```

---

calculate\_ave\_power1    *Calculate average power distribution using a splicing table*

---

**Description**

Calculate average power distribution using a splicing table

**Usage**

```
calculate_ave_power1(
  jv,
  splicing_df,
  splice_name,
  num_segment_samples,
  column,
  show_plot = TRUE
)
```

**Arguments**

jv	JoinedView object.
splicing_df	Splice object.
splice_name	Name to give randomly spliced segments.
num_segment_samples	number of segments to randomly sample.
column	name of data column on which to calculate average power.
show_plot	show the plot? (Default is TRUE).

**Value**

a data frame: containing average power on the spliced JoinedView.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = 'Nose', n = 41, p = 3)
jv <- get_joined_view(fv_list)
splicing_df <- splice_time(list(a = c(0, 5), b = c(10, 15)))
output_dfr <- calculate_ave_power1(jv, splicing_df, 'Splice', 10, 'Nose_x_Central_Tabla')
```

---

clip\_splice

*Clip a splice so segments are of fixed duration*


---

**Description**

Clip a splice so segments are of fixed duration

**Usage**

```
clip_splice(splice_dfr, duration, location = "middle")
```

**Arguments**

splice_dfr	Splice object.
duration	window duration in seconds.
location	'beginning', 'middle' or 'end'.

**Value**

a Splice object.

**See Also**

Other splicing functions: [get\\_spliced\\_view\(\)](#), [is\\_splice\\_overlapping\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

**Examples**

```

l <- list(a = c(10, 20), b = c(30, 40), c = c(50, 55))
splice_dfr <- splice_time(1)
clip_splice(splice_dfr, duration = 1)
clip_splice(splice_dfr, duration = 6)
clip_splice(splice_dfr, duration = 1, location = 'beginning')
clip_splice(splice_dfr, duration = 10, location = 'beginning')
clip_splice(splice_dfr, duration = 1, location = 'end')
clip_splice(splice_dfr, duration = 10, location = 'end')

```

---

compare\_ave\_cross\_power1

*Compare average cross power distribution using a splicing table*

---

**Description**

Compare average cross power distribution using a splicing table

**Usage**

```

compare_ave_cross_power1(
  jv,
  splicing_df,
  splice_name,
  num_segment_samples,
  num_splice_samples,
  columns,
  sampling_type = "offset",
  rejection_list = list(),
  show_plot = TRUE
)

```

**Arguments**

jv	JoinedView object.
splicing_df	Splice object.
splice_name	Name to give randomly spliced segments.
num_segment_samples	number of segments to randomly sample.
num_splice_samples	number of randomly chosen splices.
columns	name of data columns on which to calculate cross average power.
sampling_type	either 'offset' or 'gap'.
rejection_list	list of splice objects that random splices must not overlap.
show_plot	show the plot? (Default is TRUE).



**Value**

list of two data frames: one containing average cross power on the first splice and the other containing the average cross power on randomly generated splices.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = 'Nose', n = 41, p = 3)
jv <- get_joined_view(fv_list)
splicing_df <- splice_time(list(a = c(0, 5), b = c(10, 15)))
output_list <- compare_ave_cross_power1(jv, splicing_df, 'Random Splices', 5, 5,
c('Nose_x_Central_Tabla', 'Nose_y_Central_Tabla'))
```

---

compare\_ave\_power1      *Compare average power distribution using a splicing table*

---

**Description**

Compare average power distribution using a splicing table

**Usage**

```
compare_ave_power1(
  jv,
  splicing_df,
  splice_name,
  num_segment_samples,
  num_splice_samples,
  column,
  sampling_type = "offset",
  rejection_list = list(),
  show_plot = TRUE
)
```

**Arguments**

jv	JoinedView object.
splicing_df	Splice object.
splice_name	Name to give randomly spliced segments.
num_segment_samples	number of segments to randomly sample.
num_splice_samples	number of randomly chosen splices.
column	name of data column on which to calculate average power.
sampling_type	either 'offset' or 'gap'.
rejection_list	list of splice objects that random splices must not overlap.
show_plot	show the plot? (Default is TRUE).

**Value**

list of two data frames: one containing average power on the first splice and the other containing the average power on randomly generated splices.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = 'Nose', n = 41, p = 3)
jv <- get_joined_view(fv_list)
splicing_df <- splice_time(list(a = c(0, 5), b = c(10, 15)))
output_list <- compare_ave_power1(jv, splicing_df, 'Random Splices', 5, 5, 'Nose_x_Central_Table')
```

---

compare\_avg\_cross\_power2

*Compare the average cross power distribution of two SplicedViews using sampling on each segment*

---

**Description**

Compare the average cross power distribution of two SplicedViews using sampling on each segment

**Usage**

```
compare_avg_cross_power2(
  sv1,
  sv2,
  name1,
  name2,
  num_samples,
  columns,
  show_plot = TRUE
)
```

**Arguments**

sv1	SplicedView object.
sv2	SplicedView object.
name1	name for first object.
name2	name for second object.
num_samples	number of samples to draw from segments.
columns	column names in the data e.g. c('Nose_x', 'Nose_y').
show_plot	show the plot?

**Value**

list of two data.frames containing the sampled data.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r <- get_sample_recording()
d1 <- get_duration_annotation_data(r)
fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)

# only one relevant section for sample data
splicing_smile_df <- splice_time(d1, tier = 'INTERACTION',
  comments = 'Mutual look and smile')
sv_duration_smile <- get_spliced_view(jv, splicing_df = splicing_smile_df)

splicing_alap_df <- splice_time(
  d1, tier = 'FORM', comments = 'Alap'
)
```

```
sv_duration_alap <- get_spliced_view(jv, splicing_df = splicing_alap_df)

sample_list <- compare_avg_cross_power2(
  sv_duration_smile, sv_duration_alap, 'Smile', 'Alap', num_samples = 100,
  columns = c("Nose_x_Central_Sitar", "Nose_y_Central_Sitar"))
```

---

compare_avg_power2	<i>Compare the average power distribution of two SplicedViews using sampling on each segment</i>
--------------------	--

---

### Description

Compare the average power distribution of two SplicedViews using sampling on each segment

### Usage

```
compare_avg_power2(
  sv1,
  sv2,
  name1,
  name2,
  num_samples,
  column,
  show_plot = TRUE
)
```

### Arguments

sv1	SplicedView object.
sv2	SplicedView object.
name1	name for first object.
name2	name for second object.
num_samples	number of samples to draw from segments.
column	column name in the data e.g. 'Nose_x_Central_Sitar'.
show_plot	show the plot?

### Value

list of two data.frames containing the sampled data.

### See Also

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```

r <- get_sample_recording()
d1 <- get_duration_annotation_data(r)
fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)

# only one relevant section for sample data
splicing_smile_df <- splice_time(d1, tier = 'INTERACTION',
  comments = 'Mutual look and smile')
sv_duration_smile <- get_spliced_view(jv, splicing_df = splicing_smile_df)

splicing_alap_df <- splice_time(
  d1, tier = 'FORM', comments = 'Alap'
)
sv_duration_alap <- get_spliced_view(jv, splicing_df = splicing_alap_df)

sample_list <- compare_avg_power2(
  sv_duration_smile, sv_duration_alap, 'Smile', 'Alap', num_samples = 100,
  column = "Nose_x_Central_Sitar")

```

---

difference\_onsets      *Get onset differences*

---

**Description**

Calculates the difference in onset times for each instrument pair in milli-seconds.

**Usage**

```
difference_onsets(onset_obj, instruments, expr = NULL, splicing_dfr = NULL)
```

**Arguments**

onset_obj	OnsetsSelected object.
instruments	character vector of instrument names.
expr	R expression to subset onsets (not required).
splicing_dfr	Splice object (not required).

**Value**

OnsetsDifference object.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

## Examples

```
r1 <- get_sample_recording()
o1 <- get_onsets_selected_data(r1)
head(difference_onsets(o1, instruments = c('Inst', 'Tabla')))
head(difference_onsets(o1, instruments = c('Inst', 'Tabla'), expr = 'Matra == 3'))
```

---

distribution_dp	<i>Distribution plot of a view object</i>
-----------------	---

---

## Description

Distribution plot of a view object

## Usage

```
distribution_dp(obj, maxpts = 50000, alpha = 0.1, ...)
```

## Arguments

obj	View object.
maxpts	maximum number of points to plot.
alpha	ggplot aesthetic value.
...	passed to <code>ggplot2::geom_point()</code> ,

## Value

a ggplot object.

## Examples

```
r1 <- get_sample_recording()
rv1 <- get_raw_view(r1, "Central", "", "Sitar")
pv1 <- get_processed_view(rv1)
dp <- c("LWrist", "RWrist", "LElbow", "RElbow", "LEye", "REye", "Neck", "MidHip")
fv1 <- apply_filter_sgolay(pv1, data_point = dp, n = 41, p = 4)
distribution_dp(fv1)
```

---

get\_data\_points      *Get the data points held in a view*

---

**Description**

Get the data points held in a view

**Usage**

```
get_data_points(obj)
```

**Arguments**

obj                  View object.

**Value**

character vector of body parts.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
get_data_points(rv)
```

---

get\_duration\_annotation\_data  
*Get duration annotation data*

---

**Description**

Get duration annotation data

**Usage**

```
get_duration_annotation_data(recording, filetype = "rda", verbose = FALSE)
```

**Arguments**

recording	Recording object.
filetype	type of file ('rda' as default), can be 'csv'.
verbose	messages the specific data loaded (default is 'FALSE').

**Value**

list of data.frames.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
df <- get_duration_annotation_data(r)
```

---

get_feature_data	<i>Get Feature Data</i>
------------------	-------------------------

---

**Description**

Output from new analysis process that generates data at the same sample rate as the video data. The user is responsible for ensuring that this data is continuous before using this function.

**Usage**

```
get_feature_data(
  recording,
  vid,
  direct,
  inst,
  interpolate_data = FALSE,
  folder_out = tempdir(),
  save_output = FALSE,
  filetype = "rda",
  verbose = FALSE
)
```



**Arguments**

recording	Recording object.
vid	camera.
direct	direction.
inst	instrument.
interpolate_data	should the data be interpolated? (default is FALSE).
folder_out	output folder relative to recording home (default is 'tempdir()').
save_output	save the output?
filetype	type of file ('rda' as default), can be 'csv'.
verbose	messages the specific data loaded (default is 'FALSE').

**Value**

a FilteredView object.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
fd <- get_feature_data(r, "Central", "", "Sitar")
fv_list <- get_filtered_views(r, 'LEar', n = 41, p = 3)
fv_list$Feature <- fd
jv <- get_joined_view(fv_list)
get_data_points(jv)
autoplot(jv)
```

---

get\_filtered\_views      *Get filtered views*

---

**Description**

Get filtered views

**Usage**

```
get_filtered_views(r, data_points, n, p, filetype = "rda")
```

**Arguments**

r	Recording object.
data_points	vector of body parts e.g. 'Nose'.
n	window size.
p	poly order.
filetype	type of file ('rda' as default), can be 'csv'.

**Value**

list of FilteredView objects.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, "Nose", n = 41, p = 3)
plot(fv_list$Central_Tabla)
```

---

get\_granger\_interactions

*Get Granger Causality interactions*

---

**Description**

Get Granger Causality interactions

**Usage**

```
get_granger_interactions(
  sv,
  columns,
  cond_column = "",
  sig_level = 0.05,
  lag = 1,
  granger_fn = ms_grangertest2
)
```

**Arguments**

sv	SplicedView object
columns	vector of column names
cond_column	name of conditioning column
sig_level	significance level
lag	in seconds (rounded to nearest frame)
granger_fn	function to perform Granger test (defaults to ms_grangertest2)

**Value**

GrangerInteraction object

**See Also**

Other Granger Causality: [autoplot.GrangerTime\(\)](#), [granger\\_test\(\)](#), [map\\_to\\_granger\\_test\(\)](#), [ms\\_condgrangertest\(\)](#), [ms\\_grangertest1\(\)](#), [ms\\_grangertest2\(\)](#), [plot.GrangerInteraction\(\)](#), [plot\\_influence\\_diagram\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, "Nose", n = 41, p = 3)
jv_sub <- get_joined_view(fv_list)
l <- list(a = c(0, 300), b = c(300, 600), c = c(600, 900))
splicing_df <- splice_time(l)
sv <- get_spliced_view(jv_sub, splicing_df)
g <- get_granger_interactions(sv, c("Nose_x_Central_Sitar", "Nose_x_Central_Tabla"), lag = 1/25)
print(g)
```

---

get_joined_view	<i>Get joined view from multiple views from the same recording</i>
-----------------	--

---

**Description**

Get joined view from multiple views from the same recording

**Usage**

```
get_joined_view(l, folder_out = "Joined", save_output = FALSE)
```

**Arguments**

l	named list of View objects.
folder_out	output folder relative to recording home (default is 'Joined').
save_output	save the output?

**Value**

JoinedView object

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv_list <- get_raw_views(r)
jv <- get_joined_view(rv_list)
plot(jv, columns = c("LEar_x_Central_Sitar", "LEar_x_Central_Tabla"), yax.flip=TRUE)
```

---

`get_local_max_average_power`

*Get periods locally maximal average power*

---

**Description**

Get periods locally maximal average power

**Usage**

```
get_local_max_average_power(obj, v)
```

**Arguments**

`obj` analyze.wavelet object.  
`v` View object.

**Value**

data.frame of Period and Local Maxima.

**See Also**

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
pv1 <- subset(pv, Time >= 10)
w <- analyze_wavelet(pv1, "Nose_x")
plot_average_power(w, pv1)
get_local_max_average_power(w, pv1)
```

---

get_metre_data	<i>Get metre files</i>
----------------	------------------------

---

**Description**

Get metre files

**Usage**

```
get_metre_data(recording, filetype = "rda", verbose = FALSE)
```

**Arguments**

recording	Recording object.
filetype	type of file ('rda' as default), can be 'csv'.
verbose	messages the specific data loaded (default is 'FALSE').

**Value**

list of data.frames.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
m <- get_metre_data(r)
```

---

`get_onsets_selected_data`*Get onsets selected files*

---

## Description

Get onsets selected files

## Usage

```
get_onsets_selected_data(  
  recording,  
  tactus = "Matra",  
  filetype = "rda",  
  verbose = FALSE  
)
```

## Arguments

<code>recording</code>	Recording object.
<code>tactus</code>	optional name of the beat column to ensure it is turned into integer.
<code>filetype</code>	type of file ('rda' as default), can be 'csv'.
<code>verbose</code>	messages the specific data loaded (default is 'FALSE').

## Value

list of data.frames

## See Also

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

## Examples

```
r <- get_sample_recording()  
o <- get_onsets_selected_data(r)
```

---

get\_osf\_recordings      *Get movementsync recording from OSF*

---

**Description**

Get movementsync recording from OSF

**Usage**

```
get_osf_recordings(  
  stems = c("NIR_ABh_Puriya", "NIRP1_VS_Hams", "NIRP1_MAK_Jaun", "Gagaku_5_Juha",  
            "NIR_DBh_Malhar"),  
  to_dir = tempdir(),  
  overwrite = FALSE  
)
```

**Arguments**

stems                  zip file stem(s).  
to\_dir                 directory to copy to (default is "tempdir").  
overwrite              overwriting existing dataset files?

**Value**

invisible vector of downloaded CSV file names.

**Examples**

```
get_osf_recordings()
```

---

get\_processed\_view      *Get processed view from Pose video data*

---

**Description**

Normalises and interpolates missing data in the view.

**Usage**

```
get_processed_view(  
  rv,  
  folder_out = tempdir(),  
  save_output = FALSE,  
  verbose = FALSE  
)
```

**Arguments**

rv	RawView object.
folder_out	output folder relative to recording home (default is 'Normalized').
save_output	save the output?
verbose	messages the specific data loaded (default is 'FALSE').

**Value**

a ProcessedView object.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
```

---

get\_processed\_views    *Get processed views*

---

**Description**

Get processed views

**Usage**

```
get_processed_views(r, data_points, filetype = "rda")
```

**Arguments**

r	Recording object.
data_points	vector of body parts e.g. 'Nose'.
filetype	type of file ('rda' as default), can be 'csv'.

**Value**

list of ProcessedView objects.



**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
pv_list <- get_processed_views(r)
plot(pv_list$Central_Tabla)
```

---

`get_raw_optflow_view` *Creates time reference and displacement from raw csv optflow data*

---

**Description**

Used to load OptFlow data.

**Usage**

```
get_raw_optflow_view(
  recording,
  vid,
  direct,
  inst,
  folder_out = tempdir(),
  save_output = FALSE,
  filetype = "rda",
  verbose = FALSE
)
```

**Arguments**

<code>recording</code>	Recording object.
<code>vid</code>	camera.
<code>direct</code>	direction.
<code>inst</code>	instrument.
<code>folder_out</code>	output folder relative to recording home (default is 'tempdir()').
<code>save_output</code>	save the output?
<code>filetype</code>	type of file ('rda' as default), can be 'csv'.
<code>verbose</code>	messages the specific data loaded (default is 'FALSE').

**Value**

an OptFlowView object.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_recording("NIR_ABh_Puriya", fps = 25)
rov <- get_raw_optflow_view(r, "Central" , "", "Sitar")
pov <- get_processed_view(rov)
fv1 <- apply_filter_sgolay(pov, c("Head"), n=19, p=4)
autoplot(fv1)
```

---

get\_raw\_view

*Get view from Pose video data*

---

**Description**

Creates time reference and displacement from raw csv data for the view.

**Usage**

```
get_raw_view(
  recording,
  vid,
  direct,
  inst,
  out_folder = tempdir(),
  save_output = FALSE,
  filetype = "rda",
  verbose = FALSE
)
```

**Arguments**

recording	Recording object.
vid	video camera.
direct	direction.
inst	instrument.
out_folder	output folder (tempdir if nothing is given).
save_output	save the output?
filetype	type of file ('rda' as default), can be 'csv'.
verbose	messages the specific data loaded (default is 'FALSE').

**Value**

a RawView object.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
v <- get_raw_view(r, "Central", "", "Sitar")
```

---

get_raw_views	<i>Get Pose views from a recording</i>
---------------	--

---

**Description**

Get Pose views from a recording

**Usage**

```
get_raw_views(recording, filetype = "rda")
```

**Arguments**

recording	Recording object.
filetype	type of file ('rda' as default), can be 'csv'.

**Value**

named list of views

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#), [get\\_sample\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
v_list <- get_raw_views(r)
```

---

get_recording	<i>Get a meta-data recording object</i>
---------------	---

---

### Description

Get a meta-data recording object

### Usage

```
get_recording(  
  stem,  
  fps,  
  folder_in = "data",  
  path = system.file(package = "movementsync"),  
  filetype = "csv",  
  verbose = FALSE  
)
```

### Arguments

stem	recording identifier.
fps	frames per second.
folder_in	input folder relative to recording home (default is 'Original').
path	recording home folder.
filetype	type of file ('rda' as default), can be 'csv'.
verbose	messages the specific data loaded (default is 'FALSE').

### Value

a Recording object.

### See Also

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_sample\\_recording\(\)](#)

### Examples

```
# Get the details of one recording  
r <- get_recording("NIR_ABh_Puriya", fps=25)
```

---

get\_sample\_recording *Get sample meta-data recording object*

---

**Description**

Get sample meta-data recording object

**Usage**

```
get_sample_recording(stem = "NIR_ABh_Puriya")
```

**Arguments**

stem                    recording identifier.

**Value**

a Recording object.

**See Also**

Other data functions: [apply\\_filter\\_sgolay\(\)](#), [get\\_data\\_points\(\)](#), [get\\_duration\\_annotation\\_data\(\)](#), [get\\_feature\\_data\(\)](#), [get\\_filtered\\_views\(\)](#), [get\\_joined\\_view\(\)](#), [get\\_metre\\_data\(\)](#), [get\\_onsets\\_selected\\_data\(\)](#), [get\\_processed\\_views\(\)](#), [get\\_processed\\_view\(\)](#), [get\\_raw\\_optflow\\_view\(\)](#), [get\\_raw\\_views\(\)](#), [get\\_raw\\_view\(\)](#), [get\\_recording\(\)](#)

**Examples**

```
r <- get_sample_recording()
```

---

get\_spliced\_view            *Get spliced view from view object*

---

**Description**

Get spliced view from view object

**Usage**

```
get_spliced_view(v, splicing_df)
```

**Arguments**

v                        View object  
splicing\_df            Splice object.

**Value**

a SplicedView object.

**See Also**

Other splicing functions: [clip\\_splice\(\)](#), [is\\_splice\\_overlapping\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
l <- list(a = c(0, 10), b = c(10, 20), c = c(20, 30))
splicing_df <- splice_time(l)
sv <- get_spliced_view(rv, splicing_df)
```

---

granger\_test

*Granger causality tests applied to a SplicedView*


---

**Description**

Granger causality tests applied to a SplicedView

**Usage**

```
granger_test(
  obj,
  var1,
  var2,
  var3 = "",
  lag = 1,
  granger_fn = ms_grangertest2,
  cond_granger_fn = ms_condgrangertest
)
```

**Arguments**

obj	SplicedView object
var1	column name of response
var2	column name of predictor
var3	column name of conditioning
lag	in seconds (rounded to nearest frame)
granger_fn	function to perform Granger test (defaults to ms_grangertest2)
cond_granger_fn	function to perform conditional Granger test (defaults to ms_condgrangertest)

**Value**

GrangerTime object

**See Also**

Other Granger Causality: [autoplot.GrangerTime\(\)](#), [get\\_granger\\_interactions\(\)](#), [map\\_to\\_granger\\_test\(\)](#), [ms\\_condgrangertest\(\)](#), [ms\\_grangertest1\(\)](#), [ms\\_grangertest2\(\)](#), [plot.GrangerInteraction\(\)](#), [plot\\_influence\\_diagram\(\)](#)

**Examples**

```
r1 <- get_sample_recording()
rv_list <- get_raw_views(r1)
pv_list <- lapply(rv_list, get_processed_view)
get_data_points(pv_list$Central_Sitar)
fv_list <- lapply(pv_list, apply_filter_sgolay, data_points = "Nose", n = 41, p = 3)
jv_sub <- get_joined_view(fv_list)
splicing_df <- splice_time(jv_sub, win_size = 5, step_size = 0.5)
sv <- get_spliced_view(jv_sub, splicing_df)
granger_test(sv, "Nose_x_Central_Sitar", "Nose_x_Central_Tabla", lag = 1/25)
granger_test(sv, "Nose_x_Central_Sitar", "Nose_x_Central_Tabla", "Nose_y_Central_Tabla", lag = 1/25)
```

---

`is_splice_overlapping` *Checks if splicing data.frames overlap*

---

**Description**

Checks if splicing data.frames overlap

**Usage**

```
is_splice_overlapping(...)
```

**Arguments**

... Each argument can be a data frame or a list of data frames

**Value**

logical

**See Also**

Other splicing functions: [clip\\_splice\(\)](#), [get\\_spliced\\_view\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

**Examples**

```
l1 <- list(a=c(1, 10), a = c(20, 30), b = c(30, 40))
dfr1 <- splice_time(l1)
l2 <- list(a=c(10, 15), b = c(15, 25))
dfr2 <- splice_time(l2)
is_splice_overlapping(dfr1, dfr2)
```

---

list\_osf\_recordings    *List available recordings for movementsync from OSF*

---

**Description**

List available recordings for movementsync from OSF

**Usage**

```
list_osf_recordings()
```

**Value**

character vector of stem names

**Examples**

```
list_osf_recordings()
```

---

map\_to\_granger\_test    *Map duration object comments to a Granger Test object*

---

**Description**

Map duration object comments to a Granger Test object

**Usage**

```
map_to_granger_test(d, g, influence1, influence2)
```

**Arguments**

d	DurationObject
g	GrangerTest object
influence1	Comment X>Y string in the Granger Test of Y~X i.e. X causes Y
influence2	Comment X>Y string in the Granger Test of Y~X i.e. X causes Y



**Value**

modified Duration object

**See Also**

Other Granger Causality: [autoplot.GrangerTime\(\)](#), [get\\_granger\\_interactions\(\)](#), [granger\\_test\(\)](#), [ms\\_condgrangertest\(\)](#), [ms\\_grangertest1\(\)](#), [ms\\_grangertest2\(\)](#), [plot.GrangerInteraction\(\)](#), [plot\\_influence\\_diagram\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv_sub <- get_joined_view(fv_list)
splicing_df <- splice_time(jv_sub, win_size = 5, step_size = 0.5)
sv <- get_spliced_view(jv_sub, splicing_df)
g <- granger_test(sv, "Nose_x_Central_Sitar", "Nose_x_Central_Tabla", lag = 1/25)
d <- get_duration_annotation_data(r)
map_to_granger_test(d, g, "Influence T>S", "Influence S>T")
```

---

merge\_splice

---

*Merge splices together using set operations*


---

**Description**

Merge splices together using set operations

**Usage**

```
merge_splice(..., operation)
```

**Arguments**

... a collection of named Splice objects.  
operation either 'union' or 'intersection'.

**Value**

a Splice object.

**See Also**

Other splicing functions: [clip\\_splice\(\)](#), [get\\_spliced\\_view\(\)](#), [is\\_splice\\_overlapping\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

**Examples**

```

l1 <- list(a1 = c(100, 200), a2 = c(250, 300), a3 = c(400, 550), a4 = c(600, 650))
split1_dfr <- splice_time(l1)
split1_dfr

l2 <- list(b1 = c(150, 275), b2 = c(610, 640))
split2_dfr <- splice_time(l2)
split2_dfr

l3 <- list(c1 = c(275, 325), c2 = c(600, 675), c3 = c(700, 725))
split3_dfr <- splice_time(l3)
split3_dfr

merge_splice(x = split1_dfr, y = split2_dfr, z = split3_dfr, operation = 'union')
merge_splice(x = split1_dfr, y = split2_dfr, z = split3_dfr, operation = 'intersection')

```

---

motion\_gram

*Motion gram of a view object*

---

**Description**

Motion gram of a view object

**Usage**

```
motion_gram(obj, maxpts = 10000, alpha = 0.5, ...)
```

**Arguments**

obj	view object
maxpts	maximum number of points to plot.
alpha	ggplot aesthetic value.
...	passed to <code>ggplot2::geom_point()</code> ,

**Value**

a gtable object.

**Examples**

```

r1 <- get_sample_recording()
rv1 <- get_raw_view(r1, "Central", "", "Sitar")
pv1 <- get_processed_view(rv1)
dp <- c("LWrist", "RWrist", "LElbow", "RElbow", "LEye", "REye", "MidHip")
fv1 <- apply_filter_sgolay(pv1, data_point = dp, n = 41, p = 4)
sub_fv1 <- subset(fv1, Time >= 0 & Time <= 20, dp, by = 2)
motion_gram(sub_fv1)

```

---

ms_condgrangertest	<i>Test for Conditional Granger Causality</i>
--------------------	---

---

### Description

Faster implementation of the vector version of `lmtest::grangertest()` with conditioning on the *history* of a third variable. The function assumes time series always have the same start date and periodicity, which is true for the data in this package.

### Usage

```
ms_condgrangertest(x, y, z, order = 1, na.action = stats::na.omit, ...)
```

### Arguments

x	response vector of observations.
y	explanatory vector of observations.
z	conditioning vector of observations
order	number of lags (in frames).
na.action	a function for eliminating NAs after aligning the series x and y.
...	passed to <code>lmtest::waldtest()</code> .

### Value

Anova object

### See Also

Other Granger Causality: [autoplot.GrangerTime\(\)](#), [get\\_granger\\_interactions\(\)](#), [granger\\_test\(\)](#), [map\\_to\\_granger\\_test\(\)](#), [ms\\_grangertest1\(\)](#), [ms\\_grangertest2\(\)](#), [plot.GrangerInteraction\(\)](#), [plot\\_influence\\_diagram\(\)](#)

### Examples

```
data(wages, package = "lmtest")
diff_wages <- diff(wages)

# Granger tests
lmtest::grangertest(diff_wages[, 'w'], diff_wages[, 'CPI'], order = 3)
ms_grangertest1(diff_wages[, 'w'], diff_wages[, 'CPI'], order = 3)
ms_grangertest2(diff_wages[, 'w'], diff_wages[, 'CPI'], order = 3)

ms_condgrangertest(diff_wages[, 'w'], diff_wages[, 'CPI'], diff_wages[, 'u'], order = 3)
```

---

ms_grangertest1	<i>Test for Granger Causality</i>
-----------------	-----------------------------------

---

### Description

Faster implementation of the vector version of `lmtest::grangertest()` which uses a vectorised lag operation.

### Usage

```
ms_grangertest1(x, y, order = 1, na.action = stats::na.omit, ...)
```

### Arguments

x	either a bivariate series (in which case y has to be missing) or a univariate series of observations.
y	a univariate series of observations (if x is univariate, too).
order	number of lags (in frames).
na.action	a function for eliminating NAs after aligning the series x and y.
...	passed to <code>lmtest::waldtest()</code> .

### Value

Anova object

### See Also

Other Granger Causality: `autoplot.GrangerTime()`, `get_granger_interactions()`, `granger_test()`, `map_to_granger_test()`, `ms_condgrangertest()`, `ms_grangertest2()`, `plot.GrangerInteraction()`, `plot_influence_diagram()`

### Examples

```
data(ChickEgg, package = "lmtest")
ms_grangertest1(ChickEgg, order = 3)
```

---

ms_grangertest2	<i>Test for Granger Causality</i>
-----------------	-----------------------------------

---

## Description

Faster implementation of the vector version of `lmtest::grangertest()`. The function assumes time series always have the same start date and periodicity, which is true for the data in this package.

## Usage

```
ms_grangertest2(x, y, order = 1, na.action = stats::na.omit, ...)
```

## Arguments

x	either a bivariate series (in which case y has to be missing) or a univariate series of observations
y	a univariate series of observations (if x is univariate, too).
order	number of lags (in frames).
na.action	a function for eliminating NAs after aligning the series x and y.
...	passed to <code>lmtest::waldtest()</code> .

## Value

Anova object

## See Also

Other Granger Causality: `autoplot.GrangerTime()`, `get_granger_interactions()`, `granger_test()`, `map_to_granger_test()`, `ms_condgrangertest()`, `ms_grangertest1()`, `plot.GrangerInteraction()`, `plot_influence_diagram()`

## Examples

```
data(ChickEgg, package = "lmtest")
ms_grangertest2(ChickEgg, order = 3)
```

NIR\_ABh\_Puriya\_Annotation

*NIR\_ABh\_Puriya\_Annotation*

---

**Description**

A subset of data from NIR\_ABh\_Puriya annotation. The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

**Usage**

```
data(NIR_ABh_Puriya_Annotation)
```

**Format**

rda:

A data frame with 161 rows and 5 columns:

**START-END** Type of annotation

**2nd colum** Onset of annotation

**3rd colum** Offset of annotation

**4th colum** Duration of annotation

**5th colum** Description ...

**Source**

<https://osf.io/tj2n5>

---

NIR\_ABh\_Puriya\_Annotation\_Influence

*NIR\_ABh\_Puriya\_Annotation\_Influence*

---

**Description**

A subset of data from NIR\_ABh\_Puriya describing the annotated influence. The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

**Usage**

```
data(NIR_ABh_Puriya_Annotation_Influence)
```

**Format**

rda:

A data frame with 306 rows and 5 columns:

**Event** Type of event

**Onset time** Start of the event in seconds

**Offset time** End of the event in seconds

**Duration** Duration of the event in seconds

**Notes** Text notes ...

**Source**

<https://osf.io/ks325/>

---

NIR\_ABh\_Puriya\_Central\_Feature\_Sitar

*NIR\_ABh\_Puriya\_Central\_Feature\_Sitar*

---

**Description**

A subset of data from NIR\_ABh\_Puriya describing sitar pitch. Dummy data for demonstration purposes. The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

**Usage**

```
data(NIR_ABh_Puriya_Central_Feature_Sitar)
```

**Format**

rda:

A data frame with 1,501 rows and 3 columns:

**X** Frame (here 25 fps)

**Pitch** Pitch in Hz - Dummy data

**Smooth** Smooth - Dummy data ...

**Source**

<https://osf.io/tj2n5>

---

 NIR\_ABh\_Puriya\_Central\_Pose\_Sitar

*NIR\_ABh\_Puriya\_Central\_Pose\_Sitar*


---

### Description

A subset of data from NIR\_ABh\_Puriya the estimate pose of the sitar player, carried out with openpose. The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

### Usage

```
data(NIR_ABh_Puriya_Central_Pose_Sitar)
```

### Format

rda:

A data frame with 1,501 rows and 27 columns:

**X** frame number, 25 fps

**LEar\_x** X coordinate of Left Ear

**LEar\_y** Y coordinate of Left Ear

**LElbow\_x** X coordinate of Left Elbow

**LElbow\_y** Y coordinate of Left Elbow

**LEye\_x** X coordinate of Left Eye

**LEye\_y** Y coordinate of Left Eye

**LShoulder\_x** X coordinate of Left Shoulder

**LShoulder\_y** Y coordinate of Left Shoulder

**LWrist\_x** X coordinate of Left Wrist

**LWrist\_y** Y coordinate of Left Wrist

**MidHip\_x** X coordinate of Left MidHip

**MidHip\_y** Y coordinate of Left MidHip

**Neck\_x** X coordinate of Left Neck

**Neck\_y** Y coordinate of Left Neck

**Nose\_x** X coordinate of Left Nose

**Nose\_y** Y coordinate of Left Nose

**REar\_x** X coordinate of Right Ear

**REar\_y** Y coordinate of Right Ear

**RElbow\_x** X coordinate of Right Elbow

**RElbow\_y** Y coordinate of Right Elbow

**REye\_x** X coordinate of Right Eye

**REye\_y** Y coordinate of Right Eye

**RShoulder\_x** X coordinate of Right Shoulder



**RShoulder\_y** Y coordinate of Right Shoulder  
**RWrist\_x** X coordinate of Right Wrist  
**RWrist\_y** Y coordinate of Right Wrist ...

### Source

<https://osf.io/tj2n5>

---

NIR\_ABh\_Puriya\_Central\_Pose\_Tabla

*NIR\_ABh\_Puriya\_Central\_Pose\_Tabla*

---

### Description

A subset of data from NIR\_ABh\_Puriya the estimate pose of the tabla player. The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

### Usage

`data(NIR_ABh_Puriya_Central_Pose_Tabla)`

### Format

rda:

A data frame with 1,501 rows and 27 columns:

**X** frame number, here 25 fps

**LEar\_x** X coordinate of Left Ear

**LEar\_y** Y coordinate of Left Ear

**LElbow\_x** X coordinate of Left Elbow

**LElbow\_y** Y coordinate of Left Elbow

**LEye\_x** X coordinate of Left Eye

**LEye\_y** Y coordinate of Left Eye

**LShoulder\_x** X coordinate of Left Shoulder

**LShoulder\_y** Y coordinate of Left Shoulder

**LWrist\_x** X coordinate of Left Wrist

**LWrist\_y** Y coordinate of Left Wrist

**MidHip\_x** X coordinate of Left MidHip

**MidHip\_y** Y coordinate of Left MidHip

**Neck\_x** X coordinate of Left Neck

**Neck\_y** Y coordinate of Left Neck

**Nose\_x** X coordinate of Left Nose

**Nose\_y** Y coordinate of Left Nose

**REar\_x** X coordinate of Right Ear  
**REar\_y** Y coordinate of Right Ear  
**RElbow\_x** X coordinate of Right Elbow  
**RElbow\_y** Y coordinate of Right Elbow  
**REye\_x** X coordinate of Right Eye  
**REye\_y** Y coordinate of Right Eye  
**RShoulder\_x** X coordinate of Right Shoulder  
**RShoulder\_y** Y coordinate of Right Shoulder  
**RWrist\_x** X coordinate of Right Wrist  
**RWrist\_y** Y coordinate of Right Wrist ...

### Source

<https://osf.io/tj2n5>

---

NIR\_ABh\_Puriya\_Metre\_DrutTeental

*NIR\_ABh\_Puriya\_Metre\_DrutTeental*

---

### Description

A subset of data from NIR\_ABh\_Puriya Describing Metre (Cycle numbers and onset times). The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

### Usage

```
data(NIR_ABh_Puriya_Metre_DrutTeental)
```

### Format

rda:

A data frame with 351 rows and 3 columns:

**Cycle** Number of the Cycle

**Time** Time in seconds

**Notes** text which is empty for this file ...

### Source

<https://osf.io/fzv3k>

---

NIR\_ABh\_Puriya\_Metre\_VilambitTeental  
*NIR\_ABh\_Puriya\_Metre\_VilambitTeental*

---

**Description**

A subset of data from NIR\_ABh\_Puriya describing the metre in Vilambit Teental section.

**Usage**

```
data(NIR_ABh_Puriya_Metre_VilambitTeental)
```

**Format**

rda:

A data frame with 72 rows and 4 columns:

**Cycle** Number of the Cycle

**Time** Time in seconds

**Notes** text which is empty for this file

**Beats** Number of beats in the cycle ...

**Details**

The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

**Source**

<https://osf.io/dyu68>

---

NIR\_ABh\_Puriya\_Onsets\_Selected\_DrutTeental  
*NIR\_ABh\_Puriya\_Onsets\_Selected\_DrutTeental*

---

**Description**

A subset of data from NIR\_ABh\_Puriya containing information about selected onsets for Drut Teental section. The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

**Usage**

```
data(NIR_ABh_Puriya_Onsets_Selected_DrutTeental)
```

**Format**

rda:

A data frame with 5,585 rows and 20 columns:

**Session** Session name

**Inst.Name** Instrument Name

**Tala** Tala name

**Label** Label for beat (111)

**Matra** Matra number

**Half.beat** logical On or Off

**Half** integer (1) for logical on or Off

**Misc.1** Descriptor e.g. 'Gat'

**Misc.2** Another descriptor, usually missing

**Cadence** Descriptor

**Tabla.solo** Descriptor where N is 'No'

**Inst** Onset time in seconds

**Tabla** Onset time in seconds of tabla

**Inst.Density** Calculated density of onsets (no/s)

**Tabla.Density** Calculated density of onsets (no/s)

**Inst.Peak** Peak of the onset (onset strength)

**Tabla.Peak** Peak of the onset (onset strength)

**Inst.Player** Name of the performer (sitar)

**Tabla.Player** Name of the performer (tabla)

**Chunk** Chunk name ...

**Source**

<https://osf.io/phv6b>

---

NIR\_ABh\_Puriya\_Onsets\_Selected\_VilambitTeental

*NIR\_ABh\_Puriya\_Onsets\_Selected\_VilambitTeental*

---

**Description**

A subset of data from NIR\_ABh\_Puriya containing information about selected onsets for Vilambit Teental section (sitar and tabla). The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

**Usage**

data(NIR\_ABh\_Puriya\_Onsets\_Selected\_VilambitTeental)

**Format**

rda:

A data frame with 2,275 rows and 20 columns:

**Session** Session name**Inst.Name** Instrument Name**Tala** Tala name**Label** Label for beat (111)**Matra** Matra number**Half.beat** logical On or Off**Half** integer (1) for logical on or Off**Misc.1** Descriptor e.g. 'Gat'**Misc.2** Another descriptor, usually missing**Cadence** Descriptor**Tabla.solo** Descriptor where N is 'No'**Inst** Onset time in seconds**Tabla** Onset time in seconds of tabla**Inst.Density** Calculated density of onsets (no/s)**Tabla.Density** Calculated density of onsets (no/s)**Inst.Peak** Peak of the onset (onset strength)**Tabla.Peak** Peak of the onset (onset strength)**Inst.Player** Name of the performer (sitar)**Tabla.Player** Name of the performer (tabla)**Chunk** Chunk name ...**Source**<https://osf.io/xcefp>

---

NIR\_ABh\_Puriya\_OptFlow\_Central\_Sitar

*NIR\_ABh\_Puriya\_OptFlow\_Central\_Sitar*

---

**Description**

A subset of data from NIR\_ABh\_Puriya describing the head movement of the sitar player extracted using Optical Flow giving X and Y coordinates. The data comes from a collection of audiovisual recordings of North Indian (Hindustani) raga performances which are part of IEMP North Indian Raga collection, collected and curated by Martin Clayton, Laura Leante, and Simone Tarsitani.

**Usage**

```
data(NIR_ABh_Puriya_OptFlow_Central_Sitar)
```

**Format**

rda:

A data frame with 1,501 rows and 4 columns:

**Frame** Frame (integer, related 25 fps)**Time** Time in seconds**X** X coordinate**Y** Y coordinate ...**Source**<https://osf.io/r4xza>

---

`open_movementsync_data`*Opens movementsync data home page at OSF*

---

**Description**

Opens movementsync data home page at OSF

**Usage**`open_movementsync_data()`**Value**

No return value, opens a browser on a specific OSF page

---

`plot.Duration`*Plot a Duration S3 object*

---

**Description**

Plot a Duration S3 object

**Usage**

```
## S3 method for class 'Duration'
plot(x, ...)
```

**Arguments**

<code>x</code>	S3 object
<code>...</code>	passed to <code>barplot()</code>

**Value**

a plot object with durations.

**Examples**

```
r <- get_sample_recording()
d <- get_duration_annotation_data(r)
plot(d)
```

---

plot.GrangerInteraction

*Plot network diagram of Granger Causalities*

---

**Description**

Plot network diagram of Granger Causalities

**Usage**

```
## S3 method for class 'GrangerInteraction'
plot(x, mfrow = NULL, mar = c(1, 1, 1, 1), oma = c(1, 1, 1, 1), ...)
```

**Arguments**

x	GrangerInteraction object
mfrow	passed to <a href="#">par()</a>
mar	passed to <a href="#">par()</a>
oma	passed to <a href="#">par()</a>
...	passed through to <a href="#">plot.igraph()</a>

**Value**

data.frame of P-Values

**See Also**

Other Granger Causality: [autoplot.GrangerTime\(\)](#), [get\\_granger\\_interactions\(\)](#), [granger\\_test\(\)](#), [map\\_to\\_granger\\_test\(\)](#), [ms\\_condgrangertest\(\)](#), [ms\\_grangertest1\(\)](#), [ms\\_grangertest2\(\)](#), [plot\\_influence\\_diagram\(\)](#)

**Examples**

```

r <- get_recording("NIR_ABh_Puriya", fps = 25)
fv_list <- get_filtered_views(r, "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)
jv <- subset(jv, Time <= 5*60)
l <- list(a = c(0, 100), b = c(100, 200), c = c(200, 300))
splicing_df <- splice_time(l)
sv <- get_spliced_view(jv, splicing_df)
gi <- get_granger_interactions(sv, c("Nose_x_Central_Sitar", "Nose_x_Central_Tabla"), lag = 1/25)
print(gi)

```

---

plot.Metre

*Plot a Metre S3 object*


---

**Description**

Plot a Metre S3 object

**Usage**

```

## S3 method for class 'Metre'
plot(x, ...)

```

**Arguments**

x	S3 object.
...	ignored.

**Value**

a plot object with metre.

**Examples**

```

r <- get_sample_recording()
m <- get_metre_data(r)
plot(m)

```



---

plot.OnsetsSelected     *Plot a OnsetsSelected S3 object*

---

**Description**

Plot a OnsetsSelected S3 object

**Usage**

```
## S3 method for class 'OnsetsSelected'  
plot(x, instrument = "Inst", tactus = "Matra", ...)
```

**Arguments**

x	S3 object.
instrument	column name.
tactus	beat column name (defaults to "Matra").
...	passed to <a href="#">barplot()</a> .

**Value**

Return an 'OnsetsSelected' object.

**Examples**

```
r <- get_sample_recording()  
o <- get_onsets_selected_data(r)  
plot(o)
```

---

plot.View     *Plot a View S3 object*

---

**Description**

Plot a View S3 object

**Usage**

```
## S3 method for class 'View'  
plot(x, columns = NULL, maxpts = 1000, ...)
```

**Arguments**

x	S3 object
columns	names of columns
maxpts	maximum number of points to plot.
...	passed to <code>plot.zoo()</code>

**Value**

a plot object.

**Examples**

```
r <- get_sample_recording()
v <- get_raw_view(r, "Central", "", "Sitar")
plot(v, columns = "LEar_x")
```

---

plot\_average\_coherency

*Plot average coherency of a coherency object*

---

**Description**

Plot average coherency of a coherency object

**Usage**

```
plot_average_coherency(obj, view, ...)
```

**Arguments**

obj	analyze.coherency object.
view	View object.
...	passed to <code>WaveletComp::wc.avg()</code> .

**Value**

a ggplot object.

**See Also**

Other wavelet functions: `analyze_coherency()`, `analyze_wavelet()`, `get_local_max_average_power()`, `plot_average_power()`, `plot_cross_spectrum()`, `plot_cwt_energy()`, `plot_phase_difference()`, `plot_power_spectrum()`, `plot_roll_resultant_length()`, `plot_sel_phases()`, `plot_wt_energy()`

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
co <- analyze_coherency(pv, columns = c("Nose_x", "Nose_y"))
plot_average_coherency(co, pv)
```

---

plot\_average\_power      *Plot average power of a wavelet object*

---

**Description**

Plot average power of a wavelet object

**Usage**

```
plot_average_power(obj, view, ...)
```

**Arguments**

obj	analyze.wavelet object.
view	View object.
...	passed to <code>WaveletComp::wt.avg()</code> .

**Value**

a ggplot object.

**See Also**

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
pv1 <- subset(pv, Time >= 10)
w <- analyze_wavelet(pv1, "Nose_x")
plot_average_power(w, pv1)
w <- analyze_wavelet(pv1, "Nose_y")
plot_average_power(w, pv1)
```

---

plot\_cross\_spectrum    *Plot a coherency of a wavelet object*

---

### Description

Plot a coherency of a wavelet object

### Usage

```
plot_cross_spectrum(obj, view, ...)
```

```
plot_coherence(obj, view, ...)
```

### Arguments

obj	analyze.coherency object.
view	View object.
...	passed to <code>WaveletComp::wc.image()</code> .

### Value

a list of class graphical parameters,

### See Also

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

### Examples

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
pv1 <- subset(pv, Time >= 10)
co <- analyze_coherency(pv1, c("Nose_x", "Nose_y"))
plot_cross_spectrum(co, pv1)
plot_coherence(co, pv1)
```

---

plot\_cwt\_energy      *Plot cross wavelet energy of a wavelet object*

---

**Description**

Plot cross wavelet energy of a wavelet object

**Usage**

```
plot_cwt_energy(obj, view)
```

**Arguments**

obj                  analyze.wavelet object.  
view                 View object.

**Value**

a ggplot object.

**See Also**

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
co <- analyze_coherency(pv, columns = c("Nose_x", "Nose_y"))
plot_cwt_energy(co, pv)
```

---

plot\_history\_xy      *Plot a set of data points over time*

---

**Description**

Plot a set of data points over time

**Usage**

```
plot_history_xy(obj, maxpts = 10000)
```

**Arguments**

obj                   View object.  
 maxpts               maximum number of points to plot.

**Value**

a ggplot object.

**Examples**

```
r1 <- get_sample_recording()
rv1 <- get_raw_view(r1, "Central", "", "Sitar")
pv1 <- get_processed_view(rv1)
fv1 <- apply_filter_sgolay(pv1, data_points = c("LElbow", "RElbow"), n = 41, p = 3)
sub_fv1 <- subset(fv1, Time >= 0 & Time <= 100, by = 10)
plot_history_xy(sub_fv1)
```

---

plot\_influence\_diagram

*Plot influence diagram from a GrangerTest object*

---

**Description**

Arrows show causality (influencing) direction.

**Usage**

```
plot_influence_diagram(obj, splicing_df, two_arrows = TRUE, lev_sig = 0.05)
```

**Arguments**

obj                   GrangerTest object  
 splicing\_df          Splicing data.frame object  
 two\_arrows          plot influence arrows both ways? (Default is TRUE).  
 lev\_sig              significance level

**Details**

By default two\_arrows is TRUE and an influencing arrow is drawn for each significant p-value. If two\_arrows is FALSE and one of the p-values is significant then  $-\log_{10}(p\_value)$  difference is plotted i.e

**Value**

ggplot object

**See Also**

Other Granger Causality: [autoplot.GrangerTime\(\)](#), [get\\_granger\\_interactions\(\)](#), [granger\\_test\(\)](#), [map\\_to\\_granger\\_test\(\)](#), [ms\\_condgrangertest\(\)](#), [ms\\_grangertest1\(\)](#), [ms\\_grangertest2\(\)](#), [plot.GrangerInteraction\(\)](#)

**Examples**

```
r1 <- get_sample_recording()
fv_list <- get_filtered_views(r1, data_points = "Nose", n = 41, p = 3)
jv_sub <- get_joined_view(fv_list)
splicing_df <- splice_time(jv_sub, win_size = 3, step_size = 0.5)
sv <- get_spliced_view(jv_sub, splicing_df)
g <- granger_test(sv, "Nose_x_Central_Sitar", "Nose_x_Central_Tabla", lag = 3/25)

plot_influence_diagram(g, splicing_df)
plot_influence_diagram(g, splicing_df, two_arrows = TRUE)

d1 <- get_duration_annotation_data(r1)
plot_influence_diagram(g, splicing_df) +
  autolayer(d1, expr = (Tier == "Influence S>T" | Tier == "Influence T>S") & Out <= 60,
            fill_col = "Tier")
```

---

plot\_phase\_difference *Plot a coherency of a wavelet object*

---

**Description**

Plot a coherency of a wavelet object

**Usage**

```
plot_phase_difference(obj, view, ...)
```

**Arguments**

obj	analyze.coherency object.
view	View object.
...	passed to <a href="#">WaveletComp::wc.phasediff.image()</a> .

**Value**

a list of class graphical parameters

**See Also**

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

## Examples

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
pv1 <- subset(pv, Time >= 10 & Time <= 20)
co <- analyze_coherency(pv1, c("Nose_x", "Nose_y"))
plot_phase_difference(co, pv1)
```

---

plot\_power\_spectrum *Plot a power spectrum of a wavelet object*

---

## Description

Plot a power spectrum of a wavelet object

## Usage

```
plot_power_spectrum(obj, view, ...)
```

## Arguments

obj	analyze.wavelet object.
view	View object.
...	passed to <code>WaveletComp::wt.image()</code> .

## Value

a list of class graphical parameters.

## See Also

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

## Examples

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
pv1 <- subset(pv, Time >= 30)
w <- analyze_wavelet(pv1, "Nose_y")
plot_power_spectrum(w, pv1)
w <- analyze_wavelet(pv1, "Nose_y", lowerPeriod = 0.01, upperPeriod = 10)
plot_power_spectrum(w, pv1)
```



---

plot\_roll\_resultant\_length  
*Plot windowed resultant length*

---

## Description

Plot windowed resultant length

## Usage

```
plot_roll_resultant_length(  
  obj,  
  window_duration = 1,  
  smooth = FALSE,  
  by = 1,  
  ref_lines = c(W = 0.7, M = 0.85, H = 0.95),  
  align = "right",  
  na.rm = TRUE  
)
```

## Arguments

obj	a sel.phases object.
window_duration	duration of window over which to take mean (default is 1 sec).
smooth	use the smoothed phase angle data (default is FALSE).
by	calculate resultant length at every by-th time point rather than every point.
ref_lines	names list of reference line values (default is c(W = 0.7, M = 0.85, H = 0.95)).
align	alignment of window (default is 'right').
na.rm	Remove NAs from the circular mean (default is TRUE).

## Value

a ggplot object.

## See Also

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_sel\\_phases\(\)](#), [plot\\_wt\\_energy\(\)](#)

**Examples**

```

r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
co <- analyze_coherency(pv, columns = c("Nose_x", "Nose_y"))
sp <- plot_sel_phases(co, pv, sel.period = 0.64)
plot_roll_resultant_length(sp, ref_lines = c(H = 0.9998))

```

---

plot_sel_phases	<i>Comparison plot of phases of a coherency object</i>
-----------------	--

---

**Description**

Comparison plot of phases of a coherency object

**Usage**

```

plot_sel_phases(
  obj,
  view,
  sel.period = NULL,
  sel.upper = NULL,
  sel.lower = NULL,
  ...
)

```

**Arguments**

obj	coherency object.
view	View object.
sel.period	a single number which determines the (closest available) Fourier period to be selected. Default: NULL.
sel.upper	a number to define an upper Fourier period (or the closest available) for the selection of a band of periods (effective if sel.period is NULL). Default: NULL.
sel.lower	a number to define a lower Fourier period (or the closest available) for the selection of a band of periods (effective if sel.period is NULL). Default: NULL.
...	passed to <a href="#">WaveletComp::wc.sel.phases()</a> .

**Value**

an object of class sel.phases.

**See Also**

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_wt\\_energy\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
co <- analyze_coherency(pv, columns = c("Nose_x", "Nose_y"))
plot_cross_spectrum(co, pv)
plot_sel_phases(co, pv, sel.period = 0.64)
plot_sel_phases(co, pv, sel.lower = 0.6, sel.upper = 0.8)
```

---

plot_wt_energy	<i>Plot wavelet energy of a wavelet object</i>
----------------	--

---

**Description**

Plot wavelet energy of a wavelet object

**Usage**

```
plot_wt_energy(obj, view)
```

**Arguments**

obj	analyze.wavelet object.
view	View object.

**Value**

a ggplot object.

**See Also**

Other wavelet functions: [analyze\\_coherency\(\)](#), [analyze\\_wavelet\(\)](#), [get\\_local\\_max\\_average\\_power\(\)](#), [plot\\_average\\_coherency\(\)](#), [plot\\_average\\_power\(\)](#), [plot\\_cross\\_spectrum\(\)](#), [plot\\_cwt\\_energy\(\)](#), [plot\\_phase\\_difference\(\)](#), [plot\\_power\\_spectrum\(\)](#), [plot\\_roll\\_resultant\\_length\(\)](#), [plot\\_sel\\_phases\(\)](#)

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
pv1 <- subset(pv, Time >= 10)
w <- analyze_wavelet(pv1, "Nose_x")
plot_wt_energy(w, pv1)
```

---

pull\_segment\_spliceview

*Apply function to SplicedView and pull out element from output*

---

### Description

Apply function to SplicedView and pull out element from output

### Usage

```
pull_segment_spliceview(sv, FUN, element, ...)
```

### Arguments

sv	SplicedView object.
FUN	function to apply.
element	name of element to pull out from output object.
...	passed to function.

### Value

list with output and input fields.

### See Also

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

### Examples

```
r <- get_sample_recording()
d1 <- get_duration_annotation_data(r)
# only one relevant section for sample data
splicing_smile_df <- splice_time(d1, tier = 'INTERACTION',
  comments = 'Mutual look and smile')

fv_list <- get_filtered_views(r, data_points = "Nose", n = 41, p = 3)
jv <- get_joined_view(fv_list)
sv_duration_smile <- get_spliced_view(jv, splicing_df = splicing_smile_df)
pull_segment_spliceview(sv_duration_smile, FUN = analyze_wavelet,
  column = "Nose_x_Central_Sitar", element = 'Power')
```

---

sample_gap_splice	<i>Randomly create matching segments from a splicing table without overlaps</i>
-------------------	---

---

### Description

Works by randomly varying the gaps between segments assuming that the gap number follow a Poisson process with rate given by the average sample gap length in the input splice. Durations of segments remain the same.

### Usage

```
sample_gap_splice(splicing_dfr, v, num_splices, rejection_list = list())
```

### Arguments

splicing\_dfr Splice object.  
v View object.  
num\_splices number of random splices to generate.  
rejection\_list list of Splice objects for rejection.

### Details

Uses rejection sampling to avoid overlaps with the input segments and additional segments from a list of splices.

### Value

list of splicing data.frames.

### See Also

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

### Examples

```
r1 <- get_sample_recording()
d1 <- get_duration_annotation_data(r1)
rv1 <- get_raw_view(r1, "Central", "", "Sitar")
splicing_df <- splice_time(d1, tier = 'INTERACTION', comments = 'Mutual look and smile')
# Only first segment relevant for sample data
x <- sample_gap_splice(splicing_df[1,], rv1, num_splices = 10)
```

---

sample\_offset\_splice *Randomly create matching segments from a splicing table without overlaps*

---

### Description

Works by adding a random offset to each start time in the splice. Uses rejection sampling to avoid overlaps with the input segments and additional segments from a list of splices.

### Usage

```
sample_offset_splice(splicing_dfr, v, num_splices, rejection_list = list())
```

### Arguments

splicing\_dfr Splice object.  
 v View object.  
 num\_splices number of random splices to generate.  
 rejection\_list list of Splice objects for rejection.

### Value

list of splicing data.frames.

### See Also

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [summary\\_onsets\(\)](#), [visualise\\_sample\\_splices\(\)](#)

### Examples

```
r1 <- get_sample_recording()
d1 <- get_duration_annotation_data(r1)
rv1 <- get_raw_view(r1, "Central", "", "Sitar")
splicing_df <- splice_time(d1, tier = 'INTERACTION', comments = 'Mutual look and smile')
# Only first segment relevant for sample data
x <- sample_offset_splice(splicing_df[1,], rv1, num_splices = 100)
```

---

`sample_time_spliced_views`*Sample the time line from a list of Views*

---

## Description

Sample the time line from a list of Views

## Usage

```
sample_time_spliced_views(  
  ...,  
  num_samples,  
  replace = FALSE,  
  na.action = stats::na.pass  
)
```

## Arguments

...	names arguments of SplicedView objects.
num_samples	number of time points to sample
replace	sample with replacement (default is FALSE)?
na.action	function to deal with NAs in data (default is na.pass).

## Value

a list of SplitView object or a SplitView object

## Examples

```
r1 <- get_sample_recording()  
fv1_list <- get_filtered_views(r1, data_points = "Nose", n = 41, p = 3)  
jv1 <- get_joined_view(fv1_list)  
l <- list(a=c(1, 2), b = c(2, 3))  
splicing_df <- splice_time(l)  
sv <- get_spliced_view(jv1, splicing_df = splicing_df)  
autoplot(sv)  
sv_new <- sample_time_spliced_views(sv, num_samples = 10, replace = FALSE)  
autoplot(sv_new)  
sv_new <- sample_time_spliced_views(sv, num_samples = 10, replace = TRUE)  
autoplot(sv_new)  
l <- list(a=c(1, 2), a = c(10, 20), b = c(30, 40))  
splicing_df <- splice_time(l)  
sv <- get_spliced_view(jv1, splicing_df = splicing_df)  
sv_new <- sample_time_spliced_views(sv, num_samples = 20, replace = TRUE)  
autoplot(sv_new)
```

---

specgram_plot	<i>Specgram Plot</i>
---------------	----------------------

---

**Description**

Specgram Plot

**Usage**

```
specgram_plot(obj, ...)
```

**Arguments**

obj	View object.
...	passed to <code>signal::specgram()</code> .

**Value**

a ggplot object.

**Examples**

```
r <- get_recording("NIR_ABh_Puriya", fps = 25)
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
sub_pv <- subset(pv, Time >= 15 & Time <= 25, columns = c("RWrist_x", "RWrist_y"))
specgram_plot(sub_pv)
fv <- apply_filter_sgolay(pv, data_points = c("RWrist"), n = 11, p = 4)
sub_fv <- subset(fv, Time >= 15 & Time <= 25, columns = c("RWrist_x", "RWrist_y"))
specgram_plot(sub_fv)
specgram_plot(sub_fv, window = 200) + ggplot2::scale_fill_gradient(low = "white", high = "black")
```

---

spectral_density	<i>Estimate the spectral density of data points</i>
------------------	---

---

**Description**

Estimates the periodicity of data points in a View object.

**Usage**

```
spectral_density(view, columns = NULL, data_points = NULL, ...)
```



**Arguments**

view            ProcessedView or FilteredView object.  
 columns        names of data columns e.g. Nose\_x.  
 data\_points    data points to process e.g. Nose.  
 ...            passed to `stats::spectrum()`.

**Value**

SpectralDensityView object.

**Examples**

```
r<-get_recording("NIR_ABh_Puriya", fps=25)
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
sd1 <- spectral_density(pv, columns = "LEar_x", spans = 5)

fv <- apply_filter_sgolay(pv, data_points = c("LEye"), n = 19, p = 4)
sd1 <- spectral_density(fv, data_points = c("LEye"), spans = 5)
```

---

splice\_time                      *S3 generic function to splice a timeline*

---

**Description**

S3 generic function to splice a timeline

**Usage**

```
splice_time(x, ...)
```

**Arguments**

x                S3 object.  
 ...             passed to relevant method.

**Value**

a Splice object.

**See Also**

Other splicing functions: `clip_splice()`, `get_spliced_view()`, `is_splice_overlapping()`, `merge_splice()`, `splice_time.Duration()`, `splice_time.Metre()`, `splice_time.OnsetsDifference()`, `splice_time.View()`, `splice_time.list()`, `split.SplicedView()`

---

splice\_time.Duration *Generate spliced timeline using a Duration object*

---

### Description

Generate spliced timeline using a Duration object

### Usage

```
## S3 method for class 'Duration'  
splice_time(  
  x,  
  expr = NULL,  
  make.unique = TRUE,  
  tier = NULL,  
  comments = NULL,  
  ...  
)
```

### Arguments

x	Duration object.
expr	R expression to filter data on.
make.unique	make the segments unique? (Default is TRUE).
tier	exact tier name to filter on.
comments	exact comment to filter on.
...	passed to <a href="#">make.unique()</a>

### Value

a Splice object.

### See Also

Other splicing functions: [clip\\_splice\(\)](#), [get\\_spliced\\_view\(\)](#), [is\\_splice\\_overlapping\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

### Examples

```
r <- get_sample_recording()  
d <- get_duration_annotation_data(r)  
splice_time(d, tier = 'Event', comments = 'tabla solo')
```

---

splice\_time.list      *Generate spliced timeline using a list*

---

### Description

Generate spliced timeline using a list

### Usage

```
## S3 method for class 'list'  
splice_time(x, ...)
```

### Arguments

x                    named list.  
...                  ignored.

### Value

a Splice object.

### See Also

Other splicing functions: [clip\\_splice\(\)](#), [get\\_spliced\\_view\(\)](#), [is\\_splice\\_overlapping\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

### Examples

```
l <- list(a = c(0, 10), b = c(10, 20), c = c(20, 30))  
splice_time(l)
```

---

splice\_time.Metre      *Generate spliced timeline using a Metre object*

---

### Description

Generate spliced timeline using a Metre object

**Usage**

```
## S3 method for class 'Metre'
splice_time(
  x,
  window_duration = NULL,
  window_proportion = NULL,
  tactus = NULL,
  ...
)
```

**Arguments**

x	Metre object.
window_duration	duration of window around beat (may lead to overlapping windows if large).
window_proportion	sets the window duration around beat based on a proportion (0, 0.5] of the gap to the previous and following cycles. The first and last beats in each Metre are removed.
tactus	vector of Metres to subset on.
...	ignored.

**Value**

a Splice object.

**See Also**

Other splicing functions: [clip\\_splice\(\)](#), [get\\_spliced\\_view\(\)](#), [is\\_splice\\_overlapping\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

**Examples**

```
r <- get_sample_recording()
m <- get_metre_data(r)
splicing_df <- splice_time(m, window_duration = 1)
head(splicing_df)
splicing_df <- splice_time(m, window_proportion = 0.25)
head(splicing_df)
```

---

`splice_time.OnsetsDifference`*Generate spliced timeline using an OnsetsDifference object*

---

## Description

Generate spliced timeline using an OnsetsDifference object

## Usage

```
## S3 method for class 'OnsetsDifference'  
splice_time(x, window_duration, metres = NULL, make.unique = TRUE, ...)
```

## Arguments

<code>x</code>	OnsetsDifference object.
<code>window_duration</code>	duration of window around onset point in seconds.
<code>metres</code>	vector of metres to subset.
<code>make.unique</code>	give unique names to each segment?
<code>...</code>	passed to <code>make.unique()</code> .

## Value

a Splice object.

## See Also

Other splicing functions: `clip_splice()`, `get_spliced_view()`, `is_splice_overlapping()`, `merge_splice()`, `splice_time.Duration()`, `splice_time.Metre()`, `splice_time.View()`, `splice_time.list()`, `splice_time()`, `split.SplicedView()`

## Examples

```
r <- get_sample_recording()  
o1 <- get_onsets_selected_data(r)  
po1 <- difference_onsets(o1, instruments = c('Inst', 'Tabla'))  
splicing_df <- splice_time(po1, window_duration = 1)  
head(splicing_df)
```

---

splice_time.View	<i>Generate spliced timeline using a view</i>
------------------	---

---

### Description

Generate spliced timeline using a view

### Usage

```
## S3 method for class 'View'  
splice_time(x, win_size, step_size, ...)
```

### Arguments

x	View object.
win_size	duration of window segment in seconds.
step_size	increment in seconds between segments.
...	ignored.

### Value

a Splice object.

### See Also

Other splicing functions: [clip\\_splice\(\)](#), [get\\_spliced\\_view\(\)](#), [is\\_splice\\_overlapping\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#), [split.SplicedView\(\)](#)

### Examples

```
r <- get_sample_recording()  
rv <- get_raw_view(r, "Central", "", "Sitar")  
df <- splice_time(rv, win_size = 3, step_size = 0.5)  
head(df)
```

---

split.SplicedView	<i>Get a list of Views from a SplicedView</i>
-------------------	---

---

### Description

Get a list of Views from a SplicedView

### Usage

```
## S3 method for class 'SplicedView'  
split(x, f, drop, ...)
```

### Arguments

x	SplicedView object.
f	ignored.
drop	ignored.
...	ignored.

### Value

list of View objects.

### See Also

Other splicing functions: [clip\\_splice\(\)](#), [get\\_spliced\\_view\(\)](#), [is\\_splice\\_overlapping\(\)](#), [merge\\_splice\(\)](#), [splice\\_time.Duration\(\)](#), [splice\\_time.Metre\(\)](#), [splice\\_time.OnsetsDifference\(\)](#), [splice\\_time.View\(\)](#), [splice\\_time.list\(\)](#), [splice\\_time\(\)](#)

### Examples

```
r <- get_sample_recording()  
rv <- get_raw_view(r, "Central", "", "Sitar")  
pv <- get_processed_view(rv)  
l <- list(a = c(0, 10), b = c(10, 20), c = c(20, 30))  
splicing_df <- splice_time(l)  
sv <- get_spliced_view(pv, splicing_df)  
v_list <- split(sv)
```

---

subset.View

*Subset a View*


---

**Description**

Simple time and column subsetting of views.

**Usage**

```
## S3 method for class 'View'
subset(x, expr = NULL, data_points = NULL, columns = NULL, by = NULL, ...)
```

**Arguments**

x	View object
expr	an R expression to subset time or other variables.
data_points	body part in the data e.g. 'Nose'.
columns	column name in the data e.g. 'Nose_x'.
by	increment of the sequence of rows to return.
...	unused.

**Value**

a View object.

**Examples**

```
r <- get_sample_recording()
v <- get_raw_view(r, "Central", "", "Sitar")
vv <- subset(v, Time < 10, data_point = "Nose")
plot(vv)
```

---

summary.analyze.wavelet

*Summarise an analyze.wavelet object*


---

**Description**

Summarise an analyze.wavelet object

**Usage**

```
## S3 method for class 'analyze.wavelet'
summary(object, v, ...)
```



**Arguments**

object	analyze.wavelet object.
v	View object
...	ignored.

**Value**

data.frame

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
w <- analyze_wavelet(pv, "Nose_x")
summary(w, pv)
```

---

summary.Duration	<i>Summarise Duration object</i>
------------------	----------------------------------

---

**Description**

Summarise Duration object

**Usage**

```
## S3 method for class 'Duration'
summary(object, ...)
```

**Arguments**

object	Duration object.
...	ignored.

**Value**

data.frame

**Examples**

```
r <- get_sample_recording()
d <- get_duration_annotation_data(r)
head(summary(d))
```

---

summary.Metre	<i>Summarise Metre object</i>
---------------	-------------------------------

---

**Description**

Summarises the cycle length for each Metre.

**Usage**

```
## S3 method for class 'Metre'
summary(object, ...)
```

**Arguments**

object	Metre object.
...	ignored.

**Value**

list of summaries.

**Examples**

```
r <- get_sample_recording()
m <- get_metre_data(r)
summary(m)
```

---

summary.OnsetsSelected	<i>Summarise OnsetsSelected object</i>
------------------------	--

---

**Description**

Summarise OnsetsSelected object

**Usage**

```
## S3 method for class 'OnsetsSelected'
summary(object, ...)
```

**Arguments**

object	OnsetsSelected object.
...	ignored.

**Value**

list of summaries.

**Examples**

```
r <- get_sample_recording()
o <- get_onsets_selected_data(r)
summary(o)
```

---

summary.Recording      *Summarise Recording object*

---

**Description**

Summarise Recording object

**Usage**

```
## S3 method for class 'Recording'
summary(object, ...)
```

**Arguments**

object	Recording object.
...	ignored.

**Value**

list

**Examples**

```
r <- get_sample_recording()
summary(r)
```

---

summary.sel.phases      *Summarises a sel.phases object*

---

### Description

Summarises a sel.phases object

### Usage

```
## S3 method for class 'sel.phases'
summary(object, na.rm = TRUE, ...)
```

### Arguments

object	sel.phases object.
na.rm	remove missings?
...	ignored.

### Value

list of Circular statistics.

### Examples

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
co <- analyze_coherency(pv, columns = c("Nose_x", "Nose_y"))
sp <- plot_sel_phases(co, pv, sel.period = NULL, sel.lower = 0.5, sel.upper = 0.7)
summary(sp)
```

---

summary.View      *Summarise a View object*

---

### Description

Summarise a View object

### Usage

```
## S3 method for class 'View'
summary(object, ...)
```

### Arguments

object	View object.
...	ignored.

**Value**

summary of data.frame.

**Examples**

```
r <- get_sample_recording()
rv <- get_raw_view(r, "Central", "", "Sitar")
pv <- get_processed_view(rv)
fv <- apply_filter_sgolay(pv, c("Nose", "RWrist", "LWrist"), n=19, p=4)
summary(rv)
summary(pv)
summary(fv)
```

---

summary_onsets	<i>Summary of difference in onsets</i>
----------------	--

---

**Description**

Summary of difference in onsets

**Usage**

```
summary_onsets(
  onset_obj,
  recording,
  instruments,
  splicing_dfr = NULL,
  expr = NULL,
  show_plot = FALSE,
  filter_pair = NULL,
  na_omit = TRUE,
  time_breaks = NULL
)
```

**Arguments**

onset_obj	OnsetsSelected object.
recording	Recording object.
instruments	character vector of instrument names.
splicing_dfr	Splice object
expr	R expression to subset onsetsSelected
show_plot	show a plot? (Default is FALSE).
filter_pair	regular expression to filter instrument pair names.
na_omit	omit NAs (Default is TRUE).
time_breaks	suggests the number of major time tick marks (default is NULL).

**Value**

a summary data frame of onset difference statistics.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [visualise\\_sample\\_splices\(\)](#)

**Examples**

```
r1 <- get_sample_recording()
o1 <- get_onsets_selected_data(r1)
d1 <- get_duration_annotation_data(r1)
splice_dfr <- splice_time(d1, tier = 'FORM')
summary_onsets(o1, r1, instruments = c('Inst', 'Tabla'),
  splicing_dfr = splice_dfr, show_plot = TRUE)
```

---

velocity\_dp

*Velocity plot of a view object*

---

**Description**

Velocity plot of a view object

**Usage**

```
velocity_dp(obj, add_mean = TRUE, vscale = 5, maxpts = 10000, alpha = 0.5, ...)
```

**Arguments**

obj	View object.
add_mean	add the mean to each line? (default is TRUE).
vscale	a vertical scaling to apply to the plot (default is 5).
maxpts	maximum number of points to plot.
alpha	ggplot aesthetic value.
...	passed to <a href="#">ggplot2::geom_point()</a> ,

**Value**

a ggplot object.

## Examples

```
r1 <- get_sample_recording()
rv1 <- get_raw_view(r1, "Central", "", "Sitar")
pv1 <- get_processed_view(rv1)
dp <- c("LWrist", "RWrist", "LElbow", "RElbow", "LEye", "REye", "Neck", "MidHip")
fv1 <- apply_filter_sgolay(pv1, data_point = dp, n = 41, p = 4)
sub_fv1 <- subset(fv1, Time >= 10 & Time <= 20, by = 2)
velocity_dp(sub_fv1)
```

---

visualise\_sample\_splices

*Visualise random splices*

---

## Description

Visualise random splices

## Usage

```
visualise_sample_splices(  
  splicing_df,  
  splicing_list,  
  jv,  
  overlay = TRUE,  
  avoid_splice_list = list(),  
  unstack = FALSE  
)
```

## Arguments

splicing_df	Splice object.
splicing_list	a list of Splice objects.
jv	JoinedView object.
overlay	overlay the segments for a density plot?
avoid_splice_list	list of Splice objects that determine times not to sample.
unstack	overlay segments on top of each other? (default is FALSE).

## Value

a ggplot object.

**See Also**

Other statistical and analysis functions: [apply\\_column\\_spliceview\(\)](#), [apply\\_segment\\_spliceview\(\)](#), [ave\\_cross\\_power\\_over\\_splices\(\)](#), [ave\\_cross\\_power\\_spliceview\(\)](#), [ave\\_power\\_over\\_splices\(\)](#), [ave\\_power\\_spliceview\(\)](#), [calculate\\_ave\\_cross\\_power1\(\)](#), [calculate\\_ave\\_power1\(\)](#), [compare\\_ave\\_cross\\_power1\(\)](#), [compare\\_ave\\_power1\(\)](#), [compare\\_avg\\_cross\\_power2\(\)](#), [compare\\_avg\\_power2\(\)](#), [difference\\_onsets\(\)](#), [pull\\_segment\\_spliceview\(\)](#), [sample\\_gap\\_splice\(\)](#), [sample\\_offset\\_splice\(\)](#), [summary\\_onsets\(\)](#)

**Examples**

```
r <- get_sample_recording()
fv_list <- get_filtered_views(r, data_points = 'Nose', n = 41, p = 3)
jv <- get_joined_view(fv_list)
splicing_df <- splice_time(list(a = c(0, 5), b = c(10, 15)))
splicing_list <- sample_offset_splice(splicing_df, jv, num_splices = 20)
visualise_sample_splices(splicing_df, splicing_list, jv)
```

---

xlim\_duration

*Get a ggplot2 xlim object based on duration data*


---

**Description**

Get a ggplot2 xlim object based on duration data

**Usage**

```
xlim_duration(object, expr = .data$Tier == "Form")
```

**Arguments**

object	Duration object.
expr	R expression to subset rows.

**Value**

a 'Duration' object.

**Examples**

```
r<-get_recording("NIR_ABh_Puriya", fps=25)
m <- get_metre_data(r)
d <- get_duration_annotation_data(r)
autoplot(m)
autoplot(m) + autolayer(d)
v <- get_raw_view(r, "Central", "", "Sitar")
autoplot(v, columns = c("LEar_x", "LEar_y")) + autolayer(d)
autoplot(v, columns = c("LEar_x", "LEar_y")) +
xlim_duration(d, expr = Tier == "FORM" & substr(Comments, 1, 1) == "J") +
autolayer(d, expr = Tier == "FORM" & substr(Comments, 1, 1) == "J")
```



# Index

## \* Granger Causality

- autoplot.GrangerTime, 14
- get\_granger\_interactions, 34
- granger\_test, 46
- map\_to\_granger\_test, 48
- ms\_condgrangertest, 51
- ms\_grangertest1, 52
- ms\_grangertest2, 53
- plot.GrangerInteraction, 63
- plot\_influence\_diagram, 70

## \* data functions

- apply\_filter\_sgolay, 9
- get\_data\_points, 31
- get\_duration\_annotation\_data, 31
- get\_feature\_data, 32
- get\_filtered\_views, 33
- get\_joined\_view, 35
- get\_metre\_data, 37
- get\_onsets\_selected\_data, 38
- get\_processed\_view, 39
- get\_processed\_views, 40
- get\_raw\_optflow\_view, 41
- get\_raw\_view, 42
- get\_raw\_views, 43
- get\_recording, 44
- get\_sample\_recording, 45

## \* datasets

- NIR\_ABh\_Puriya\_Annotation, 54
- NIR\_ABh\_Puriya\_Annotation\_Influence, 54
- NIR\_ABh\_Puriya\_Central\_Feature\_Sitar, 55
- NIR\_ABh\_Puriya\_Central\_Pose\_Sitar, 56
- NIR\_ABh\_Puriya\_Central\_Pose\_Tabla, 57
- NIR\_ABh\_Puriya\_Metre\_DrutTeental, 58
- NIR\_ABh\_Puriya\_Metre\_VilambitTeental,

59

- NIR\_ABh\_Puriya\_Onsets\_Selected\_DrutTeental, 59

- NIR\_ABh\_Puriya\_Onsets\_Selected\_VilambitTeental, 60

- NIR\_ABh\_Puriya\_OptFlow\_Central\_Sitar, 61

## \* splicing functions

- clip\_splice, 23
- get\_spliced\_view, 45
- is\_splice\_overlapping, 47
- merge\_splice, 49
- splice\_time, 81
- splice\_time.Duration, 82
- splice\_time.list, 83
- splice\_time.Metre, 83
- splice\_time.OnsetsDifference, 85
- splice\_time.View, 86
- split.SplicedView, 87

## \* statistical and analysis functions

- apply\_column\_spliceview, 7
- apply\_segment\_spliceview, 10
- ave\_cross\_power\_over\_splices, 16
- ave\_cross\_power\_spliceview, 17
- ave\_power\_over\_splices, 18
- ave\_power\_spliceview, 20
- calculate\_ave\_cross\_power1, 21
- calculate\_ave\_power1, 22
- compare\_ave\_cross\_power1, 24
- compare\_ave\_power1, 25
- compare\_avg\_cross\_power2, 26
- compare\_avg\_power2, 28
- difference\_onsets, 29
- pull\_segment\_spliceview, 76
- sample\_gap\_splice, 77
- sample\_offset\_splice, 78
- summary\_onsets, 93
- visualise\_sample\_splices, 95

## \* wavelet functions

- analyze\_coherency, 4
- analyze\_wavelet, 5
- get\_local\_max\_average\_power, 36
- plot\_average\_coherency, 66
- plot\_average\_power, 67
- plot\_cross\_spectrum, 68
- plot\_cwt\_energy, 69
- plot\_phase\_difference, 71
- plot\_power\_spectrum, 72
- plot\_roll\_resultant\_length, 73
- plot\_sel\_phases, 74
- plot\_wt\_energy, 75
  
- analyze\_coherency, 4, 6, 36, 66–69, 71–75
- analyze\_coherency(), 18
- analyze\_wavelet, 5, 5, 36, 66–69, 71–75
- analyze\_wavelet(), 20
- apply\_column\_spliceview, 7, 10, 17–20, 22, 23, 25–29, 76–78, 94, 96
- apply\_filter, 8
- apply\_filter\_sgolay, 9, 31–34, 36–38, 40–45
- apply\_segment\_spliceview, 7, 10, 17–20, 22, 23, 25–29, 76–78, 94, 96
- autolayer, 11
- autoplot, 13
- autoplot.GrangerTime, 14, 35, 47, 49, 51–53, 63, 71
- autoplot.SpectralDensityView, 15
- ave\_cross\_power\_over\_splices, 7, 10, 16, 18–20, 22, 23, 25–29, 76–78, 94, 96
- ave\_cross\_power\_spliceview, 7, 10, 17, 17, 19, 20, 22, 23, 25–29, 76–78, 94, 96
- ave\_power\_over\_splices, 7, 10, 17, 18, 18, 20, 22, 23, 25–29, 76–78, 94, 96
- ave\_power\_spliceview, 7, 10, 17–19, 20, 22, 23, 25–29, 76–78, 94, 96
  
- barplot(), 62, 65
  
- calculate\_ave\_cross\_power1, 7, 10, 17–20, 21, 23, 25–29, 76–78, 94, 96
- calculate\_ave\_power1, 7, 10, 17–20, 22, 22, 25–29, 76–78, 94, 96
- clip\_splice, 23, 46, 47, 49, 81–87
- compare\_ave\_cross\_power1, 7, 10, 17–20, 22, 23, 24, 26–29, 76–78, 94, 96
- compare\_ave\_power1, 7, 10, 17–20, 22, 23, 25, 25, 27–29, 76–78, 94, 96
  
- compare\_avg\_cross\_power2, 7, 10, 17–20, 22, 23, 25, 26, 26, 28, 29, 76–78, 94, 96
- compare\_avg\_power2, 7, 10, 17–20, 22, 23, 25–27, 28, 29, 76–78, 94, 96
  
- difference\_onsets, 7, 10, 17–20, 22, 23, 25–28, 29, 76–78, 94, 96
- distribution\_dp, 30
  
- get\_data\_points, 9, 31, 32–34, 36–38, 40–45
- get\_duration\_annotation\_data, 9, 31, 31, 33, 34, 36–38, 40–45
- get\_feature\_data, 9, 31, 32, 32, 34, 36–38, 40–45
- get\_filtered\_views, 9, 31–33, 33, 36–38, 40–45
- get\_granger\_interactions, 15, 34, 47, 49, 51–53, 63, 71
- get\_joined\_view, 9, 31–34, 35, 37, 38, 40–45
- get\_local\_max\_average\_power, 5, 6, 36, 66–69, 71–75
- get\_metre\_data, 9, 31–34, 36, 37, 38, 40–45
- get\_onsets\_selected\_data, 9, 31–34, 36, 37, 38, 40–45
- get\_osf\_recordings, 39
- get\_processed\_view, 9, 31–34, 36–38, 39, 41–45
- get\_processed\_views, 9, 31–34, 36–38, 40, 40, 42–45
- get\_raw\_optflow\_view, 9, 31–34, 36–38, 40, 41, 41, 43–45
- get\_raw\_view, 9, 31–34, 36–38, 40–42, 42, 43–45
- get\_raw\_views, 9, 31–34, 36–38, 40–43, 43, 44, 45
- get\_recording, 9, 31–34, 36–38, 40–43, 44, 45
- get\_sample\_recording, 9, 31–34, 36–38, 40–44, 45
- get\_spliced\_view, 23, 45, 47, 49, 81–87
- ggplot2::geom\_point(), 30, 50, 94
- granger\_test, 15, 35, 46, 49, 51–53, 63, 71
  
- is\_splice\_overlapping, 23, 46, 47, 49, 81–87
  
- list\_osf\_recordings, 48

- lmtest::grangertest(), 51–53
- lmtest::waldtest(), 51–53
- make.unique(), 82, 85
- map\_to\_granger\_test, 15, 35, 47, 48, 51–53, 63, 71
- merge\_splice, 23, 46, 47, 49, 81–87
- motion\_gram, 50
- ms\_condgrangertest, 15, 35, 47, 49, 51, 52, 53, 63, 71
- ms\_grangertest1, 15, 35, 47, 49, 51, 52, 53, 63, 71
- ms\_grangertest2, 15, 35, 47, 49, 51, 52, 53, 63, 71
- NIR\_ABh\_Puriya\_Annotation, 54
- NIR\_ABh\_Puriya\_Annotation\_Influence, 54
- NIR\_ABh\_Puriya\_Central\_Feature\_Sitar, 55
- NIR\_ABh\_Puriya\_Central\_Pose\_Sitar, 56
- NIR\_ABh\_Puriya\_Central\_Pose\_Tabla, 57
- NIR\_ABh\_Puriya\_Metre\_DrutTeental, 58
- NIR\_ABh\_Puriya\_Metre\_VilambitTeental, 59
- NIR\_ABh\_Puriya\_Onsets\_Selected\_DrutTeental, 59
- NIR\_ABh\_Puriya\_Onsets\_Selected\_VilambitTeental, 60
- NIR\_ABh\_Puriya\_OptFlow\_Central\_Sitar, 61
- open\_movementsync\_data, 62
- par(), 63
- plot.Duration, 62
- plot.GrangerInteraction, 15, 35, 47, 49, 51–53, 63, 71
- plot.igraph(), 63
- plot.Metre, 64
- plot.OnsetsSelected, 65
- plot.View, 65
- plot.zoo(), 66
- plot\_average\_coherency, 5, 6, 36, 66, 67–69, 71–75
- plot\_average\_power, 5, 6, 36, 66, 67, 68, 69, 71–75
- plot\_coherence (plot\_cross\_spectrum), 68
- plot\_cross\_spectrum, 5, 6, 36, 66, 67, 68, 69, 71–75
- plot\_cwt\_energy, 5, 6, 36, 66–68, 69, 71–75
- plot\_history\_xy, 69
- plot\_influence\_diagram, 15, 35, 47, 49, 51–53, 63, 70
- plot\_phase\_difference, 5, 6, 36, 66–69, 71, 72–75
- plot\_power\_spectrum, 5, 6, 36, 66–69, 71, 72, 73–75
- plot\_roll\_resultant\_length, 5, 6, 36, 66–69, 71, 72, 73, 74, 75
- plot\_sel\_phases, 5, 6, 36, 66–69, 71–73, 74, 75
- plot\_wt\_energy, 5, 6, 36, 66–69, 71–74, 75
- pull\_segment\_spliceview, 7, 10, 17–20, 22, 23, 25–29, 76, 77, 78, 94, 96
- sample\_gap\_splice, 7, 10, 17–20, 22, 23, 25–29, 76, 77, 78, 94, 96
- sample\_offset\_splice, 7, 10, 17–20, 22, 23, 25–29, 76, 77, 78, 94, 96
- sample\_time\_spliced\_views, 79
- sapply(), 7
- sapply\_column\_spliceview (apply\_column\_spliceview), 7
- signal::specgram(), 80
- specgram\_plot, 80
- spectral\_density, 80
- splice\_time, 23, 46, 47, 49, 81, 82–87
- splice\_time.Duration, 23, 46, 47, 49, 81, 82, 83–87
- splice\_time.list, 23, 46, 47, 49, 81, 82, 83, 84–87
- splice\_time.Metre, 23, 46, 47, 49, 81–83, 83, 85–87
- splice\_time.OnsetsDifference, 23, 46, 47, 49, 81–84, 85, 86, 87
- splice\_time.View, 23, 46, 47, 49, 81–85, 86, 87
- split.SplicedView, 23, 46, 47, 49, 81–86, 87
- stats::spectrum(), 81
- subset.View, 88
- summary.analyze.wavelet, 88
- summary.Duration, 89
- summary.Metre, 90
- summary.OnsetsSelected, 90
- summary.Recording, 91
- summary.sel.phases, 92
- summary.View, 92

summary\_onsets, [7](#), [10](#), [17–20](#), [22](#), [23](#), [25–29](#),  
[76–78](#), [93](#), [96](#)

velocity\_dp, [94](#)

visualise\_sample\_splices, [7](#), [10](#), [17–20](#),  
[22](#), [23](#), [25–29](#), [76–78](#), [94](#), [95](#)

WaveletComp::analyze.coherency(), [5](#)

WaveletComp::analyze.wavelet(), [6](#)

WaveletComp::wc.avg(), [66](#)

WaveletComp::wc.image(), [68](#)

WaveletComp::wc.phasediff.image(), [71](#)

WaveletComp::wc.sel.phases(), [74](#)

WaveletComp::wt.avg(), [67](#)

WaveletComp::wt.image(), [72](#)

xlim\_duration, [96](#)

zoo::plot.zoo(), [13](#)