Package 'tweenr'

September 6, 2022

Type Package

Title Interpolate Data for Smooth Animations

Version 2.0.2

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description In order to create smooth animation between states of data, tweening is necessary. This package provides a range of functions for creating tweened data that can be used as basis for animation. Furthermore it adds a number of vectorized interpolaters for common R data types such as numeric, date and colour.

URL https://github.com/thomasp85/tweenr

BugReports https://github.com/thomasp85/tweenr/issues

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.2.0)

Imports farver, magrittr, rlang, vctrs

LinkingTo cpp11 (>= 0.4.2)

RoxygenNote 7.2.1

Suggests testthat, covr

SystemRequirements C++11

NeedsCompilation yes

Author Thomas Lin Pedersen [aut, cre]
(<https://orcid.org/0000-0002-5147-4711>)

Repository CRAN

Date/Publication 2022-09-06 08:00:02 UTC

R topics documented:

`tweenr-package` *tweenr: Interpolate Data for Smooth Animations*

Description

In order to create smooth animation between states of data, tweening is necessary. This package provides a range of functions for creating tweened data that can be used as basis for animation. Furthermore it adds a number of vectorized interpolaters for common R data types such as numeric, date and colour.

Details

tweenr is a small collection of functions to help you in creating intermediary representations of your data, i.e. interpolating states of data. As such it's a great match for packages such as animate and gganimate, since it can work directly with data.frames of data, but it also provide fast and efficient interpolaters for numeric, date, datetime and colour that are vectorized and thus more efficient to use than the build in interpolation functions (mainly `stats::approx()` and `grDevices::colorRamp()`).

The main functions for data.frames are `tween_states()`, `tween_elements()` and `tween_appear()`, while the standard interpolaters can be found at `tween()`

Author(s)

Maintainer: Thomas Lin Pedersen <thomasp85@gmail.com> (ORCID)

See Also

Useful links:

- https://github.com/thomasp85/tweenr
- Report bugs at https://github.com/thomasp85/tweenr/issues

`display_ease` *Display an easing function*

Description

This simple helper lets you explore how the different easing functions govern the interpolation of data.

Usage

```
display_ease(ease)
```

Arguments

`ease`	The name of the easing function to display (see details)

Details

How transitions proceed between states are defined by an easing function. The easing function converts the parameterized progression from one state to the next to a new number between 0 and 1. `linear` easing is equivalent to an identity function that returns the input unchanged. In addition there are a range of additional easers available, each with three modifiers.

Easing modifiers:

-in The easing function is applied as-is

-out The easing function is applied in reverse

-in-out The first half of the transition it is applied as-is, while in the last half it is reversed

Easing functions

quadratic Models a power-of-2 function

cubic Models a power-of-3 function

quartic Models a power-of-4 function

quintic Models a power-of-5 function

sine Models a sine function

circular Models a pi/2 circle arc

exponential Models an exponential function

elastic Models an elastic release of energy

back Models a pullback and relase

bounce Models the bouncing of a ball

In addition to this function a good animated explanation can be found here.

Value

This function is called for its side effects

Examples

```
# The default - identity
display_ease('linear')

# A more fancy easer
display_ease('elastic-in')
```

---

gen_along *Generator for tweening along a variable*

Description

This is a generator version of `tween_along()`. It returns a generator that can be used with `get_frame()` and `get_raw_frames()` to extract frames for a specific time point scaled between 0 and 1.

Usage

```
gen_along(
  .data,
  ease,
  along,
  id = NULL,
  range = NULL,
  history = TRUE,
  keep_last = FALSE
)
```

Arguments

`.data`	A data.frame with components at different stages
`ease`	The easing function to use. Either a single string or one for each column in the data set.
`along`	The "time" point for each row
`id`	An unquoted expression giving the component id for each row. Will be evaluated in the context of `.data` so can refer to a column from that
`range`	The range of time points to include in the tween. If `NULL` it will use the range of `time`
`history`	Should earlier datapoints be kept in subsequent frames
`keep_last`	Should the last point of each id be kept beyond its time

Value

An `along_generator` object

See Also

Other Other generators: gen_at(), gen_components(), gen_events(), gen_keyframe()

Examples

```
# Default behaviour
gen <- gen_along(airquality, ease = "linear", along = Day, id = Month)
get_frame(gen, 0.22)

# Overwrite keep_last or history in get_frame
get_frame(gen, 0.67, history = FALSE)
```

---

gen_at *Generator for interpolating between two data frames*

Description

This is a generator version of tween_at() with the additional functionality of supporting enter and exit functions. It returns a generator that can be used with get_frame() and get_raw_frames() to extract frames for a specific time point scaled between 0 and 1.

Usage

```
gen_at(from, to, ease, id = NULL, enter = NULL, exit = NULL)
```

Arguments

from, to	A data.frame or vector of the same type. If either is of length/nrow 1 it will get repeated to match the length of the other
ease	A character vector giving valid easing functions. Recycled to match the ncol of from
id	The column to match observations on. If NULL observations will be matched by position. See the *Match, Enter, and Exit* section for more information.
enter, exit	functions that calculate a start state for new observations that appear in to or an end state for observations that are not present in to. If NULL the new/old observations will not be part of the tween. The function gets a data.frame with either the start state of the exiting observations, or the end state of the entering observations and must return a modified version of that data.frame. See the *Match, Enter, and Exit* section for more information.

Value

A keyframe_generator object

See Also

Other Other generators: gen_along(), gen_components(), gen_events(), gen_keyframe()

Examples

```
gen <- gen_at(mtcars[1:6, ], mtcars[6:1, ], 'cubic-in-out')

get_frame(gen, 0.3)
```

---

gen_components *Generator for tweening components separately from each other*

Description

This is a generator versions of `tween_components()`. It returns a generator that can be used with `get_frame()` and `get_raw_frames()` to extract frames for a specific time point scaled between 0 and 1.

Usage

```
gen_components(
  .data,
  ease,
  nframes,
  time,
  id = NULL,
  range = NULL,
  enter = NULL,
  exit = NULL,
  enter_length = 0,
  exit_length = 0
)
```

Arguments

`.data`	A data.frame with components at different stages
`ease`	The easing function to use. Either a single string or one for each column in the data set.
`nframes`	The number of frames to calculate for the tween
`time`	An unquoted expression giving the timepoint for the different stages of the components. Will be evaluated in the context of `.data` so can refer to a column from that
`id`	An unquoted expression giving the component id for each row. Will be evaluated in the context of `.data` so can refer to a column from that
`range`	The range of time points to include in the tween. If `NULL` it will use the range of `time`

`enter, exit` functions that calculate a start state for new observations that appear in `to` or an end state for observations that are not present in `to`. If `NULL` the new/old observations will not be part of the tween. The function gets a data.frame with either the start state of the exiting observations, or the end state of the entering observations and must return a modified version of that data.frame. See the *Match, Enter, and Exit* section for more information.

`enter_length, exit_length` The lenght of the opening and closing transitions if `enter` and/or `exit` is given. Measured in the same units as `time`

Value

A `component_generator` object

See Also

Other Other generators: `gen_along()`, `gen_at()`, `gen_events()`, `gen_keyframe()`

Examples

```
from_zero <- function(x) {x$x <- 0; x}

data <- data.frame(
  x = c(1, 2, 2, 1, 2, 2),
  y = c(1, 2, 2, 2, 1, 1),
  time = c(1, 4, 8, 4, 8, 10),
  id = c(1, 1, 1, 2, 2, 2)
)

gen <- gen_components(data, 'cubic-in-out', time = time, id = id,
                      enter = from_zero, enter_length = 4)

get_frame(gen, 0.3)
```

---

gen_events *Generator for tweening the appearance of elements*

Description

This is a generator version of `tween_events()`. It returns a generator that can be used with `get_frame()` and `get_raw_frames()` to extract frames for a specific time point scaled between 0 and 1.

Usage

```
gen_events(
  .data,
  ease,
```

```
  start,
  end = NULL,
  range = NULL,
  enter = NULL,
  exit = NULL,
  enter_length = 0,
  exit_length = 0
)
```

Arguments

`.data`	A data.frame with components at different stages
`ease`	The easing function to use. Either a single string or one for each column in the data set.
`start, end`	The start (and potential end) of the event encoded in the row, as unquoted expressions. Will be evaluated in the context of `.data` so can refer to columns in it. If `end = NULL` the event will be without extend and only visible in a single frame, unless `enter` and/or `exit` is given.
`range`	The range of time points to include in the tween. If `NULL` it will use the range of `time`
`enter, exit`	functions that calculate a start state for new observations that appear in `to` or an end state for observations that are not present in `to`. If `NULL` the new/old observations will not be part of the tween. The function gets a data.frame with either the start state of the exiting observations, or the end state of the entering observations and must return a modified version of that data.frame. See the *Match, Enter, and Exit* section for more information.
`enter_length, exit_length`	The lenght of the opening and closing transitions if `enter` and/or `exit` is given. Measured in the same units as `time`

Value

A `component_generator` object

See Also

Other Other generators: `gen_along()`, `gen_at()`, `gen_components()`, `gen_keyframe()`

Examples

```
d <- data.frame(
  x = runif(20),
  y = runif(20),
  time = runif(20),
  duration = runif(20, max = 0.1)
)
from_left <- function(x) {
  x$x <- -0.5
  x
```

```
}
to_right <- function(x) {
  x$x <- 1.5
  x
}

gen <- gen_events(d, 'cubic-in-out', start = time, end = time + duration,
                  enter = from_left, exit = to_right, enter_length = 0.1,
                  exit_length = 0.05)

get_frame(gen, 0.65)
```

---

gen_keyframe *Generator for keyframe based tweening*

Description

This is a generator version of `tween_state()` and its utility functions. It returns a generator that can be used with `get_frame()` and `get_raw_frames()` to extract frames for a specific time point scaled between 0 and 1.

Usage

```
gen_keyframe(keyframe = NULL, pause = 0)

add_pause(.data, pause = 0)

add_keyframe(
  .data,
  keyframe,
  ease,
  length,
  id = NULL,
  enter = NULL,
  exit = NULL
)
```

Arguments

keyframe	A data frame to use as a keyframe state
pause	The length of the pause at the current keyframe
.data	A data.frame to start from. If `.data` is the result of a prior tween, only the last frame will be used for the tween. The new tween will then be added to the prior tween
ease	The easing function to use. Either a single string or one for each column in the data set.

length	The length of the transition
id	The column to match observations on. If NULL observations will be matched by position. See the *Match, Enter, and Exit* section for more information.
enter, exit	functions that calculate a start state for new observations that appear in to or an end state for observations that are not present in to. If NULL the new/old observations will not be part of the tween. The function gets a data.frame with either the start state of the exiting observations, or the end state of the entering observations and must return a modified version of that data.frame. See the *Match, Enter, and Exit* section for more information.

Value

A keyframe_generator object

See Also

Other Other generators: gen_along(), gen_at(), gen_components(), gen_events()

Examples

```
df1 <- data.frame(
  country = c('Denmark', 'Sweden', 'Norway'),
  population = c(5e6, 10e6, 3.5e6)
)
df2 <- data.frame(
  country = c('Denmark', 'Sweden', 'Norway', 'Finland'),
  population = c(6e6, 10.5e6, 4e6, 3e6)
)
df3 <- data.frame(
  country = c('Denmark', 'Norway'),
  population = c(10e6, 6e6)
)
to_zero <- function(x) {
  x$population <- 0
  x
}
gen <- gen_keyframe(df1, 10) %>%
  add_keyframe(df2, 'cubic-in-out', 35, id = country, enter = to_zero) %>%
  add_pause(10) %>%
  add_keyframe(df3, 'cubic-in-out', 35, id = country, enter = to_zero,
               exit = to_zero) %>%
  add_pause(10)

get_frame(gen, 0.25)
```

get_frame *Extract a frame from a generator*

Description

Using the generators in tweenr you can avoid calculating all needed frames up front, which can be prohibitive in memory. With a generator you can use `get_frame()` to extract any frame at a fractional location between 0 and 1 one by one as you need them. You can further get all raw data before and/or after a given point in time using `get_raw_frames()`.

Usage

```
get_frame(generator, at, ...)

get_raw_frames(generator, at, before = 0, after = 0, ...)
```

Arguments

`generator`	A `frame_generator` object
`at`	A scalar numeric between 0 and 1
`...`	Arguments passed on to methods
`before, after`	Scalar numerics that define the time before and after `at` to search for raw data

Examples

```
data <- data.frame(
  x = c(1, 2, 2, 1, 2, 2),
  y = c(1, 2, 2, 2, 1, 1),
  time = c(1, 4, 8, 4, 8, 10),
  id = c(1, 1, 1, 2, 2, 2)
)

gen <- gen_components(data, 'cubic-in-out', time = time, id = id)

get_frame(gen, 0.3)

get_raw_frames(gen, 0.5, before = 0.5, after = 0.2)
```

tween *Create simple tweens*

Description

This set of functions can be used to interpolate between single data types, i.e. data not part of data.frames but stored in vectors. All functions come in two flavours: the standard and a *_t version. The standard reads the data as a list of states, each tween matched element-wise from state to state. The *_t version uses the transposed representation where each element is a vector of states. The standard approach can be used when each tween has the same number of states and you want to control the number of point in each state transition. The latter is useful when each tween consists of different numbers of states and/or you want to specify the total number of points for each tween.

Usage

```
tween(data, n, ease = "linear")

tween_t(data, n, ease = "linear")

tween_colour(data, n, ease = "linear")

tween_color(data, n, ease = "linear")

tween_colour_t(data, n, ease = "linear")

tween_color_t(data, n, ease = "linear")

tween_constant(data, n, ease = "linear")

tween_constant_t(data, n, ease = "linear")

tween_date(data, n, ease = "linear")

tween_date_t(data, n, ease = "linear")

tween_datetime(data, n, ease = "linear")

tween_datetime_t(data, n, ease = "linear")

tween_numeric(data, n, ease = "linear")

tween_numeric_t(data, n, ease = "linear")
```

Arguments

data	A list of vectors or a single vector. In the standard functions each element in the list must be of equal length; for the *_t functions lengths can differ. If a

single vector is used it will be eqivalent to using `as.list(data)` for the standard functions and `list(data)` for the *_t functions.

`n` The number of elements per transition or tween. See details

`ease` The easing function to use for each transition or tween. See details. Defaults to `'linear'`

Details

`tween` and `tween_t` are wrappers around the other functions that tries to guess the type of input data and choose the appropriate tween function. Unless you have data that could be understood as a colour but is in fact a character vector it should be safe to use these wrappers. It is probably safer and more verbose to use the explicit functions within package code as they circumvent the type inference and checks whether the input data matches the tween function.

`tween_numeric` will provide a linear interpolation between the points based on the sequence returned by the easing function. `tween_date` and `tween_datetime` converts to numeric, produces the tweening, and converts back again. `tween_colour` converts colours into Lab and does the interpolation there, converting back to sRGB after the tweening is done. `tween_constant` is a catchall that converts the input into character and interpolates by switching between states halfway through the transition.

The meaning of the `n` and `ease` arguments differs somewhat between the standard and *_t versions of the functions. In the standard function `n` and `ease` refers to the length and easing function of each transition, being recycled if necessary to `length(data) - 1`. In the *_t functions `n` and `ease` refers to the total length of each tween and the easing function to be applied to all transition for each tween. The will both be recycled to `length(data)`.

Value

A list with an element for each tween. That means that the length of the return is equal to the length of the elements in `data` for the standard functions and equal to the length of `data` for the *_t functions.

Difference Between `tween_numeric` and `approx()`

`tween_numeric` (and `tween_numeric_t`) is superficially equivalent to `stats::approx()`, but there are differences. `stats::approx()` will create evenly spaced points, at the expense of not including the actual points in the input, while the reverse is true for `tween_numeric`. Apart from that `tween_numeric` of course supports easing functions and is vectorized.

Examples

```
tween_numeric(list(1:3, 10:8, c(20, 60, 30)), 10)

tween_colour_t(list(colours()[1:4], colours()[1:2], colours()[25:100]), 100)
```

tween_along *Interpolate data along a given dimension*

Description

This tween takes groups of rows along with the time for each row and calculates the exact value at each at each frame. Further it allows for keeping the subsequent raw data from previous frame as well as letting the final row linger beyond its time. It especially useful for data that should be visualised as lines that are drawn along the x-axis, but can of course also be used for other dimensions as well (even dimensions not corresponding to any axis).

Usage

```
tween_along(
  .data,
  ease,
  nframes,
  along,
  id = NULL,
  range = NULL,
  history = TRUE,
  keep_last = FALSE
)
```

Arguments

.data	A data.frame with components at different stages
ease	The easing function to use. Either a single string or one for each column in the data set.
nframes	The number of frames to calculate for the tween
along	The "time" point for each row
id	An unquoted expression giving the component id for each row. Will be evaluated in the context of .data so can refer to a column from that
range	The range of time points to include in the tween. If NULL it will use the range of time
history	Should earlier datapoints be kept in subsequent frames
keep_last	Should the last point of each id be kept beyond its time

Value

A data.frame with the same columns as .data along with .id giving the component id, .phase giving the state of each component in each frame, and .frame giving the frame membership of each row.

See Also

Other data.frame tween: tween_appear(), tween_components(), tween_elements(), tween_events(), tween_states()

`tween_appear` *Tween a data.frame of appearances*

Description

This function is intended for use when you have a data.frame of events at different time points. This could be the appearance of an observation for example. This function replicates your data `nframes` times and calculates the duration of each frame. At each frame each row is assigned an age based on the progression of frames and the entry point of in time for that row. A negative age means that the row has not appeared yet.

Usage

```
tween_appear(data, time, timerange, nframes)
```

Arguments

`data`	A data.frame to tween
`time`	The name of the column that holds the time dimension. This does not need to hold time data in the strictest sence - any numerical type will do
`timerange`	The range of time to create the tween for. If missing it will defaults to the range of the time column
`nframes`	The number of frames to create for the tween. If missing it will create a frame for each full unit in `timerange` (e.g. `timerange = c(1, 10)` will give `nframes = 10`)

Value

A data.frame as `data` but repeated `nframes` times and with the additional columns `.age` and `.frame`

See Also

Other data.frame tween: tween_along(), tween_components(), tween_elements(), tween_events(), tween_states()

Examples

```
data <- data.frame(
  x = rnorm(100),
  y = rnorm(100),
  time = sample(50, 100, replace = TRUE)
)

data <- tween_appear(data, 'time', nframes = 200)
```

tween_at *Get a specific position between two states*

Description

This tween allows you to query a specific postion between two states rather than generate evenly spaced states. It can work with either data.frames or single vectors and each row/element can have its own position and easing.

Usage

```
tween_at(from, to, at, ease)
```

Arguments

from, to	A data.frame or vector of the same type. If either is of length/nrow 1 it will get repeated to match the length of the other
at	A numeric between 0 and 1 recycled to match the nrow/length of from
ease	A character vector giving valid easing functions. Recycled to match the ncol of from

Value

If from/to is a data.frame then a data.frame with the same columns. If from/to is a vector then a vector.

Examples

```
tween_at(mtcars[1:6, ], mtcars[6:1, ], runif(6), 'cubic-in-out')
```

tween_components *Interpolate individual component*

Description

This function is much like tween_elements() but with a slightly different syntax and support for many of the newer features such as enter/exits and tween phase identification. Furthermore it uses tidy evaluation for time and id, making it easier to change these on the fly. The biggest change in terms of functionality compared to tween_elements() is that the easing function is now given per column and not per row. If different easing functions are needed for each transition then tween_elements() is needed.

Usage

```
tween_components(
  .data,
  ease,
  nframes,
  time,
  id = NULL,
  range = NULL,
  enter = NULL,
  exit = NULL,
  enter_length = 0,
  exit_length = 0
)
```

Arguments

`.data`	A data.frame with components at different stages
`ease`	The easing function to use. Either a single string or one for each column in the data set.
`nframes`	The number of frames to calculate for the tween
`time`	An unquoted expression giving the timepoint for the different stages of the components. Will be evaluated in the context of `.data` so can refer to a column from that
`id`	An unquoted expression giving the component id for each row. Will be evaluated in the context of `.data` so can refer to a column from that
`range`	The range of time points to include in the tween. If `NULL` it will use the range of `time`
`enter, exit`	functions that calculate a start state for new observations that appear in `to` or an end state for observations that are not present in `to`. If `NULL` the new/old observations will not be part of the tween. The function gets a data.frame with either the start state of the exiting observations, or the end state of the entering observations and must return a modified version of that data.frame. See the *Match, Enter, and Exit* section for more information.
`enter_length, exit_length`	The lenght of the opening and closing transitions if `enter` and/or `exit` is given. Measured in the same units as `time`

Value

A data.frame with the same columns as `.data` along with `.id` giving the component id, `.phase` giving the state of each component in each frame, and `.frame` giving the frame membership of each row.

See Also

Other data.frame tween: `tween_along()`, `tween_appear()`, `tween_elements()`, `tween_events()`, `tween_states()`

Examples

```
from_zero <- function(x) {x$x <- 0; x}

data <- data.frame(
  x = c(1, 2, 2, 1, 2, 2),
  y = c(1, 2, 2, 2, 1, 1),
  time = c(1, 4, 10, 4, 8, 10),
  id = c(1, 1, 1, 2, 2, 2)
)

data <- tween_components(data, 'cubic-in-out', nframes = 100, time = time,
                         id = id, enter = from_zero, enter_length = 4)
```

---

tween_elements *Create frames based on individual element states*

Description

This function creates tweens for each observation individually, in cases where the data doesn't pass through collective states but consists of fully independent transitions. Each observation is identified by an id and each state must have a time associated with it.

Usage

```
tween_elements(data, time, group, ease, timerange, nframes)
```

Arguments

data	A data.frame consisting at least of a column giving the observation id, a column giving timepoints for each state and a column giving the easing to apply when transitioning away from the state.
time	The name of the column holding timepoints
group	The name of the column holding the observation id
ease	The name of the column holding the easing function name
timerange	The range of time to span. If missing it will default to `range(data[[time]])`
nframes	The number of frames to generate. If missing it will default to `ceiling(diff(timerange) + 1)` (At least one frame for each individual timepoint)

Value

A data.frame with the same columns as `data` except for the group and ease columns, but replicated `nframes` times. Two additional columns called `.frame` and `.group` will be added giving the frame number and observation id for each row.

See Also

Other data.frame tween: tween_along(), tween_appear(), tween_components(), tween_events(), tween_states()

Examples

```
data <- data.frame(
  x = c(1, 2, 2, 1, 2, 2),
  y = c(1, 2, 2, 2, 1, 1),
  time = c(1, 4, 10, 4, 8, 10),
  group = c(1, 1, 1, 2, 2, 2),
  ease = rep('cubic-in-out', 6)
)

data <- tween_elements(data, 'time', 'group', 'ease', nframes = 100)
```

---

tween_events *Transition in and out of events*

Description

This tweening function is a more powerful version of tween_appear(), with support for newer features such as enter/exits and tween phase identification. The tweener treats each row in the data as unique events in time, and creates frames with the correct events present at any given time.

Usage

```
tween_events(
  .data,
  ease,
  nframes,
  start,
  end = NULL,
  range = NULL,
  enter = NULL,
  exit = NULL,
  enter_length = 0,
  exit_length = 0
)
```

Arguments

.data	A data.frame with components at different stages
ease	The easing function to use. Either a single string or one for each column in the data set.
nframes	The number of frames to calculate for the tween

start, end
: The start (and potential end) of the event encoded in the row, as unquoted expressions. Will be evaluated in the context of `.data` so can refer to columns in it. If `end = NULL` the event will be without extend and only visible in a single frame, unless `enter` and/or `exit` is given.

range
: The range of time points to include in the tween. If `NULL` it will use the range of `time`

enter, exit
: functions that calculate a start state for new observations that appear in `to` or an end state for observations that are not present in `to`. If `NULL` the new/old observations will not be part of the tween. The function gets a data.frame with either the start state of the exiting observations, or the end state of the entering observations and must return a modified version of that data.frame. See the *Match, Enter, and Exit* section for more information.

enter_length, exit_length
: The lenght of the opening and closing transitions if `enter` and/or `exit` is given. Measured in the same units as `time`

Value

A data.frame with the same columns as `.data` along with `.id` giving the component id, `.phase` giving the state of each component in each frame, and `.frame` giving the frame membership of each row.

See Also

Other data.frame tween: `tween_along()`, `tween_appear()`, `tween_components()`, `tween_elements()`, `tween_states()`

Examples

```
d <- data.frame(
  x = runif(20),
  y = runif(20),
  time = runif(20),
  duration = runif(20, max = 0.1)
)
from_left <- function(x) {
  x$x <- -0.5
  x
}
to_right <- function(x) {
  x$x <- 1.5
  x
}

tween_events(d, 'cubic-in-out', 50, start = time, end = time + duration,
             enter = from_left, exit = to_right, enter_length = 0.1,
             exit_length = 0.05)
```

tween_fill *Fill out missing values by interpolation*

Description

This tween fills out NA elements (or NULL elements if data is a list) by interpolating between the prior and next non-missing values.

Usage

```
tween_fill(data, ease)
```

Arguments

data	A data.frame or vector.
ease	A character vector giving valid easing functions. Recycled to match the ncol of data

Value

If data is a data.frame then a data.frame with the same columns. If data is a vector then a vector.

Examples

```
# Single vector
tween_fill(c(1, NA, NA, NA, NA, NA, 2, 6, NA, NA, NA, -2), 'cubic-in-out')

# Data frame
tween_fill(mtcars[c(1, NA, NA, NA, NA, 4, NA, NA, NA, 10), ], 'cubic-in')
```

tween_state *Compose tweening between states*

Description

The tween_state() is a counterpart to tween_states() that is aimed at letting you gradually build up a scene by composing state changes one by one. This setup lets you take more control over each state change and allows you to work with datasets with uneven number of rows, flexibly specifying what should happen with entering and exiting data. keep_state() is a simple helper for letting you pause at a state. open_state() is a shortcut from tweening from an empty dataset with a given enter() function while close_state() is the same but will instead tween into an empty dataset with a given exit() function.

Usage

```
tween_state(.data, to, ease, nframes, id = NULL, enter = NULL, exit = NULL)

keep_state(.data, nframes)

open_state(.data, ease, nframes, enter)

close_state(.data, ease, nframes, exit)
```

Arguments

`.data`	A data.frame to start from. If `.data` is the result of a prior tween, only the last frame will be used for the tween. The new tween will then be added to the prior tween
`to`	A data.frame to end at. It must contain the same columns as .data (exluding `.frame`)
`ease`	The easing function to use. Either a single string or one for each column in the data set.
`nframes`	The number of frames to calculate for the tween
`id`	The column to match observations on. If `NULL` observations will be matched by position. See the *Match, Enter, and Exit* section for more information.
`enter, exit`	functions that calculate a start state for new observations that appear in `to` or an end state for observations that are not present in `to`. If `NULL` the new/old observations will not be part of the tween. The function gets a data.frame with either the start state of the exiting observations, or the end state of the entering observations and must return a modified version of that data.frame. See the *Match, Enter, and Exit* section for more information.

Value

A data.frame containing all the intermediary states in the tween, each state will be enumerated by the `.frame` column

Match, Enter, and Exit

When there are discrepancies between the two states to tweeen between you need a way to resolve the discrepancy before calculating the intermediary states. With discrepancies we mean that some data points are present in the start state and not in the end state, and/or some are present in the end state but not in the start state. A simple example is that the start state contains 100 rows and the end state contains 70. There are 30 missing rows that we need to do something about before we can calculate the tween.

Making pairs The first question to answer is "How do we know which observations are disappearing (*exiting*) and/or appearing (*entering*)?". This is done with the `id` argument which should give a column name to match rows between the two states on. If `id = NULL` the rows will be matched by position (in the above example the last 30 rows in the start state will be entering). The `id` column must only contain unique values in order to work.

Making up states Once the rows in each state has been paired you'll end up with three sets of data. One containing rows that is present in both the start and end state, one containing rows only present in the start state, and one only containing rows present in the end state. The first group is easy - here you just tween between each rows - but for the other two we'll need some state to start or end the tween with. This is really the purpose of the `enter` and `exit` functions. They take a data frame containing the subset of data that has not been matched and must return a new data frame giving the state that these rows must be tweened from/into. A simple example could be an `enter` function that sets the variable giving the opacity in the plot to 0 - this will make the new points fade into view during the transition.

Ignoring discrepancies The default values for `enter` and `exit` is `NULL`. This value indicate that non-matching rows should simply be ignored for the transition and simply appear in the last frame of the tween. This is the default.

Examples

```
data1 <- data.frame(
  x = 1:20,
  y = 0,
  colour = 'forestgreen',
  stringsAsFactors = FALSE
)
data2 <- data1
data2$x <- 20:1
data2$y <- 1

data <- data1 %>%
  tween_state(data2, 'linear', 50) %>%
  keep_state(20) %>%
  tween_state(data1, 'bounce-out', 50)

# Using enter and exit (made up numbers)
df1 <- data.frame(
  country = c('Denmark', 'Sweden', 'Norway'),
  population = c(5e6, 10e6, 3.5e6)
)
df2 <- data.frame(
  country = c('Denmark', 'Sweden', 'Norway', 'Finland'),
  population = c(6e6, 10.5e6, 4e6, 3e6)
)
df3 <- data.frame(
  country = c('Denmark', 'Norway'),
  population = c(10e6, 6e6)
)
to_zero <- function(x) {
  x$population <- 0
  x
}
pop_devel <- df1 %>%
  tween_state(df2, 'cubic-in-out', 50, id = country, enter = to_zero) %>%
  tween_state(df3, 'cubic-in-out', 50, id = country, enter = to_zero,
              exit = to_zero)
```

tween_states *Tween a list of data.frames representing states*

Description

This function is intended to create smooth transitions between states of data. States are defined as full data.frames or data.frames containing only the columns with change. Each state can have a defined period of pause, the transition length between each states can be defined as well as the easing function.

Usage

```
tween_states(data, tweenlength, statelength, ease, nframes)
```

Arguments

data	A list of data.frames. Each data.frame must contain the same number of rows, but only the first data.frame needs to contain all columns. Subsequent data.frames need only contain the columns that shows change.
tweenlength	The lengths of the transitions between each state.
statelength	The length of the pause at each state.
ease	The easing functions to use for the transitions. See details.
nframes	The number of frames to generate. The actual number of frames might end up being higher depending on the regularity of tweenlength and statelength.

Value

A data.frame with the same columns as the first data.frame in data, but replicated nframes times. An additional column called .frame will be added giving the frame number.

See Also

Other data.frame tween: tween_along(), tween_appear(), tween_components(), tween_elements(), tween_events()

Examples

```
data1 <- data.frame(
  x = 1:20,
  y = 0,
  colour = 'forestgreen',
  stringsAsFactors = FALSE
)
data2 <- data1
data2$x <- 20:1
```

```
data2$y <- 1

data <- tween_states(list(data1, data2), 3, 1, 'cubic-in-out', 100)
```

Index