Package ‘tweenr’

March 23, 2021

Type     Package
Title    Interpolate Data for Smooth Animations
Version  1.0.2
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
LazyData TRUE
Depends  R (>= 3.2.0)
Imports  Rcpp (>= 0.12.3), grDevices, farver, magrittr, rlang
LinkingTo Rcpp
RoxygenNote 7.1.1.9001
Suggests testthat, covr
NeedsCompilation yes
Author   Thomas Lin Pedersen [aut, cre]
          (<https://orcid.org/0000-0002-5147-4711>)
Repository CRAN
Date/Publication 2021-03-23 15:40:02 UTC

R topics documented:
tweenr-package .................................................. 2
display_ease ...................................................... 3
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](https://github.com/thomasp85/tweenr)
- Report bugs at [https://github.com/thomasp85/tweenr/issues](https://github.com/thomasp85/tweenr/issues)
display_ease

<table>
<thead>
<tr>
<th></th>
<th>Display an easing function</th>
</tr>
</thead>
</table>

**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 reliance
- **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')

---

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 equivalent 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
tween_appear

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 sense - 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(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

```r
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** 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.
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 length 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)
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

```r
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 ex-
pressions. 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 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.
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 The length of the opening and closing transitions if enter and/or exit is given.
  Measured in the same units as time
exit_length The length 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(data, ease)

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, 2, 6, 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.data</code></td>
<td>A data.frame to start from. If <code>.data</code> 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</td>
</tr>
<tr>
<td><code>to</code></td>
<td>A data.frame to end at. It must contain the same columns as <code>.data</code> (excluding <code>.frame</code>)</td>
</tr>
<tr>
<td><code>ease</code></td>
<td>The easing function to use. Either a single string or one for each column in the data set.</td>
</tr>
<tr>
<td><code>nframes</code></td>
<td>The number of frames to calculate for the tween</td>
</tr>
<tr>
<td><code>id</code></td>
<td>The column to match observations on. If <code>NULL</code> observations will be matched by position. See the <em>Match, Enter, and Exit</em> section for more information.</td>
</tr>
</tbody>
</table>
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 tween 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

- `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**

```r
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

* data.frame tween
  tween_along, 6
  tween_appear, 7
  tween_components, 9
  tween_elements, 10
  tween_events, 11
  tween_states, 16

close_state (tween_state), 14
display_ease, 3
grDevices::colorRamp(), 2
keep_state (tween_state), 14
open_state (tween_state), 14
stats::approx(), 2, 5
tween, 4
tween(), 2
tween_along, 6, 7, 10, 11, 13, 17
tween_appear, 7, 7, 10, 11, 13, 17
tween_appear(), 2, 11
tween_at, 8
tween_color (tween), 4
tween_color_t (tween), 4
tween_colour (tween), 4
tween_colour_t (tween), 4
tween_components, 7, 9, 11, 13, 17
tween_constant (tween), 4
tween_constant_t (tween), 4
tween_date (tween), 4
tween_date_t (tween), 4
tween_datetime (tween), 4
tween_datetime_t (tween), 4
tween_elements, 7, 10, 10, 13, 17
tween_elements(), 2, 9
tween_events, 7, 10, 11, 11, 17
tween_fill, 13
tween_n (tween), 4
tween_numeric (tween), 4
tween_numeric_t (tween), 4
tween_state, 14
tween_states, 7, 10, 11, 13, 16
tween_states(), 2
tween_t (tween), 4
tweenr (tweenr-package), 2
tweenr-package, 2