Package ‘anim.plots’

October 12, 2022

Type Package
Title Simple Animated Plots for R
Version 0.2.2
Author David Hugh-Jones <davidhughjones@gmail.com>
Maintainer David Hugh-Jones <davidhughjones@gmail.com>
Description Simple animated versions of basic R plots, using the 'animation' package. Includes animated versions of plot, barplot, persp, contour, filled.contour, hist, curve, points, lines, text, symbols, segments, and arrows.
License GPL-2
LazyData TRUE
URL https://github.com/hughjonesd/anim.plots
BugReports https://github.com/hughjonesd/anim.plots/issues
VignetteBuilder knitr
Imports animation
Suggests maps, knitr, mapdata, testthat, rmarkdown
RoxygenNote 7.1.1
NeedsCompilation no
Repository CRAN
Date/Publication 2021-04-30 15:50:02 UTC

R topics documented:

anim.plots-package .................................................. 2
anim.barplot .......................................................... 2
anim.contour ......................................................... 4
anim.curve .......................................................... 5
anim.hist ............................................................ 6
anim.plot ............................................................ 7
anim.save ........................................................... 11
anim.plots-package  

anim.plots: simple animated plots For R

Description

anim.plots provides simple animated versions of basic R plots, using the 'animation' package. It includes animated versions of plot, barplot, persp, contour, filled.contour, hist, curve, points, lines, text, symbols, segments, and arrows.

Details

For more information, run vignette('anim.plots-stub'), or check the vignette out on the web at https://hughjonesd.github.io/anim.plots/anim.plots.html.

Be aware that anim.plots is just a simple wrapper around Yihui Xie's "animation" package. You may want to consider more modern solutions such as gganimate.

anim.barplot  

Create an animated barplot.

Description

Create an animated barplot.

Usage

anim.barplot(...)

## Default S3 method:
anim.barplot(
  height,
  times = NULL,
  show = TRUE,
  speed = 1,
Arguments

height  a vector, matrix or array. If a vector it is divided up by times and barplot is called on each chunk. If a matrix, barplot is called on each column. If an array, barplot is called on each matrix of form height[,i].

times  a vector of times. If NULL and height is a matrix, the last dimension of height will be used

show, speed, use.times, window, window.process

see anim.plot

width, space, beside, names.arg, density, angle, col, border, horiz, xlim, ylim, xlab, ylab, main, sub, offset, legend.text, ...

Arguments passed to barplot.

Details

Arguments width, names.arg, density, angle, col, border and offset may be either vectors of length length(tbl) or matrices with one column for each unique value of times. Other arguments should be length 1 or vectors.

Examples

anim.barplot(1:100, times=rep(1:10, each=10), ylim=c(0,100))
## barplot with a matrix
ChickWeight$qw <- cut(ChickWeight$weight, 5)
tbl <- as.array(xtabs(~ qw + Diet + Time, data=ChickWeight))
ptbl <- prop.table(tbl, 2:3)
anim.barplot(ptbl, xlab="Diet", ylab="N", xlim=c(0,8), legend.text=paste(
anim.contour

Create an animated contour plot or perspective plot

Description

Create an animated contour plot or perspective plot of 3D data.

Usage

anim.contour(...)

anim.barplot(tbl, xlab="Diet", ylab="N", beside=TRUE, ylim=c(0,20),
             legend.text=paste("Quintile", 1:5), col=1:5)

Arguments

x, y, z, ... arguments passed to contour or persp

times, speed, use.times, window, window.process, show

see anim.plot for details.

fn underlying function to use.
Examples

tmp <- volcano
tmp[] <- 200 - ((row(tmp) - 43)^2 + (col(tmp) - 30)^2)/20
cplot <- array(NA, dim=c(87,61,20))
cplot[,1] <- tmp
cplot[,20] <- volcano
cplot <- apply(cplot, 1:2, function(x) seq(x[1], x[20], length.out=20))
cplot <- aperm(cplot, c(2,3,1))
anim.contour(z=cplot, times=1:20, speed=3, levels=80 + 1:12*10, lty=c(1,2,2))
anim.filled.contour(z=cplot, times=1:20, speed=3, levels=80 + 1:12*10,
                   color.palette=terrain.colors)
cplot2 <- apply(cplot, 1:2, function(x) seq(0, x[20], length.out=20))
cplot2 <- aperm(cplot2, c(2,3,1))
anim.persp(z=cplot2, times=1:20, xlab="", ylab="", zlab="Height",
           phi=45, theta=30, speed=5, border=NA, r=3, col="yellowgreen",
           shade=.5, box=FALSE)

Description

This function is the animated version of curve.

Usage

anim.curve(expr, x = NULL, from = 0, to = 1, n = 255, times, type = "l", ...)

Arguments

expr a function which takes two arguments, or an expression involving x and t.
x values of x at which the function will be evaluated in each frame. Alternatively,
you may specify from, to and n.
from, to endpoints of x
n number of values of x at which the function will be evaluated for each frame
times vector of values of t at which the function will be evaluated. Each unique value
creates a single animation frame.
type, ... parameters passed to anim.plot.default

Details

Note that times is interpreted differently here than elsewhere. In particular, it cannot be a length-1 vector.
Examples

```r
anim.curve(x^t, times=10:50/10, n=20)
anim.curve(sin(x*t), times=1:30, n=100, speed=12, col="darkgreen", from=-1, to=1)
```

## curve is constant in t, but window moves.
## NB: 'from' and 'to' control where the expression is evaluated.
## 'xlim' just controls the window.
```r
anim.curve(sin(cos(-x)*exp(x/2)), times=0:100/10, from=-5, to=10, n=500, 
    col="red", lwd=2, xlim=rbind(top <- seq(-5, 10, 1/10), top+5))
```

---

**anim.hist**  
*Draw an animated histogram.*

**Description**

Draw an animated histogram.

**Usage**

```r
anim.hist(
  x, density, angle, col, border, ...
  parameters passed to hist.
  times, show, speed, use.times, window, window.process
  see anim.plot.
)
```

**Arguments**

- `x, density, angle, col, border, ...`  
  parameters passed to `hist`.
- `times, show, speed, use.times, window, window.process`  
  see `anim.plot`.

**Details**

Parameters `x`, `density`, `angle`, `col` and `border` are all "chunked", i.e. first recycled to the length of `times` or `x` (whichever is longer), then split according to the unique values of `times`. See **anim.plot** for more details.
**Examples**

```r
anim.hist(rep(rnorm(5000), 7), times=rep(1:7, each=5000),
  breaks=c(5,10,20,50,100,200, 500, 1000))
```

---

**Description**

`anim.plot`

**Usage**

```r
anim.plot(...)  
anim.points(...)  
anim.lines(...)  
anim.text(...)  
## Default S3 method:  
anim.plot(  
  x,  
  y = NULL,  
  times = 1:length(x),  
  speed = 1,  
  show = TRUE,  
  use.times = TRUE,  
  window = if (identical(fn, lines)) t:(t + 1) else t,  
  window.process = NULL,  
  xlim = NULL,  
  ylim = NULL,  
  col = par("col"),  
  xaxp = NULL,  
  yaxp = NULL,  
  pch = par("pch"),  
  cex = 1,  
  labels = NULL,  
  asp = NULL,  
  lty = par("lty"),  
  lwd = par("lwd"),  
  fn = plot,  
  ...  
)
## S3 method for class 'formula'
```
anim.plot(
  formula,
  data = parent.frame(),
  subset = NULL,
  fn = plot,
  window = t,
  ...
)

## Default S3 method:
anim.points(...)

## Default S3 method:
anim.lines(...)

## Default S3 method:
anim.text(...)

anim.symbols(...)

## S3 method for class 'formula'
anim.points(formula, ...)

## S3 method for class 'formula'
anim.lines(formula, ...)

## S3 method for class 'formula'
anim.text(formula, ...)

Arguments

x, y  vectors of x and y coordinates. These can be passed in any way accepted by
       xy.coords.

times a vector of times. If times is length one, there will be that many frames, equally
divided over the length of x and y.
speed animation speed.
show if false, do not show plot; just return calls.
use.times if TRUE, animation speed is determined by the times argument. If FALSE, ani-
mation speed is constant.
window what window of times to show in each animation. The default, t, shows just
       plots from time t. To draw a plot incrementally, use window=1:t.
window.process function to call on each window of each times. See details.
xlim, ylim, col, pch arguments passed to plot.
labels, cex, lty, lwd  as above.
**anim.plot**

```r
asp, xaxp, yaxp, ...
```

- function called to create each frame.

**formula**

- a formula of the form \( y \sim x + \text{time} \).

**data**

- a data frame from where the values in formula should be taken.

**subset**

- a vector specifying which rows of data to use.

**Details**

Each unique level of `times` will generate a single frame of animation. The frames will be ordered by `times`.

In general:

- Parameters that apply to each point of the plot, such as `xlim`, `ylim`, `col`, `pch`, `labels` and `cex`, can be passed as vectors which will be recycled to `length(times)`.
- Parameters that apply to the plot as a whole, and always have length 1, such as `xlab` and `main`, can be passed as vectors and will be recycled to the number of frames.
- Parameters that apply to the plot as a whole, and can have length > 1, such as `xlim` and `ylim`, can be passed as vectors or matrices. If vectors, the same vector will be passed to every frame. If matrices, column `i` will be passed to the `i`'th frame.

`window.process` should be a function which takes two arguments: a list of potential arguments for the underlying call to `plot`, and a vector of times. The function should return the list of arguments after modification. This allows e.g. drawing "trails" of plot points. See the example

**Examples**

```r
x <- rep(1:100/10, 10)
times <- rep(1:10, each=100)
y <- sin(x*times/4)
anim.plot(x, y, times, type="l", col="orange", lwd=2)
```

```r
## changing colours - a per-point parameter
anim.plot(x, y, times, ylab="Sine wave", type="p", col=rainbow(100)[x *10])
```

```r
## changing line width - a whole-plot parameter
anim.plot(x, y, times, lwd=1:10, type="l")
```

```r
## times as a single number
anim.plot(1:10, 1:10, times=5)
```

```r
## incremental plot
anim.plot(1:10, 1:10, window=1:t)
```

```r
## moving window
anim.plot(1:10, 1:10, window=(t-2):t)
```

```r
## Formula interface
ChickWeight$chn <- as.numeric(as.factor(ChickWeight$Chick))
```
tmp <- anim.plot(weight ~ chn + Time, data=ChickWeight, col=as.numeric(Diet),
                 pch=as.numeric(Diet), speed=3)

# adding extra arguments:
replay(tmp, after=legend("topleft", legend=paste("Diet", 1:4), pch=1:4, col=1:4))

## Zooming in:
x <- rnorm(4000); y<- rnorm(4000)
x <- rep(x, 10); y <- rep(y, 10)
xlims <- 4*2^(-(1:10/10))
ylims <- xlims <- rbind(xlims, -xlims)
anim.plot(x, y, times=10, speed=5, xlim=xlims, ylim=ylims,
          col=rgb(0,0,0,.3), pch=19)

## window.process to create a faded "trail":
anim.plot(1:50, 1:50, speed=12, window=t:(t+5),
window.process=function(args, times){
  times <- times - min(times)
  alpha <- times/max(times)
  alpha[is.na(alpha)] <- 1
  args$col <- rgb(0,0,0, alpha)
  return(args)
})

## gapminder plot:
pl <- palette(adjustcolor(rainbow(23), 1, .6, .6, .6,
                          offset=c(0,0,0,-0.1)))
anim.plot(lifex ~ GDP + year, data=gm_data, log="x",
cex=sqrt(pop)*0.0004, pch=19, col=region, xlab="GDP",
        ylab="Life expectancy", speed=10, subset=year > 1850 & !year %% 5)
palette(pl)

## Not run:
## Earthquakes this week
if (require(\'maps\')) {
  eq = read.table(
file="http://earthquake.usgs.gov/earthquakes/catalogs/eqs7day-M1.txt",
fill=TRUE, sep=",", header=TRUE)
eq$time <- as.numeric(strptime(eq$Datetime, "%A, %B %d, %Y %X UTC"))
eq <- eq[-1,]
map('world')
maxdepth <- max(max(eq$Depth), 200)
tmp <- anim.points(Lat ~ Lon + time, data=eq, cex=Magnitude, col=rgb(1-Depth/maxdepth, 0, Depth/maxdepth,.7), pch=19, speed=3600*12,
show=FALSE)
replay(tmp, before=map('world', fill=TRUE, col="wheat"))
}

## Minard's plot
if (require(\'maps\')) {
  map('world', xlim=c(22, 40), ylim=c(52,58))
title("March of the Grande Armee on Moscow")
anim.save

points(cities$long, cities$lat, pch=18)
text(cities$long, cities$lat, labels=cities$city, pos=4, cex=.7)
with(troops[troops$group==1,], anim.lines(x=long,
y=lat, window=t:(t+1), speed=3, lwd=survivors/10000))
}

## End(Not run)

anim.save

Save an anim.frames object in various formats.

Description

This function simply calls replay on the object and then calls saveGIF and friends on the result.

Usage

anim.save(
  obj,
  filename,
  type = switch(file_ext(filename), gif = "GIF", mp4 = "Video", swf = "SWF", html = 
    "HTML", tex = "Latex"),
  ...
)

Arguments

obj an anim.frames object, or an expression to evaluate
filename file to save to
type one of 'GIF', 'Video', 'SWF', 'HTML', or 'Latex'
... arguments passed to e.g. saveGIF

Examples

## Not run:
tmp <- anim.plot(1:10, 1:10, pch=1:10, show=FALSE)
anim.save(tmp, "mygif.gif")

anim.save(replay(tmp, after=legend("topleft", legend="My legend")),
  "mygif2.gif")

## End(Not run)
**Description**

Draw an animation of line segments or arrows.

**Usage**

```r
anim.segments(
  x0, y0, x1 = NULL, y1 = NULL,
  times = NULL, speed = 1,
  show = TRUE,
  use.times = TRUE,
  window = t,
  window.process = NULL,
  fn = segments,
  col = NULL,
  lty = NULL,
  lwd = NULL,
  ...
)
```

```r
anim.arrows(..., length = 0.25, angle = 30, code = 2)
```

```r
anim.segmentplot(...)
```

```r
anim.arrowplot(...)
```

**Arguments**

- `x0, y0, x1, y1, col, lty, lwd, length, angle, code, ...`
  - arguments passed to `segments` or `arrows`
- `times, speed, show, use.times, window, window.process`
  - see `anim.plot` for details
- `fn`
  - underlying function to use

**Details**

`anim.segments` and `anim.arrows` draw lines on to an existing plot. If you want to redraw the plot between each frame, use `anim.arrowplot` or `anim.segmentplot`.

If both `x1` and `y1` are missing, then segments are plotted from the current time to the following time in each frame. If only `x1` is missing it is set equal to `x0`, similarly if only `y1` is missing.
Examples

```r
anim.segments(x0=rep(1:5, 5), y0=rep(1:5, each=5), y1=rep(2:6, each=5),
    times=rep(1:5, each=5))
```

## Short version

```r
anim.arrowplot(rep(1:5, 5), rep(1:5, each=5), times=5)
```

```r
if (require("maps")) {
    hr <- subset(hurricanes, lat > 0 & lat < 50 & lon > -95 & lon < -20 &
            Shour %% 6 == 0)
    hr$dlat <- cos(hr$diruv/360*2*pi) * hr$maguv / 8
    hr$dlon <- sin(hr$diruv/360*2*pi) * hr$maguv / 8
    hr$name <- sub("\s+", ", hr$name
    map("world", xlim=c(-95,-20), ylim=c(0,50))
    title("Hurricanes, 2009")
    with(hr[!duplicated(hr$name),], text(lon, lat,
            labels=paste0(name, "\n", Yr), cex=0.8))
    with(hr, anim.arrows(x0=lon, y0=lat, y1=lat+dlat, x1=lon+dlon,
            times=Shour, speed=12, col=rgb(0,0,1,0.8), length=.1, lwd=2))
}
```

---

**anim.smooth**

Smooth an *anim.frames* object

---

**Description**

Some export formats ignore information in the `times` attribute and plot frames at constant speed. *anim.smooth* creates a smoothed version of the *anim.frames* object with frames at constant intervals, suitable for export.

**Usage**

```r
anim.smooth(x, fps = 10)
```

**Arguments**

- `x` an *anim.frames* object
- `fps` how many frames per second to smooth to

**Details**

Note that plot parameters such as `x` and `y` positions are not interpolated. If you want your whole animation to look smoother, you have to do the work yourself using e.g. `approx`.

If you smooth to a large value of `fps`, the animations may look bad in R because they overtax the graphics engine. They should still look good when saved, though.

**Value**

A smoothed *anim.frames* object, with the speed attribute equal to `fps`. 
Examples

```
accel <- anim.plot(1, 1:30, times=sqrt(1:30))
## Not run:
anim.save(accel, "GIF", "wrong.gif")

## Not run:
accel <- anim.smooth(accel, fps=20)
## Not run:
anim.save(accel, "GIF", "better.gif")
## End(Not run)
```

cities

**Cities near the Grande Armée’s march on Moscow**

Description

Cities near the Grande Armée’s march on Moscow

---

gm_data

**Gapminder GDP, life expectancy and population data**

Description

Gapminder GDP, life expectancy and population data

Source

http://gapminder.org

---

hurricanes

**Wind speed data for hurricanes in 2009**

Description

Wind speed data for hurricanes in 2009

Source

http://myweb.fsu.edu/jelsner/Data.html
**merge.anim.frames**

**Merge anim.frames objects**

**Description**

Merge two or more anim.frames objects to create a new anim.frames object.

**Usage**

```r
## S3 method for class 'anim.frames'
merge(..., speed = NULL)
```

**Arguments**

- `...`: anim.frames objects returned from, e.g. `anim.plot`
- `speed`: speed for the merged object. This may be left unspecified only if all objects have the same speed.

**Details**

If two or more calls in the merged animation are at the same time, calls from the earlier object in `...` will be run first.

If you merge two animations from `anim.plot`, `plot.window` will be called before each frame of the merged animation. This may not be what you want. Instead, use `anim.points` or similar for all but the first animation.

**Examples**

```r
tmp <- anim.plot(1:5, 1:5, speed=2)
tmp2 <- anim.plot(1:5, 5:1, col="red", speed=2)
## Not what you want:
replay(merge(tmp, tmp2))

## better:
tmp3 <- anim.points(1:5, 5:1,col="red", speed=2)
newf <- merge(tmp, tmp3)
replay(newf)
## NB: result of the merge looks different from the two
## individual animations

## not the same:
newf2 <- merge(tmp2, tmp)
## points will be called before plot!
replay(newf2)
```
**Description**

A 2x3x20 array of data from a laboratory public goods game. Dimensions are Picked (was subject picked for punishment?), Contribution (of subject: Non-unique lowest, Not lowest/all same and Unique lowest), and Period.

**Details**

Provided by the package author.

---

**replay**

*Replay an anim.frames object*

---

**Description**

Replay all or some of the frames of an object.

**Usage**

replay(...)

```r
## S3 method for class 'anim.frames'
replay(
  x,
  frames = 1:length(x),
  speed = attr(x, "speed"),
  after = NULL,
  before = NULL,
  ...
)
```

```r
## S3 method for class 'anim.frames'
plot(x, ...)
```

**Arguments**

- `...`: other arguments (not currently used)
- `x`: an anim.frames object
- `frames`: numeric vector specifying which frames to replay
- `speed`: a new speed
- `after`: an expression to evaluate after each frame is plotted
- `before`: an expression to evaluate before each frame is plotted
Details

before and after will have the arguments from the frame’s call available in their environment - see the example.

The plot method simply calls replay.

Examples

```r
calling <- anim.plot(1:10, 1:10, speed=3)
calling <- anim.plot(1:10, 1:10, speed=5)
calling <- anim.plot(1:10, 1:10, frames=c(1,5,6,10))
```

```r
calling <- anim.plot(x<-rnorm(100), x+rnorm(100,0,3), 20, window=1:t, show=FALSE, main="Regressions as sample size increases")
calling <- anim.plot(1:10, 1:10, after=abline(lm(y~x), col="red"))
```
# Index

<table>
<thead>
<tr>
<th>Function/Package</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>anim.arrowplot (anim.segments)</td>
<td>12</td>
</tr>
<tr>
<td>anim.arrows (anim.segments)</td>
<td>12</td>
</tr>
<tr>
<td>anim.barplot</td>
<td>2</td>
</tr>
<tr>
<td>anim.contour</td>
<td>4</td>
</tr>
<tr>
<td>anim.curve</td>
<td>5</td>
</tr>
<tr>
<td>anim.filled.contour (anim.contour)</td>
<td>4</td>
</tr>
<tr>
<td>anim.hist</td>
<td>6</td>
</tr>
<tr>
<td>anim.lines (anim.plot)</td>
<td>7</td>
</tr>
<tr>
<td>anim.persp (anim.contour)</td>
<td>4</td>
</tr>
<tr>
<td>anim.plot.3, 4, 6, 7, 12, 15</td>
<td></td>
</tr>
<tr>
<td>anim.plot.default</td>
<td>5</td>
</tr>
<tr>
<td>anim.plots-package</td>
<td>2</td>
</tr>
<tr>
<td>anim.points (anim.plot)</td>
<td>7</td>
</tr>
<tr>
<td>anim.save</td>
<td>11</td>
</tr>
<tr>
<td>anim.segmentplot (anim.segments)</td>
<td>12</td>
</tr>
<tr>
<td>anim.segments</td>
<td>12</td>
</tr>
<tr>
<td>anim.smooth</td>
<td>13</td>
</tr>
<tr>
<td>anim.symbols (anim.plot)</td>
<td>7</td>
</tr>
<tr>
<td>anim.text (anim.plot)</td>
<td>7</td>
</tr>
<tr>
<td>approx</td>
<td>13</td>
</tr>
<tr>
<td>arrows</td>
<td>12</td>
</tr>
<tr>
<td>barplot</td>
<td>3</td>
</tr>
<tr>
<td>cities</td>
<td>14</td>
</tr>
<tr>
<td>contour</td>
<td>4</td>
</tr>
<tr>
<td>curve</td>
<td>5</td>
</tr>
<tr>
<td>formula</td>
<td>9</td>
</tr>
<tr>
<td>gm_data</td>
<td>14</td>
</tr>
<tr>
<td>hist</td>
<td>6</td>
</tr>
<tr>
<td>hurricanes</td>
<td>14</td>
</tr>
<tr>
<td>merge.anim.frames</td>
<td>15</td>
</tr>
<tr>
<td>persp</td>
<td>4</td>
</tr>
<tr>
<td>PGgame</td>
<td>16</td>
</tr>
<tr>
<td>plot</td>
<td>8</td>
</tr>
<tr>
<td>plot.anim.frames (replay)</td>
<td>16</td>
</tr>
<tr>
<td>replay</td>
<td>16</td>
</tr>
<tr>
<td>saveGIF</td>
<td>11</td>
</tr>
<tr>
<td>segments</td>
<td>12</td>
</tr>
<tr>
<td>temps</td>
<td>17</td>
</tr>
<tr>
<td>troops</td>
<td>17</td>
</tr>
<tr>
<td>xy.coords</td>
<td>8</td>
</tr>
</tbody>
</table>