Package ‘animation’

August 16, 2015

Type Package

Title A Gallery of Animations in Statistics and Utilities to Create Animations

Version 2.4

Date 2015-08-16

Maintainer Yihui Xie <xie@yihui.name>

Description Provides functions for animations in statistics, covering topics in probability theory, mathematical statistics, multivariate statistics, nonparametric statistics, sampling survey, linear models, time series, computational statistics, data mining and machine learning. These functions may be helpful in teaching statistics and data analysis. Also provided in this package are a series of functions to save animations to various formats, e.g. Flash, GIF, HTML pages, PDF and videos (saveSWF(), saveGIF(), saveHTML(), saveLatex(), and saveVideo() respectively). PDF animations can be inserted into Sweave/knitr easily.

SystemRequirements ImageMagick (http://imagemagick.org) or GraphicsMagick (http://www.graphicsmagick.org) or LyX (http://www.lyx.org) for saveGIF(); (PDF)LaTeX for saveLatex(); SWF Tools (http://swftools.org) for saveSWF(); FFmpeg (http://ffmpeg.org) or avconv (https://libav.org/avconv.html) for saveVideo()

Depends R (>= 2.14.0)

Suggests MASS, testit

License GPL

URL http://yihui.name/animation

BugReports https://github.com/yihui/animation/issues

LazyData yes

NeedsCompilation no

Author Yihui Xie [cre, aut],
Christian Mueller [ctb],
Lijia Yu [ctb],
Weicheng Zhu [ctb]

Repository CRAN

Date/Publication 2015-08-16 07:30:50

R topics documented:

`animation-package` *A Gallery of Animations in Statistics and Utilities to Create Animations*

Description

This package contains a variety functions for animations in statistics which could probably aid in teaching statistics and data analysis; it also has several utilities to export R animations to other formats.

Details

This package mainly makes use of HTML & JavaScript and R windows graphics devices (such as `x11`) to demonstrate animations in statistics; other kinds of output such as Flash (SWF) or GIF animations or PDF animations are also available if required software packages have been installed. See below for details on each type of animation.

On-screen Animations

It's natural and easy to create an animation in R using the windows graphics device, e.g. in `x11()` or `windows()`. A basic scheme is like the Example 1 (see below).

On-screen animations do not depend on any third-party software, but the rendering speed of the windows graphics devices is often slow, so the animation might not be smooth (especially under Linux and Mac OS).

HTML Pages

The generation of HTML animation pages does not rely on any third-party software either, and we only need a web browser to watch the animation. See `saveHTML`.

The HTML interface is just like a movie player – it comes with a series of buttons to control the animation (play, stop, next, previous, ...).

This HTML approach is flexible enough to be used even in Rweb, which means we do not really have to install R to create animations! There is a demo in `system.file('misc', 'Rweb', 'demo.html', package = 'ani`

We can use `saveHTML` to create animations directly in Rweb; this can be helpful when we do not have R or cannot install R.

GIF Animations

If ImageMagick or GraphicsMagick has been installed, we can use `im.convert` or `gm.convert` to create a GIF animation (combining several R plots together), or use `saveGIF` to create a GIF animation from an R code chunk.

Flash Animations

If SWF Tools has been installed, we can use `saveSWF` to create a Flash animation (again, combining R plots).

PDF Animations

If LaTeX is present in the system, we can use `saveLatex` to insert animations into a PDF document and watch the animation using the Adobe reader.

The animation is created by the LaTeX package `animate`.

Video

The function `saveVideo` can use FFmpeg to convert images to various video formats (e.g. '`mp4`', '`avi`' and '`wmv`', etc).

Note

Bug reports and feature requests can be sent to `https://github.com/yihui/animation/issues`.

Author(s)

Yihui Xie

References

The associated website for this package: `http://vis.supstat.com`

Yihui Xie and Xiaoyue Cheng. animation: A package for statistical animations. *R News*, **8**(2):23–27, October 2008. URL: `http://CRAN.R-project.org/doc/Rnews/Rnews_2008-2.pdf`

(NB: some functions mentioned in the above article have been slightly modified; see the help pages for the up-to-date usage.)

Yihui Xie (2013). animation: An R Package for Creating Animations and Demonstrating Statistical Methods. *Journal of Statistical Software*, **53**(1), 1-27. URL `http://www.jstatsoft.org/v53/i01/`.

See Also

`saveHTML`, `saveGIF`, `saveSWF`, `saveVideo`, `saveLatex`

Examples

```
### 1. How to setup a simple animation ###

## set some options first
oopt = ani.options(interval = 0.2, nmax = 10)
## use a loop to create images one by one
for (i in 1:ani.options("nmax")) {
    plot(rnorm(30))
    ani.pause()  ## pause for a while ('interval')
}
## restore the options
ani.options(oopt)

## see ?ani.record for an alternative way to set up an animation

### 2. Animations in HTML pages ###
saveHTML({
    ani.options(interval = 0.05, nmax = 30)
    par(mar = c(3, 3, 2, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8,
        cex.lab = 0.8, cex.main = 1)
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow",
        main = "Demonstration of Brownian Motion")
}, img.name = "bm_plot", title = "Demonstration of Brownian Motion",
    description = c("Random walk on the 2D plane: for each point",
        "(x, y), x = x + rnorm(1) and y = y + rnorm(1)."))

### 3. GIF animations ###
saveGIF({
    ani.options(nmax = 30)
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, interval = 0.05, movie.name = "bm_demo.gif", ani.width = 600, ani.height = 600)


### 4. Flash animations ###
saveSWF({
    par(mar = c(3, 2.5, 1, 0.2), pch = 20, mgp = c(1.5, 0.5, 0))
    buffon.needle(type = "S")
}, ani.dev = "pdf", ani.type = "pdf", swf.name = "buffon.swf", interval = 0.1,
    nmax = 40, ani.height = 7, ani.width = 7)


### 5. PDF animations ###
saveLatex({
    par(mar = c(3, 3, 1, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8,
        cex.lab = 0.8, cex.main = 1)
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow",
        main = "Brownian Motion")
}, img.name = "BM_plot", latex.filename = ifelse(interactive(), "brownian_motion.tex",
    ""), interval = 0.1, nmax = 20)
```

`ani.options` *Set or query animation options*

Description

There are various parameters that control the behaviour of the animation, such as time interval, maximum number of animation frames, height and width, etc.

Usage

```
ani.options(...)
```

Arguments

`...`	arguments in `tag = value` form, or a list of tagged values. The tags usually come from the animation parameters described below, but they are not restricted to these tags (any tag can be used; this is similar to `options`).

Value

`ani.options()` returns a list containing the options: when parameters are set, their former values are returned in an invisible named list. Such a list can be passed as an argument to `ani.options` to restore the parameter values.

`ani.options('tag')` returns the value of the option `'tag'`.

`ani.options(c('tag1', 'tag2'))` or `ani.options('tag1', 'tag2')` returns a list containing the corresponding options.

Animation options

The supported animation parameters:

interval a positive number to set the time interval of the animation (unit in seconds); default to be 1.

nmax maximum number of steps in a loop (e.g. iterations) to create animation frames. Note: the actual number of frames can be less than this number, depending on specific animations. Default to be 50.

ani.width, ani.height width and height of image frames (unit in px); see graphics devices like `png`, `jpeg`, ...; default to be 480. NB: for different graphics devices, the units of these values might be different, e.g. PDF devices usually use inches, whereas bitmap devices often use pixels.

imgdir character: the name of the directory (a relative path) for images when creating HTML animation pages; default to be `'images'`.

htmlfile character: name of the target HTML main file (without path name; basename only; default to be `'index.html'`)

ani.dev a function or a function name: the graphics device; e.g. (`png`, `pdf`, ...); default to be `'png'`

ani.type character: image format for animation frames, e.g. png, jpeg, ...; default to be 'png'; this will be used as the file extension of images, so don't forget to change this option as well when you changed the option `ani.dev`

title, description character: the title and description of the animation in the HTML page created by `saveHTML`

verbose logical or character: if `TRUE`, write a footer part in the HTML page containing detailed technical information; if given a character string, it will be used as the footer message; in other cases, the footer of the page will be blank.

loop whether to iterate or not (default `TRUE` to iterate for infinite times)

autobrowse logical: whether auto-browse the animation page immediately after it is created? (default to be `interactive()`)

autoplay logical: whether to autoplay the animation when the HTML page is loaded (default to be `TRUE`); only applicable to `saveHTML`

use.dev whether to use the graphics device specified in `ani.options('ani.dev')` (default to be `TRUE`); if `FALSE`, we need to generate image files by our own approaches in the expression `expr` (see functions `saveHTML`, `saveGIF`, `saveLatex` and `saveSWF`); this can be useful when the output cannot be captured by standard R graphics devices – a typical example is the **rgl** graphics (we can use `rgl.snapshot` to capture **rgl** graphics to png files, or `rgl.postscript` to save plots as postscript/pdf; see `demo('rgl_animation')` or `demo('use_Cairo')` for examples or the last example below). Note, however, we do not really have to create the images using R graphics devices – see `demo('flowers')` on how to download images from the Internet and create an HTML animation page!

Hidden options

There are a couple of "hidden" options which are designed to facilitate the usage of some functions but are not initialized like the above options when the package is loaded, including:

convert this option will be checked first when calling `im.convert` (or `saveGIF`) to see if it contains the path to '`convert.exe`'; we can specify it beforehand to save the efforts in searching for '`convert.exe`' in ImageMagick under Windows. For example, `ani.options(convert = 'c:/program files/imagemagick/convert.exe')`; note this option also works for Mac and Linux (see `help(im.convert)`)

swftools this can help `saveSWF` save the efforts of searching for the software package "SWF Tools" under Windows; e.g. we can specify `ani.options(swftools = 'c:/program files/swftools')` in advance

img.fmt the value of this option can be used to determine the image filename format when we want to use custom graphics devices to record images, e.g. in `saveLatex`, if `ani.options('use.dev') == FALSE`, then `ani.options('img.fmt')` will be a string like `'path/to/output/img.name%d.png'`, so we can use it to generate file names in the argument `expr`; see `demo('rgl_animation')` for example or the last example below

qpdf the path of the program `qpdf`, e.g. `ani.options(qpdf = 'C:/Software/qpdf/bin/qpdf.exe')`; `qpdf` is mainly used to compress PDF files in this package, and it is a smaller tool than `pdftk`. It is recommended over `pdftk` especially under Linux, because tests show that `pdftk` does not work well under Linux in compressing PDF files, while `qpdf` is much better.

pdftk the path of the program Pdftk, e.g. `ani.options(pdftk = 'C:/Software/pdftk.exe')` or `ani.options(pdftk = '/home/john/bin/pdftk')`; pdftk will be used to compress the PDF graphics output in the function pdftk; compression will not be tried if this options is NULL. This option will only affect saveGIF, saveLatex and saveSWF when `ani.options('ani.type')` is 'pdf'.

ffmpeg the path of the progam ffmpeg, e.g. `ani.options(ffmpeg = 'C:/Software/ffmpeg/bin/ffmpeg.exe')`; FFmpeg is used to convert a sequence of images to a video. See saveVideo.

Note

Please note that nmax is not always equal to the number of animation frames. Sometimes there is more than one frame recorded in a single step of a loop, for instance, there are 2 frames generated in each step of kmeans.ani, and 4 frames in knn.ani, etc; whereas for newton.method, the number of animation frames is not definite, because there are other criteria to break the loop.

This function can be used for almost all the animation functions such as brownian.motion, boot.iid, buffon.needle, cv.ani, flip.coin, kmeans.ani, knn.ani, etc. Most of the options here will affect the behaviour of animations of the formats HTML, GIF, SWF and PDF; on-screen animations are only affected by interval and nmax.

Author(s)

Yihui Xie

See Also

options, dev.interactive, saveHTML, saveGIF, saveLatex, saveSWF, pdftk

http://qpdf.sourceforge.net/

http://www.pdflabs.com/docs/pdftk-man-page/

Examples

```
## see the first example in help(animation) on how to set and
## restore animation options

## use the PDF device: remember to set 'ani.type' accordingly
oopt = ani.options(ani.dev = "pdf", ani.type = "pdf", ani.height = 5,
    ani.width = 7)

## use the Cairo PDF device if (require('Cairo')) {
## ani.options(ani.dev = CairoPDF, ani.type = 'pdf', ani.height =
## 6, ani.width = 6) }

## don't loop for GIF/HTML animations
ani.options(loop = FALSE)

## don't try to open the output automatically
ani.options(autobrowse = FALSE)

## it's a good habit to restore the options in the end so that
## other code will not be affected
```

```
ani.options(oopt)

## how to make use of the hidden option 'img.fmt'
saveHTML(expr = {
    png(ani.options("img.fmt"))
    for (i in 1:5) plot(runif(10))
    dev.off()
}, img.name = "custom_plot", use.dev = FALSE, ani.type = "png",
    description = "Note how we use our own graphics device in 'expr'.",
    htmlfile = "custom_device.html")
```

`ani.pause` *Pause for a while and flush the current graphical device*

Description

If this function is called in an interactive graphics device, it will pause for a time interval (by default specified in `ani.options('interval')`) and flush the current device; otherwise it will do nothing.

Usage

```
ani.pause(interval = ani.options("interval"))
```

Arguments

`interval`	a time interval to pause (in seconds)

Value

Invisible `NULL`.

Author(s)

Yihui Xie

See Also

`dev.interactive`, `Sys.sleep`, `dev.flush`

Examples

```
## pause for 2 seconds
oopt = ani.options(interval = 2)

for (i in 1:5) {
    plot(runif(10), ylim = c(0, 1))
    ani.pause()
}

ani.options(oopt)
```

```
## see demo('Xmas2', package = 'animation') for another example
```

`ani.record` *Record and replay animations*

Description

These two functions use `recordPlot` and `replayPlot` to record image frames and replay the animation respectively.

Replay the animation

Usage

```
ani.record(reset = FALSE, replay.cur = FALSE)

ani.replay(list)
```

Arguments

`reset`	if `TRUE`, the recording list will be cleared, otherwise new plots will be appended to the existing list of recorded plots
`replay.cur`	whether to replay the current plot (we can set both `reset` and `replay.cur` to `TRUE` so that low-level plotting changes can be captured by off-screen graphics devices without storing all the plots in memory; see Note)
`list`	a list of recorded plots; if missing, the recorded plots by `ani.record` will be used

Details

One difficulty in capturing images in R (base graphics) is that the off-screen graphics devices cannot capture low-level plotting commands as *new* image files – only high-level plotting commands can produce new image files; `ani.record` uses `recordPlot` to record the plots when any changes are made on the current plot. For a graphical device to be recordable, you have to call `dev.control('enable')` before plotting.

`ani.replay` can replay the recorded plots as an animation. Moreover, we can convert the recorded plots to other formats too, e.g. use `saveHTML` and friends.

The recorded plots are stored as a list in `.ani.env$.images`, which is the default value to be passed to `ani.replay`; `.ani.env` is an invisible `environment` created when this package is loaded, and it will be used to store some commonly used objects such as animation options (`ani.options`).

Value

Invisible `NULL`.

Note

Although we can record changes made by low-level plotting commands using `ani.record`, there is a price to pay – we need memory to store the recorded plots, which are usually verg large when the plots are complicated (e.g. we draw millions of points or polygons in a single plot). However, we can set `replay.cur` to force R to produce a new copy of the current plot, which will be automatically recorded by off-screen grapihcs devices as *new* image files. This method has a limitation: we must open a screen device to assist R to record the plots. See the last example below. We must be very careful that no other graphics devices are opened before we use this function.

If we use base graphics, we should bear in mind that the background colors of the plots might be transparent, which could lead to problems in HTML animation pages when we use the `png` device (see the examples below).

Author(s)

Yihui Xie

See Also

`recordPlot` and `replayPlot`; `ani.pause`

Examples

```
library(animation)

n = 20
x = sort(rnorm(n))
y = rnorm(n)
## set up an empty frame, then add points one by one
par(bg = "white")  # ensure the background color is white
plot(x, y, type = "n")

ani.record(reset = TRUE)  # clear history before recording

for (i in 1:n) {
    points(x[i], y[i], pch = 19, cex = 2)
    ani.record()  # record the current frame
}

## now we can replay it, with an appropriate pause between frames
oopts = ani.options(interval = 0.5)
ani.replay()

## or export the animation to an HTML page
saveHTML(ani.replay(), img.name = "record_plot")

## record plots and replay immediately
saveHTML({
    dev.control("enable")  # enable recording
    par(bg = "white")  # ensure the background color is white
    plot(x, y, type = "n")
    for (i in 1:n) {
```

```
        points(x[i], y[i], pch = 19, cex = 2)
        ani.record(reset = TRUE, replay.cur = TRUE)  # record the current frame
    }
})

ani.options(oopts)
```

`bisection.method` *Demonstration of the Bisection Method for root-finding on an interval*

Description

This is a visual demonstration of finding the root of an equation $f(x) = 0$ on an interval using the Bisection Method.

Usage

```
bisection.method(FUN = function(x) x^2 - 4, rg = c(-1, 10), tol = 0.001,
    interact = FALSE, main, xlab, ylab, ...)
```

Arguments

`FUN`	the function in the equation to solve (univariate)
`rg`	a vector containing the end-points of the interval to be searched for the root; in a `c(a, b)` form
`tol`	the desired accuracy (convergence tolerance)
`interact`	logical; whether choose the end-points by cliking on the curve (for two times) directly?
`xlab, ylab, main`	axis and main titles to be used in the plot
`...`	other arguments passed to `curve`

Details

Suppose we want to solve the equation $f(x) = 0$. Given two points a and b such that $f(a)$ and $f(b)$ have opposite signs, we know by the intermediate value theorem that f must have at least one root in the interval $[a, b]$ as long as f is continuous on this interval. The bisection method divides the interval in two by computing $c = (a + b)/2$. There are now two possibilities: either $f(a)$ and $f(c)$ have opposite signs, or $f(c)$ and $f(b)$ have opposite signs. The bisection algorithm is then applied recursively to the sub-interval where the sign change occurs.

During the process of searching, the mid-point of subintervals are annotated in the graph by both texts and blue straight lines, and the end-points are denoted in dashed red lines. The root of each iteration is also plotted in the right margin of the graph.

Value

A list containing

root	the root found by the algorithm
value	the value of FUN(root)
iter	number of iterations; if it is equal to ani.options('nmax'), it's quite likely that the root is not reliable because the maximum number of iterations has been reached

Note

The maximum number of iterations is specified in ani.options('nmax').

Author(s)

Yihui Xie

References

http://en.wikipedia.org/wiki/Bisection_method

See Also

deriv, uniroot, curve

Examples

```
oopt = ani.options(nmax = ifelse(interactive(), 30, 2))

## default example
xx = bisection.method()
xx$root  # solution

## a cubic curve
f = function(x) x^3 - 7 * x - 10
xx = bisection.method(f, c(-3, 5))

## interaction: use your mouse to select the two
## end-points
if (interactive()) bisection.method(f, c(-3, 5), interact = TRUE)

## HTML animation pages
saveHTML({
    par(mar = c(4, 4, 1, 2))
    bisection.method(main = "")
}, img.name = "bisection.method", htmlfile = "bisection.method.html",
    ani.height = 400, ani.width = 600, interval = 1,
    title = "The Bisection Method for Root-finding on an Interval",
    description = c("The bisection method is a root-finding algorithm",
        "which works by repeatedly dividing an interval in half and then",
        "selecting the subinterval in which a root exists."))
```

```
ani.options(oopt)
```

BM.circle *Brownian Motion in a circle*

Description

Several points moving randomly in a circle.

Usage

```
BM.circle(n = 20, col = rainbow(n), ...)
```

Arguments

`n`	number of points
`col`	colors of points
`...`	other parameters passed to `points`

Details

This is a solution to the question raised in R-help: `https://stat.ethz.ch/pipermail/r-help/2008-December/183018.html`.

Value

Invisible `NULL`.

Note

The maximum number of steps in the motion is specified in `ani.options('nmax')`.

Author(s)

Yihui Xie

References

`http://vis.supstat.com/2012/11/brownian-motion-with-r/`

See Also

`brownian.motion`, `rnorm`

Examples

```
oopt = ani.options(interval = 0.1, nmax = ifelse(interactive(),
    300, 2))
par(mar = rep(0.5, 4))
BM.circle(cex = 2, pch = 19)

saveHTML({
    par(mar = rep(0.5, 4), pch = 19)
    ani.options(interval = 0.05, nmax = ifelse(interactive(),
        100, 10))
    BM.circle(cex = 2, pch = 19)
}, img.name = "BM.circle", htmlfile = "BM.circle.html",
    ani.height = 450, ani.width = 450, single.opts = paste("'controls':",
        "['first', 'previous', 'play', 'next', 'last', 'loop', 'speed'],",
        "'delayMin': 0"), title = "Brownian Motion in a Circle",
    description = "Brownian Motion in a circle.")

ani.options(oopt)
```

`boot.iid` *Demonstrate bootstrapping for iid data*

Description

Use a sunflower scatter plot to illustrate the results of resampling, and a histogram to show the distribution of the statistic of interest.

Usage

```
boot.iid(x = runif(20), statistic = mean, m = length(x), mat = matrix(1:2, 2),
   widths = rep(1, ncol(mat)), heights = rep(1, nrow(mat)), col = c("black", "red",
        "bisque", "red", "gray"), cex = c(1.5, 0.8), main, ...)
```

Arguments

`x` a numerical vector (the original data).

`statistic` A function which returns a value of the statistic of interest when applied to the data x.

`m` the sample size for bootstrapping (m-out-of-n bootstrap)

`mat, widths, heights` arguments passed to `layout` to set the layout of the two graphs

`col` a character vector of length 5 specifying the colors of: points of original data, points for the sunflowerplot, rectangles of the histogram, the density line, and the rug.

`cex` a numeric vector of length 2: magnification of original data points and the sunflowerplot points.

main	a character vector of length 2: the main titles of the two graphs.
...	other arguments passed to hist

Details

This is actually a very naive version of bootstrapping but may be useful for novices. By default, the circles denote the original dataset, while the red sunflowers (probably) with leaves denote the points being resampled; the number of leaves just means how many times these points are resampled, as bootstrap samples *with* replacement. The x-axis is the sample values, and y-axis is the indices of sample points.

The whole process has illustrated the steps of resampling, computing the statistic and plotting its distribution based on bootstrapping.

Value

A list containing

t0	The observed value of 'statistic' applied to 'x'.
tstar	Bootstrap versions of the 'statistic'.

Note

The maximum times of resampling is specified in `ani.options('nmax')`.

Author(s)

Yihui Xie

References

There are many references explaining the bootstrap and its variations.

Efron, B. and Tibshirani, R. (1993) *An Introduction to the Bootstrap*. Chapman & Hall.

See Also

sunflowerplot

Examples

```
## bootstrap for 20 random numbers from U(0, 1)
par(mar = c(1.5, 3, 1, 0.1), cex.lab = 0.8, cex.axis = 0.8, mgp = c(2,
    0.5, 0), tcl = -0.3)
oopt = ani.options(nmax = ifelse(interactive(), 50, 2))
## don't want the titles
boot.iid(main = c("", ""))

## for the median of 15 points from chi-square(5)
boot.iid(x = rchisq(15, 5), statistic = median, main = c("",
    ""))
```

```
## change the layout; or you may try 'mat = matrix(1:2, 1)'
par(mar = c(1.5, 3, 2.5, 0.1), cex.main = 1)
boot.iid(heights = c(1, 2))

## save the animation in HTML pages
saveHTML({
    par(mar = c(2.5, 4, 0.5, 0.5))
    ani.options(nmax = ifelse(interactive(), 50, 10))
    boot.iid(main = c("", ""), heights = c(1, 2))
}, img.name = "boot.iid", htmlfile = "boot.iid.html", ani.height = 500,
    ani.width = 600, title = "Bootstrapping the i.i.d data",
    description = c("This is a naive version of bootstrapping but",
        "may be useful for novices."))

ani.options(oopt)
```

`boot.lowess` *Bootstrapping with LOWESS*

Description

Sample the original data with replacement and fit LOWESS curves accordingly.

Usage

```
boot.lowess(x, y = NULL, f = 2/3, iter = 3, line.col = "#FF000033", ...)
```

Arguments

`x, y, f, iter`	passed to `lowess`
`line.col`	the color of the LOWESS lines
`...`	other arguments passed to the scatterplot by `plot`

Details

We keep on resampling the data and finally we will see several bootstrapped LOWESS curves, which may give us a rough idea about a "confidence interval" of the LOWESS fit.

Author(s)

Yihui Xie

Examples

```
oopt = ani.options(nmax = if (interactive()) 100 else 2,
    interval = 0.02)

boot.lowess(cars, pch = 20, xlab = "speed", ylab = "dist")

boot.lowess(cars, f = 1/3, pch = 20)

## save in HTML pages
saveHTML({
    par(mar = c(4.5, 4, 0.5, 0.5))
    boot.lowess(cars, f = 1/3, pch = 20, xlab = "speed",
        ylab = "dist")
}, img.name = "boot_lowess", imgdir = "boot_lowess",
    interval = 0.1, title = "Bootstrapping with LOWESS",
    description = "Fit LOWESS curves repeatedly via bootstrapping.")

ani.options(oopt)
```

`brownian.motion` *Demonstration of Brownian motion on the 2D plane*

Description

Brownian motion, or random walk, can be regarded as the trace of some cumulative normal random numbers.

Usage

```
brownian.motion(n = 10, xlim = c(-20, 20), ylim = c(-20, 20), ...)
```

Arguments

`n`	Number of points in the scatterplot
`xlim, ylim`	Arguments passed to `plot.default` to control the apperance of the scatterplot (title, points, etc), see `points` for details.
`...`	other arguments passed to `plot.default`

Details

The location of the next step is "current location + random Gaussian numbers", i.e.,

$$x_{k+1} = x_k + rnorm(1)$$

$$y_{k+1} = y_k + rnorm(1)$$

where *(x, y)* stands for the location of a point.

Value

None (invisible NULL).

Note

The maximum number of steps in the motion is specified in ani.options('nmax').

Author(s)

Yihui Xie

References

http://vis.supstat.com/2012/11/brownian-motion-with-r

See Also

rnorm

Examples

```
oopt = ani.options(interval = 0.05, nmax = ifelse(interactive(),
    150, 2))
brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow",
    main = "Demonstration of Brownian Motion")
ani.options(oopt)

## create an HTML animation page
saveHTML({
    par(mar = c(3, 3, 1, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3,
        cex.axis = 0.8, cex.lab = 0.8, cex.main = 1)
    ani.options(interval = 0.05, nmax = ifelse(interactive(),
        150, 10))
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, description = c("Random walk on the 2D plane: for each point",
    "(x, y), x = x + rnorm(1) and y = y + rnorm(1)."),
    title = "Demonstration of Brownian Motion")

ani.options(oopt)
```

buffon.needle *Simulation of Buffon's Needle*

Description

This function provides a simulation for the problem of Buffon's Needle, which is one of the oldest problems in the field of geometrical probability.

Usage

```
buffon.needle(l = 0.8, d = 1, redraw = TRUE, mat = matrix(c(1, 3, 2, 3), 2),
   heights = c(3, 2), col = c("lightgray", "red", "gray", "red", "blue", "black",
        "red"), expand = 0.4, type = "l", ...)
```

Arguments

`l`	numerical. length of the needle; shorter than `d`.
`d`	numerical. distances between lines; it should be longer than `l`.
`redraw`	logical. redraw former 'needles' or not for each drop.
`mat, heights`	arguments passed to `layout` to set the layout of the three graphs.
`col`	a character vector of length 7 specifying the colors of: background of the area between parallel lines, the needles, the sin curve, points below / above the sin curve, estimated π values, and the true π value.
`expand`	a numerical value defining the expanding range of the y-axis when plotting the estimated π values: the `ylim` will be `(1 +/- expand) * pi`.
`type`	an argument passed to `plot` when plotting the estimated π values (default to be lines).
`...`	other arguments passed to `plot` when plotting the values of estimated π.

Details

This is quite an old problem in probability. For mathematical background, please refer to `http://en.wikipedia.org/wiki/Buffon's_needle` or `http://www.mste.uiuc.edu/reese/buffon/buffon.html`.

'Needles' are denoted by segments on the 2D plane, and dropped randomly to check whether they cross the parallel lines. Through many times of 'dropping' needles, the approximate value of π can be calculated out.

There are three graphs made in each step: the top-left one is a simulation of the scenario, the top-right one is to help us understand the connection between dropping needles and the mathematical method to estimate π, and the bottom one is the result for each drop.

Value

The values of estimated π are returned as a numerical vector (of length `nmax`).

Note

Note that `redraw` has great influence on the speed of the simulation (animation) if the control argument `nmax` (in `ani.options`) is quite large, so you'd better specify it as `FALSE` when doing a large amount of simulations.

The maximum number of drops is specified in `ani.options('nmax')`.

Author(s)

Yihui Xie

References

Ramaley, J. F. (Oct 1969). Buffon's Noodle Problem. *The American Mathematical Monthly* **76** (8): 916-918.

http://vis.supstat.com/2013/04/buffons-needle

Examples

```
## it takes several seconds if 'redraw = TRUE'
oopt = ani.options(nmax = ifelse(interactive(), 500, 2), interval = 0.05)
par(mar = c(3, 2.5, 0.5, 0.2), pch = 20, mgp = c(1.5, 0.5, 0))
buffon.needle()

## this will be faster
buffon.needle(redraw = FALSE)

## create an HTML animation page
saveHTML({
    par(mar = c(3, 2.5, 1, 0.2), pch = 20, mgp = c(1.5, 0.5, 0))
    ani.options(nmax = ifelse(interactive(), 300, 10), interval = 0.1)
    buffon.needle(type = "S")
}, img.name = "buffon.needle", htmlfile = "buffon.needle.html",
    ani.height = 500, ani.width = 600, title = "Simulation of Buffon's Needle",
    description = c("There are three graphs made in each step: the",
        "top-left, one is a simulation of the scenario, the top-right one",
        "is to help us understand the connection between dropping needles",
        "and the mathematical method to estimate pi, and the bottom one is",
        "the result for each dropping."))

ani.options(oopt)
```

CLELAL09 — *The NBA game between CLE Cavaliers and LAL Lakers on Dec 25, 2009*

Description

Cleveland Cavaliers played against Los Angeles Lakers at Staples Center in LA on Dec 25, 2009 and won the game by 102:87. This data recorded the locations of players on the court and the results of the shots.

Format

A data frame with 455 observations on the following 7 variables.

`player` a character vector: the current player

`time` a character vector: the time

`period` a numeric vector: the period (1 - 4)

`realx` a numeric vector: the x-axis location `realy` a numeric vector: the y-axis location

`result` a factor with levels `made` and `missed`

`team` a factor with levels `CLE`, `LAL` and `OFF`

Note

We view the court with CLE in the left and LAL in the right: `realx` is the distance to the left border of CLE's court, and `realy` is the distance to the bottom border of the court; notice that the size of the court is 94×50 (feet).

Source

http://www.basketballgeek.com/data/ (transformed based on the original data)

Examples

```
## see demo('CLEvsLAL', package = 'animation') for a `replay' of the game
```

`clt.ani` *Demonstration of the Central Limit Theorem*

Description

First of all, a number of obs observations are generated from a certain distribution for each variable X_j, $j = 1, 2, \cdots, n$, and $n = 1, 2, \cdots, nmax$, then the sample means are computed, and at last the density of these sample means is plotted as the sample size n increases (the theoretical limiting distribution is denoted by the dashed line), besides, the P-values from the normality test `shapiro.test` are computed for each n and plotted at the same time.

Usage

```
clt.ani(obs = 300, FUN = rexp, mean = 1, sd = 1, col = c("bisque", "red", "blue",
    "black"), mat = matrix(1:2, 2), widths = rep(1, ncol(mat)), heights = rep(1,
    nrow(mat)), xlim, ...)
```

Arguments

`obs`	the number of sample means to be generated from the distribution based on a given sample size n; these sample mean values will be used to create the histogram
`FUN`	the function to generate `n` random numbers from a certain distribution
`mean, sd`	the expectation and standard deviation of the population distribution (they will be used to plot the density curve of the theoretical Normal distribution with mean equal to `mean` and sd equal to $sd/\sqrt{n}$; if any of them is `NA`, the density curve will be suppressed)
`col`	a vector of length 4 specifying the colors of the histogram, the density curve of the sample mean, the theoretical density cuve and P-values.

mat, widths, heights
: arguments passed to layout to set the layout of the two graphs.

xlim
: the x-axis limit for the histogram (it has a default value if not specified)

...
: other arguments passed to plot.default to plot the P-values

Details

As long as the conditions of the Central Limit Theorem (CLT) are satisfied, the distribution of the sample mean will be approximate to the Normal distribution when the sample size n is large enough, no matter what is the original distribution. The largest sample size is defined by nmax in ani.options.

Value

A data frame of P-values.

Author(s)

Yihui Xie

References

http://vis.supstat.com/2013/04/central-limit-theorem

See Also

hist, density

Examples

```
oopt = ani.options(interval = 0.1, nmax = ifelse(interactive(), 150, 2))
op = par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
clt.ani(type = "s")
par(op)

## HTML animation page
saveHTML({
    par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
    ani.options(interval = 0.1, nmax = ifelse(interactive(), 150, 10))
    clt.ani(type = "h")
}, img.name = "clt.ani", htmlfile = "clt.ani.html", ani.height = 500,
    ani.width = 600, title = "Demonstration of the Central Limit Theorem",
    description = c("This animation shows the distribution of the sample",
        "mean as the sample size grows."))

## other distributions: Chi-square with df = 5 (mean = df, var = 2*df)
f = function(n) rchisq(n, 5)
clt.ani(FUN = f, mean = 5, sd = sqrt(2 * 5))

ani.options(oopt)
```

conf.int *Demonstration of the concept of confidence intervals*

Description

This function gives a demonstration of the concept of confidence intervals in mathematical statistics.

Usage

```
conf.int(level = 0.95, size = 50, cl = c("red", "gray"), ...)
```

Arguments

`level`	the confidence level $(1-\alpha)$, e.g. 0.95
`size`	the sample size for drawing samples from N(0, 1)
`cl`	two different colors to annotate whether the confidence intervals cover the true mean (`cl[1]`: no; `cl[2]`: yes)
`...`	other arguments passed to `plot.default`

Details

Keep on drawing samples from the Normal distribution N(0, 1), computing the intervals based on a given confidence level and plotting them as segments in a graph. In the end, we may check the coverage rate against the given confidence level.

Intervals that cover the true parameter are denoted in color `cl[2]`, otherwise in color `cl[1]`. Each time we draw a sample, we can compute the corresponding confidence interval. As the process of drawing samples goes on, there will be a legend indicating the numbers of the two kinds of intervals respectively and the coverage rate is also denoted in the top-left of the plot.

The argument `nmax` in `ani.options` controls the maximum times of drawing samples.

Value

A list containing

`level`	confidence level
`size`	sample size
`CI`	a matrix of confidence intervals for each sample
`CR`	coverage rate

Author(s)

Yihui Xie

References

George Casella and Roger L. Berger. *Statistical Inference*. Duxbury Press, 2th edition, 2001.

Examples

```
oopt = ani.options(interval = 0.1, nmax = ifelse(interactive(), 100, 2))
## 90% interval
conf.int(0.9, main = "Demonstration of Confidence Intervals")

## save the animation in HTML pages
saveHTML({
    ani.options(interval = 0.15, nmax = ifelse(interactive(), 100, 10))
    par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
    conf.int()
}, img.name = "conf.int", htmlfile = "conf.int.html", ani.height = 400,
    ani.width = 600, title = "Demonstration of Confidence Intervals",
    description = c("This animation shows the concept of the confidence",
        "interval which depends on the observations: if the samples change,",
        "the interval changes too. At last we can see that the coverage rate",
        "will be approximate to the confidence level."))

ani.options(oopt)
```

---

cv.ani *Demonstration for the process of cross-validation*

Description

This function uses rectangles to illustrate the k folds and mark the test set and the training set with different colors.

Usage

```
cv.ani(x = runif(150), k = 10, col = c("green", "red", "blue"), pch = c(4, 1), ...)
```

Arguments

x	a numerical vector which stands for the sample points.
k	an integer: how many parts should we split the data into? (comes from the k-fold cross-validation.)
col	a character vector of length 3 specifying the colors of: the rectangle representing the test set, the points of the test set, and points of the training set.
pch	a numeric vector of length 2 specifying the symbols of the test set and training set respectively.
...	other arguments passed to `plot.default`

Details

Briefly speaking, the process of cross-validation is just to split the whole data set into several parts and select one part as the test set and the rest parts as the training set.

The computation of sample sizes is base on `kfcv`.

Value

None (invisible NULL).

Note

For the 'leave-one-out' cross-validation, just specify k as length(x), then the rectangles will 'shrink' into single lines.

The final number of animation frames is the smaller one of ani.options('nmax') and k.

This function has nothing to do with specific models used in cross-validation.

Author(s)

Yihui Xie

See Also

kfcv

Examples

```
oopt = ani.options(interval = 2, nmax = 15)
cv.ani(main = "Demonstration of the k-fold Cross Validation", bty = "l")

## leave-one-out CV
cv.ani(x = runif(15), k = 15)

## save the animation in HTML pages
saveHTML({
    ani.options(interval = 2)
    par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
    cv.ani(bty = "l")
}, img.name = "cv.ani", htmlfile = "cv.ani.html", ani.height = 400,
    ani.width = 600, title = "Demonstration of the k-fold Cross Validation",
    description = c("This is a naive demonstration for the k-fold cross",
        "validation. The k rectangles in the plot denote the k folds of data.",
        "Each time a fold will be used as the test set and the rest parts",
        "as the training set."))

ani.options(oopt)
```

---

cv.nfeaturesLDA *Cross-validation to find the optimum number of features (variables) in LDA*

Description

This function provids an illustration of the process of finding out the optimum number of variables using k-fold cross-validation in a linear discriminant analysis (LDA).

Usage

```
cv.nfeaturesLDA(data = matrix(rnorm(600), 60), cl = gl(3, 20), k = 5, cex.rg = c(0.5,
    3), col.av = c("blue", "red"), ...)
```

Arguments

`data`	a data matrix containg the predictors in columns
`cl`	a factor indicating the classification of the rows of `data`
`k`	the number of folds
`cex.rg`	the range of the magnification to be used to the points in the plot
`col.av`	the two colors used to respectively denote rates of correct predictions in the i-th fold and the average rates for all k folds
`...`	arguments passed to `points` to draw the points which denote the correct rate

Details

For a classification problem, usually we wish to use as less variables as possible because of difficulties brought by the high dimension.

The selection procedure is like this:

- Split the whole data randomly into k folds:
  - For the number of features $g = 1, 2, \cdots, g_{max}$, choose g features that have the largest discriminatory power (measured by the F-statistic in ANOVA):
    - For the fold i ($i = 1, 2, \cdots, k$):
      - Train a LDA model without the i-th fold data, and predict with the i-th fold for a proportion of correct predictions p_{gi};
  - Average the k proportions to get the correct rate p_g;
- Determine the optimum number of features with the largest p.

Note that g_{max} is set by `ani.options('nmax')` (i.e. the maximum number of features we want to choose).

Value

A list containing

`accuracy`	a matrix in which the element in the i-th row and j-th column is the rate of correct predictions based on LDA, i.e. build a LDA model with j variables and predict with data in the i-th fold (the test set)
`optimum`	the optimum number of features based on the cross-validation

Author(s)

Yihui Xie <http://yihui.name>

References

Maindonald J, Braun J (2007). *Data Analysis and Graphics Using R - An Example-Based Approach*. Cambridge University Press, 2nd edition. pp. 400

See Also

kfcv, cv.ani, lda

Examples

```
oopt = ani.options(nmax = ifelse(interactive(), 10, 2))
par(mar = c(3, 3, 0.2, 0.7), mgp = c(1.5, 0.5, 0))
cv.nfeaturesLDA(pch = 19)

## save the animation in HTML pages
saveHTML({
    ani.options(interval = 0.5, nmax = 10)
    par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0),
        tcl = -0.3, pch = 19, cex = 1.5)
    cv.nfeaturesLDA(pch = 19)
}, img.name = "cv.nfeaturesLDA", htmlfile = "cv.nfeaturesLDA.html",
    ani.height = 480, ani.width = 600, description = c("This animation provides",
        " an illustration of the process of finding",
        "out the optimum number of variables using k-fold cross-validation",
        "in a linear discriminant analysis (LDA)."),
    title = "Cross-validation to find the optimum number of features in LDA")

ani.options(oopt)
```

---

ecol.death.sim — *A simulation of the death of two species with certain probabilities*

Description

Suppose there are two plant species in a field: A and B. One of them will die at each time and a new plant will grow in the place where the old plant died; the species of the new plant depends on the proportions of two species: the larger the proportion is, the greater the probability for this species to come up will be.

Usage

```
ecol.death.sim(nr = 10, nc = 10, num.sp = c(50, 50), col.sp = c(1, 2), pch.sp = c(1,
    2), col.die = 1, pch.die = 4, cex = 3, ...)
```

Arguments

`nr, nc`	number of rows and columns of the field (plants grow on a `nr` x `nc` grid)
`num.sp`	number of two plants respectively
`col.sp, pch.sp`	colors and point symbols of the two species respectively
`col.die, pch.die, cex`	the color, point symbol and magnification to annotate the plant which dies (symbol default to be an 'X')
`...`	other arguments passed to `plot` to set up the plot

Value

a vector (factor) containing 1's and 2's, denoting the plants finally survived

Note

`2 * ani.options('nmax')` image frames will actually be produced.

Author(s)

Yihui Xie

References

This animation is motivated by a question raised from Jing Jiao, a student in biology, to show the evolution of two species.

The original post is in the forum of the "Capital of Statistics": http://cos.name/cn/topic/14093 (in Chinese)

Examples

```
oopt = ani.options(nmax = ifelse(interactive(), 50, 2), interval = 0.3)
par(ann = FALSE, mar = rep(0, 4))
ecol.death.sim()

## large scale simulation
ani.options(nmax = ifelse(interactive(), 1000, 2), interval = 0.02)
ecol.death.sim(col.sp = c(8, 2), pch.sp = c(20, 17))

ani.options(oopt)
```

flip.coin *Probability in flipping coins*

Description

This function provides a simulation to the process of flipping coins and computes the frequencies for 'heads' and 'tails'.

Usage

```
flip.coin(faces = 2, prob = NULL, border = "white", grid = "white", col = 1:2,
    type = "p", pch = 21, bg = "transparent", digits = 3)
```

Arguments

`faces`	an integer or a character vector. See details below.
`prob`	the probability vector of showing each face. If `NULL`, each face will be shown in the same probability.
`border`	The border style for the rectangles which stand for probabilities.
`grid`	the color for horizontal grid lines in these rectangles
`col`	The colors to annotate different faces of the 'coin'.
`type, pch, bg`	See `points`.
`digits`	integer indicating the precision to be used in the annotation of frequencies in the plot

Details

If `faces` is a single integer, say 2, a sequence of integers from 1 to `faces` will be used to denote the faces of a coin; otherwise this character vector just gives the names of each face.

When the i-th face shows up, a colored thin rectangle will be added to the corresponding place (the i-th bar), and there will be corresponding annotations for the number of tosses and frequencies.

The special argument `grid` is for consideration of a too large number of flipping, in which case if you still draw horizontal lines in these rectangles, the rectangles will be completely covered by these lines, thus we should specify it as `NA`.

At last the frequency for each face will be computed and shown in the header of the plot – this shall be close to `prob` if `ani.options('nmax')` is large enough.

Value

A list containing

`freq`	A vector of frequencies (simulated probabilities)
`nmax`	the total number of tosses

Note

You may change the colors of each face using the argument col (repeated if shorter than the number of faces).

Author(s)

Yihui Xie

References

http://vis.supstat.com/2013/03/simulation-of-coin-flipping

See Also

points, sample

Examples

```
oopt = ani.options(interval = 0.2, nmax = ifelse(interactive(),
    100, 2))
## a coin would stand on the table?? just kidding :)
flip.coin(faces = c("Head", "Stand", "Tail"), type = "n", prob = c(0.45,
    0.1, 0.45), col = c(1, 2, 4))

flip.coin(bg = "yellow")

## HTML animation page
saveHTML({
    ani.options(interval = 0.2, nmax = ifelse(interactive(), 100,
        2))
    par(mar = c(2, 3, 2, 1.5), mgp = c(1.5, 0.5, 0))
    flip.coin(faces = c("Head", "Stand", "Tail"), type = "n",
        prob = c(0.45, 0.1, 0.45), col = c(1, 2, 4))
}, img.name = "flip.coin", htmlfile = "flip.coin.html", ani.height = 500,
    ani.width = 600, title = "Probability in flipping coins",
    description = c("This animation has provided a simulation of flipping coins",
        "which might be helpful in understanding the concept of probability."))

ani.options(oopt)
```

---

g.brownian.motion *Brownian Motion using Google Visualization API*

Description

We can use R to generate random numbers from the Normal distribution and write them into an HTML document, then the Google Visualization gadget "motionchart" will prepare the animation for us (a Flash animation with several buttons).

Usage

```
g.brownian.motion(p = 20, start = 1900, digits = 14, file = "index.html", width = 800,
    height = 600)
```

Arguments

p	number of points
start	start “year”; it has no practical meaning in this animation but it’s the required by the Google gadget
digits	the precision to round the numbers
file	the HTML filename
width, height	width and height of the animation

Value

NULL. An HTML page will be opened as the side effect.

Note

The number of frames is controlled by ani.options('nmax') as usual.

Due to the “security settings” of Adobe Flash player, you might not be able to view the generated Flash animation locally, i.e. using an address like ‘file:///C:/Temp/index.html’. In this case, you can upload the HTML file to a web server and use the http address to view the Flash file.

Author(s)

Yihui Xie

References

http://code.google.com/apis/visualization/ and http://bit.ly/12w1sYi

See Also

brownian.motion, BM.circle, rnorm

Examples

```
g.brownian.motion(15, digits = 2, width = 600, height = 500,
    file = "BM-motion-chart.html")
```

grad.desc *Gradient Descent Algorithm for the 2D case*

Description

This function provids a visual illustration for the process of minimizing a real-valued function through Gradient Descent Algorithm.

Usage

```
grad.desc(FUN = function(x, y) x^2 + 2 * y^2, rg = c(-3, -3, 3, 3), init = c(-3,
    3), gamma = 0.05, tol = 0.001, gr = NULL, len = 50, interact = FALSE,
    col.contour = "red", col.arrow = "blue", main)
```

Arguments

`FUN` a bivariate objective function to be minimized (variable names do not have to be `x` and `y`); if the gradient argument `gr` is `NULL`, `deriv` will be used to calculate the gradient, in which case we should not put braces around the function body of `FUN` (e.g. the default function is `function(x, y) x^2 + 2 * y^2`)

`rg` ranges for independent variables to plot contours; in a `c(x0,y0, x1, y1)` form

`init` starting values

`gamma` size of a step

`tol` tolerance to stop the iterations, i.e. the minimum difference between $F(x_i)$ and $F(x_{i+1})$

`gr` the gradient of `FUN`; it should be a bivariate function to calculate the gradient (not the negative gradient!) of `FUN` at a point (x, y), e.g. `function(x, y) 2 * x + 4 * y`. If it is `NULL`, R will use `deriv` to calculate the gradient

`len` desired length of the independent sequences (to compute z values for contours)

`interact` logical; whether choose the starting values by clicking on the contour plot directly?

`col.contour, col.arrow` colors for the contour lines and arrows respectively (default to be red and blue)

`main` the title of the plot; if missing, it will be derived from `FUN`

Details

Gradient descent is an optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or the approximate gradient) of the function at the current point. If instead one takes steps proportional to the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.

The arrows are indicating the result of iterations and the process of minimization; they will go to a local minimum in the end if the maximum number of iterations `ani.options('nmax')` has not been reached.

Value

A list containing

- `par` — the solution for the local minimum
- `value` — the value of the objective function corresponding to `par`
- `iter` — the number of iterations; if it is equal to `ani.options('nmax')`, it's quite likely that the solution is not reliable because the maximum number of iterations has been reached
- `gradient` — the gradient function of the objective function
- `persp` — a function to make the perspective plot of the objective function; can accept further arguments from `persp` (see the examples below)

Note

Please make sure the function `FUN` provided is differentiable at `init`, what's more, it should also be 'differentiable' using `deriv` if you do not provide the gradient function `gr`.

If the arrows cannot reach the local minimum, the maximum number of iterations `nmax` in `ani.options` may need to be increased.

Author(s)

Yihui Xie

References

http://vis.supstat.com/2013/03/gradient-descent-algorithm-with-r/

See Also

`deriv`, `persp`, `contour`, `optim`

Examples

```
## default example
oopt = ani.options(interval = 0.3, nmax = ifelse(interactive(), 50, 2))
xx = grad.desc()
xx$par  # solution
xx$persp(col = "lightblue", phi = 30)  # perspective plot

## define more complex functions; a little time-consuming
f1 = function(x, y) x^2 + 3 * sin(y)
xx = grad.desc(f1, pi * c(-2, -2, 2, 2), c(-2 * pi, 2))
xx$persp(col = "lightblue", theta = 30, phi = 30)

## need to provide the gradient when deriv() cannot handle the function
grad.desc(FUN = function(x1, x2) {
    x0 = cos(x2)
    x1^2 + x0
}, gr = function(x1, x2) {
```

```
    c(2 * x1, -sin(x2))
}, rg = c(-3, -1, 3, 5), init = c(-3, 0.5), main = expression(x[1]^2 + cos(x[2])))

## or a even more complicated function
ani.options(interval = 0, nmax = ifelse(interactive(), 200, 2))
f2 = function(x, y) sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2 * x + 1 - exp(y))
xx = grad.desc(f2, c(-2, -2, 2, 2), c(-1, 0.5), gamma = 0.1, tol = 1e-04)

## click your mouse to select a start point
if (interactive()) {
    xx = grad.desc(f2, c(-2, -2, 2, 2), interact = TRUE, tol = 1e-04)
    xx$persp(col = "lightblue", theta = 30, phi = 30)
}

## HTML animation pages
saveHTML({
    ani.options(interval = 0.3)
    grad.desc()
}, img.name = "grad.desc", htmlfile = "grad.desc.html", ani.height = 500,
    ani.width = 500, title = "Demonstration of the Gradient Descent Algorithm",
    description = "The arrows will take you to the optimum step by step.")

ani.options(oopt)
```

HuSpeech *Word counts of a speech by the Chinese President Hu*

Description

This speech came on the 30th anniversary of China's economic reform in 1978.

Format

int [1:75] 119 175 222 204 276 168 257 89 61 288 ...

Details

On Dec 18, 2008, Chinese President Hu gave a speech on the 30th anniversary of China's economic reform in 1978, and this data has recorded the number of words used in each paragraph of his speech.

Source

The full text of speech is at http://cpc.people.com.cn/GB/64093/64094/8544901.html

Examples

```
## clear pattern: 1/3 short, 1/3 long, 1/3 short again
plot(HuSpeech, type = "b", pch = 20, xlab = "paragraph index", ylab = "word count")
## see ?moving.block for an animation example
```

iatemp	*Average yearly temperatures in central Iowa*

Description

Temperatures in central Iowa over 106 years.

Format

Time-Series [1:116] from 1895 to 2010: 32.7 27.8 32.7 30.4 42.6 31.9 34.5 39.8 32.6 39.6 ...

Source

http://www.wrcc.dri.edu/cgi-bin/divplot1_form.pl?1305

Examples

```
plot(iatemp)
```

im.convert	*A wrapper for the 'convert' utility of ImageMagick or GraphicsMagick*

Description

The main purpose of these two functions is to create GIF animations.

Usage

```
im.convert(files, output = "animation.gif", convert = c("convert", "gm convert"),
   cmd.fun = if (.Platform$OS.type == "windows") shell else system, extra.opts = "",
    clean = FALSE)

gm.convert(..., convert = "gm convert")
```

Arguments

files	either a character vector of file names, or a single string containing wildcards (e.g. 'Rplot*.png')
output	the file name of the output (with proper extensions, e.g. gif)
convert	the convert command; it must be either 'convert' or 'gm convert'; and it can be pre-specified as an option in ani.options('convert'), e.g. (Windows users) ani.options(convert = 'c:/program files/imagemagick/convert.exe'), or (Mac users) ani.options(convert = '/opt/local/bin/convert'); see the Note section for more details
cmd.fun	a function to invoke the OS command; by default system

extra.opts	additional options to be passed to convert (or gm convert)
clean	logical: delete the input files or not
...	arguments to be passed to im.convert

Details

The function im.convert simply wraps the arguments of the convert utility of ImageMagick to make it easier to call ImageMagick in R;

The function gm.convert is a wrapper for the command gm convert of GraphicsMagick.

Value

The command for the conversion.

If ani.options('autobrowse') == TRUE, this function will also try to open the output automatically.

Note

If files is a character vector, please make sure the order of filenames is correct! The first animation frame will be files[1], the second frame will be files[2], ...

Both ImageMagick and GraphicsMagick may have a limit on the number of images to be converted. It is a known issue that this function can fail with more than (approximately) 9000 images. The function saveVideo is a better alternative in such a case.

Most Windows users do not have read the boring notes below after they have installed ImageMagick or GraphicsMagick. For the rest of Windows users:

ImageMagick users Please install ImageMagick from http://www.imagemagick.org, and make sure the the path to convert.exe is in your 'PATH' variable, in which case the command convert can be called without the full path. Windows users are often very confused about the ImageMagick and 'PATH' setting, so I'll try to search for ImageMagick in the Registry Hive by readRegistry('SOFTWARE\ImageMagick\Current')$BinPath, thus you might not really need to modify your 'PATH' variable.

For Windows users who have installed LyX, I will also try to find the convert utility in the LyX installation directory, so they do not really have to install ImageMagick if LyX exists in their system (of course, the LyX should be installed with ImageMagick).

Once the convert utility is found, the animation option 'convert' will be set (ani.options(convert = 'path/to/convert.exe')); this can save time for searching for convert in the operating system next time.

GraphicsMagick users During the installation of GraphicsMagick, you will be asked if you allow it to change the PATH variable; please do check the option.

A reported problem is cmd.fun = shell might not work under Windows but cmd.fun = system works fine. Try this option in case of failures.

Author(s)

Yihui Xie

References

ImageMagick: http://www.imagemagick.org/script/convert.php GraphicsMagick: http://www.graphicsmagick.org

See Also

Other utilities: saveGIF, saveMovie; saveHTML; saveLatex; saveSWF; saveVideo

Examples

```
## generate some images
owd = setwd(tempdir())
oopt = ani.options(interval = 0.05, nmax = 20)
png("bm%03d.png")
brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow",
    main = "Demonstration of Brownian Motion")
dev.off()

## filenames with a wildcard *
im.convert("bm*.png", output = "bm-animation1.gif")
## use GraphicsMagick
gm.convert("bm*.png", output = "bm-animation2.gif")

## or a filename vector
bm.files = sprintf("bm%03d.png", 1:20)
im.convert(files = bm.files, output = "bm-animation3.gif")

ani.options(oopt)
setwd(owd)
```

---

kfcv — *Sample sizes for k-fold cross-validation*

Description

Compute sample sizes for k-fold cross-validation.

Usage

```
kfcv(k, N)
```

Arguments

k	number of groups.
N	total sample size.

Details

If N/k is an integer, the sample sizes are k 'N/k's (N/k, N/k, ...), otherwise the remainder will be allocated to each group as 'uniformly' as possible, and at last these sample sizes will be permuted randomly.

Value

A vector of length k containing k sample sizes.

Author(s)

Yihui Xie

See Also

cv.ani

Examples

```
## divisible
kfcv(5, 25)

## not divisible
kfcv(10, 77)
```

---

kmeans.ani *Demonstration of the k-Means clustering algorithm*

Description

This function provides a demo of the k-Means cluster algorithm for data containing only two variables (columns).

Usage

```
kmeans.ani(x = cbind(X1 = runif(50), X2 = runif(50)), centers = 3,
    hints = c("Move centers!", "Find cluster?"), pch = 1:3, col = 1:3)
```

Arguments

x	A numercal matrix or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns) containing *only* 2 columns.
centers	Either the number of clusters or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in x is chosen as the initial centres.
hints	Two text strings indicating the steps of k-means clustering: move the center or find the cluster membership?
pch, col	Symbols and colors for different clusters; the length of these two arguments should be equal to the number of clusters, or they will be recycled.

Details

The k-Means cluster algorithm may be regarded as a series of iterations of: finding cluster centers, computing distances between sample points, and redefining cluster membership.

The data given by x is clustered by the k-means method, which aims to partition the points into k groups such that the sum of squares from points to the assigned cluster centers is minimized. At the minimum, all cluster centres are at the mean of their Voronoi sets (the set of data points which are nearest to the cluster centre).

Value

A list with components

cluster	A vector of integers indicating the cluster to which each point is allocated.
centers	A matrix of cluster centers.

Note

This function is only for demonstration purpose. For practical applications please refer to kmeans.

Note that ani.options('nmax') is defined as the maximum number of iterations in such a sense: an iteration includes the process of computing distances, redefining membership and finding centers. Thus there should be 2 * ani.options('nmax') animation frames in the output if the other condition for stopping the iteration has not yet been met (i.e. the cluster membership will not change any longer).

Author(s)

Yihui Xie

See Also

kmeans

Examples

```
## set larger 'interval' if the speed is too fast
oopt = ani.options(interval = 2)
par(mar = c(3, 3, 1, 1.5), mgp = c(1.5, 0.5, 0))
kmeans.ani()

## the kmeans() example; very fast to converge!
x = rbind(matrix(rnorm(100, sd = 0.3), ncol = 2), matrix(rnorm(100, mean = 1,
    sd = 0.3), ncol = 2))
colnames(x) = c("x", "y")
kmeans.ani(x, centers = 2)

## what if we cluster them into 3 groups?
kmeans.ani(x, centers = 3)

## create an HTML animation page
saveHTML({
```

```
    ani.options(interval = 2)
    par(mar = c(3, 3, 1, 1.5), mgp = c(1.5, 0.5, 0))

    cent = 1.5 * c(1, 1, -1, -1, 1, -1, 1, -1)
    x = NULL
    for (i in 1:8) x = c(x, rnorm(25, mean = cent[i]))
    x = matrix(x, ncol = 2)
    colnames(x) = c("X1", "X2")

    kmeans.ani(x, centers = 4, pch = 1:4, col = 1:4)

}, img.name = "kmeans.ani", htmlfile = "kmeans.ani.html", ani.height = 480,
    ani.width = 480, title = "Demonstration of the K-means Cluster Algorithm",
    description = "Move! Average! Cluster! Move! Average! Cluster! ...")

ani.options(oopt)
```

---

knn.ani *Demonstration of the k-Nearest Neighbour classification*

Description

Demonstrate the process of k-Nearest Neighbour classification on the 2D plane.

Usage

```
knn.ani(train, test, cl, k = 10, interact = FALSE, tt.col = c("blue", "red"),
   cl.pch = seq_along(unique(cl)), dist.lty = 2, dist.col = "gray", knn.col = "green",
    ...)
```

Arguments

`train`	matrix or data frame of training set cases containing only 2 columns
`test`	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case. It should also contain only 2 columns. This data set will be *ignored* if `interact = TRUE`; see `interact` below.
`cl`	factor of true classifications of training set
`k`	number of neighbours considered.
`interact`	logical. If `TRUE`, the user will have to choose a test set for himself using mouse click on the screen; otherwise compute kNN classification based on argument `test`.
`tt.col`	a vector of length 2 specifying the colors for the training data and test data.
`cl.pch`	a vector specifying symbols for each class
`dist.lty, dist.col`	the line type and color to annotate the distances
`knn.col`	the color to annotate the k-nearest neighbour points using a polygon
`...`	additional arguments to create the empty frame for the animation (passed to `plot.default`)

Details

For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. For a single test sample point, the basic steps are:

1. locate the test point
2. compute the distances between the test point and all points in the training set
3. find k shortest distances and the corresponding training set points
4. vote for the result (find the maximum in the table for the true classifications)

As there are four steps in an iteration, the total number of animation frames should be `4 * min(nrow(test), ani.options('` at last.

Value

A vector of class labels for the test set.

Note

There is a special restriction (only two columns) on the training and test data set just for sake of the convenience for making a scatterplot. This is only a rough demonstration; for practical applications, please refer to existing kNN functions such as `knn` in **class**, etc.

If either one of `train` and `test` is missing, there'll be random matrices prepared for them. (It's the same for `cl`.)

Author(s)

Yihui Xie

References

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth edition. Springer.

See Also

`knn`

Examples

```
## a binary classification problem
oopt = ani.options(interval = 2, nmax = ifelse(interactive(), 10, 2))
x = matrix(c(rnorm(80, mean = -1), rnorm(80, mean = 1)), ncol = 2, byrow = TRUE)
y = matrix(rnorm(20, mean = 0, sd = 1.2), ncol = 2)
knn.ani(train = x, test = y, cl = rep(c("first class", "second class"),
    each = 40), k = 30)

x = matrix(c(rnorm(30, mean = -2), rnorm(30, mean = 2), rnorm(30, mean = 0)),
    ncol = 2, byrow = TRUE)
y = matrix(rnorm(20, sd = 2), ncol = 2)
```

```
knn.ani(train = x, test = y, cl = rep(c("first", "second", "third"),
    each = 15), k = 25, cl.pch = c(2, 3, 19), dist.lty = 3)

## an interactive demo: choose the test set by mouse-clicking
if (interactive()) {
    ani.options(nmax = 5)
    knn.ani(interact = TRUE)
}

## HTML page
saveHTML({
    ani.options(nmax = ifelse(interactive(), 10, 2), interval = 2)
    par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0))
    knn.ani(cl.pch = c(3, 19), asp = 1)
}, img.name = "knn_ani", htmlfile = "knn.ani.html", ani.height = 500,
    ani.width = 600, title = "Demonstration for kNN Classification",
    description = c("For each row of the test set", "the k nearest (in Euclidean",
        "distance) training set vectors are found, and the classification is",
        "decided by majority vote, with ties broken at random."))

ani.options(oopt)
```

---

least.squares — *Demonstrate the least squares method*

Description

This is a simple demonstration of the meaning of least squares in univariate linear regression.

Usage

```
least.squares(x, y, n = 15, ani.type = c("slope", "intercept"), a, b, a.range, b.range,
   ab.col = c("gray", "black"), est.pch = 19, v.col = "red", v.lty = 2, rss.pch = 19,
    rss.type = "o", mfrow = c(1, 2), ...)
```

Arguments

`x`	a numeric vector: the independent variable
`y`	a numeric vector: the dependent variable
`n`	the sample size: when x and y are missing, we use simulated values of y (`x = 1:n` and `y = a + b * x + rnorm(n)`)
`ani.type`	`'slope'`: the slope is changing with the intercept fixed; `'intercept'`: intercept changing and slope fixed
`a, b`	the fixed intercept and slope; depending on `ani.type`, we only need to specify one of them; e.g. when `ani.type == 'slope'`, we need to specify the value of `a`

a.range, b.range
: a vector of length 2 to define the range of the intercept and the slope; only one of them need to be specified; see above

ab.col
: the colors of two lines: the real regression line and the moving line with either intercept or slope changing

est.pch
: the point character of the 'estimated' values given x

v.col, v.lty
: the color and line type of the vetical lines which demonstrate the residuals

rss.pch, rss.type
: the point character and plot type of the residual plot

mfrow
: defines the layout of the graph; see par

...
: other parameters passed to plot to define the appearance of the scatterplot

Details

With either the intercept or the slope changing, the lines will be moving in the graph and corresponding residuals will be plotted. We can finally see the best estimate of the intercept and the slope from the residual plot.

Value

The value returned depends on the animation type.

If it is a slope animation, the value will be a list containing

lmfit
: the estimates of the intercept and slope with lm

anifit
: the estimate of the slope in the animation

If it is an intercept animation, the second component of the above list will be the estimate of the intercept.

Note the estimate will not be precise generally.

Note

ani.options('nmax') specifies the maximum number of steps for the slope or intercept to move.

Author(s)

Yihui Xie

See Also

lm

Examples

```
par(mar = c(5, 4, 0.5, 0.1))
oopt = ani.options(interval = 0.3, nmax = ifelse(interactive(),
    50, 2))

## default animation: with slope changing
least.squares()

## intercept changing
least.squares(ani.type = "intercept")

## save the animation in HTML pages
saveHTML({
    ani.options(interval = 0.3, nmax = ifelse(interactive(), 50,
        2))
    par(mar = c(4, 4, 0.5, 0.1), mgp = c(2, 0.5, 0), tcl = -0.3)
    least.squares()
}, img.name = "least.squares", htmlfile = "least.squares.html",
    ani.height = 450, ani.width = 600, title = "Demonstration of Least Squares",
    description = c("We want to find an estimate for the slope",
        "in 50 candidate slopes, so we just compute the RSS one by one. "))

ani.options(oopt)
```

`lln.ani` *Demonstration of Law of Large Numbers*

Description

This function plots the sample mean as the sample size grows to check whether the sample mean approaches to the population mean.

Usage

```
lln.ani(FUN = rnorm, mu = 0, np = 30, pch = 20, col.poly = "bisque", col.mu = "gray",
    ...)
```

Arguments

FUN	a function to generate random numbers from a certain distribution: `function(n, mu)`
mu	population mean; passed to FUN
np	times for sampling from a distribution (not the sample size!); to examine the behaviour of the sample mean, we need more times of sampling to get a series of mean values
pch	symbols for points; see Details
col.poly	the color of the polygon to annotate the range of sample means
col.mu	the color of the horizontal line which denotes the population mean
...	other arguments passed to points

Details

np points are plotted to denote the distribution of the sample mean; we will observe that the range of the sample mean just becomes smaller and smaller as the sample size increases and ultimately there will be an obvious trend that the sample mean converges to the population mean mu.

The parameter nmax in ani.options means the maximum sample size.

Value

None (invisible NULL).

Note

The argument pch will influence the speed of plotting, and for a very large sample size (say, 300), it is suggested that this argument be specified as '.'.

Author(s)

Yihui Xie

References

http://vis.supstat.com/2013/04/law-of-large-numbers/

Examples

```
oopt = ani.options(interval = 0.01, nmax = ifelse(interactive(), 150,
    2))

lln.ani(pch = ".")

## chi-square distribution; population mean = df
lln.ani(FUN = function(n, mu) rchisq(n, df = mu), mu = 5, cex = 0.6)

## save the animation in HTML pages
saveHTML({
    par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0))
    ani.options(interval = 0.1, nmax = ifelse(interactive(), 150, 2))
    lln.ani(cex = 0.6)
}, img.name = "lln.ani", htmlfile = "lln.ani.html", ani.height = 480,
    ani.width = 600, title = "Demonstration of the Law of Large Numbers",
    description = c("The sample mean approaches to the population mean as",
        "the sample size n grows."))

ani.options(oopt)
```

`MC.hitormiss` *Hit or Miss Monte Carlo integration*

Description

Integrate a function using the Hit-or-Miss Monte Carlo algorithm.

Usage

```
MC.hitormiss(FUN = function(x) x - x^2, n = ani.options("nmax"), from = 0, to = 1,
    col.points = c("black", "red"), pch.points = c(20, 4), ...)
```

Arguments

`FUN`	the function to be integrated
`n`	number of points to be sampled from the Uniform(0, 1) distribution
`from, to`	the limits of integration
`col.points, pch.points`	colors and point characters for points which "hit" or "miss" the area under the curve
`...`	other arguments passed to `points`

Details

We compute the proportion of points hitting the area under the curve, and the integral can be estimated by the proportion multiplied by the total area of the rectangle (from xmin to xmax, ymin to ymax).

Value

A list containing

`x1`	the Uniform random numbers generated on x-axis
`x2`	the Uniform random numbers generated on y-axis
`y`	function values evaluated at `x1`
`n`	number of points drawn from the Uniform distribtion
`est`	the estimated value of the integral

Note

This function is for demonstration purpose only; the integral might be very inaccurate when `n` is small.

`ani.options('nmax')` specifies the maximum number of trials.

Author(s)

Yihui Xie

See Also

integrate, MC.samplemean

Examples

```
oopt = ani.options(interval = 0.2, nmax = ifelse(interactive(),
    100, 2))

## should be close to 1/6
MC.hitormiss()$est

## should be close to 1/12
MC.hitormiss(from = 0.5, to = 1)$est

## HTML animation page
saveHTML({
    ani.options(interval = 0.5, nmax = ifelse(interactive(), 100,
        2))
    MC.hitormiss()
}, img.name = "MC.hitormiss", htmlfile = "MC.hitormiss.html",
    title = "Hit or Miss Monte Carlo Integration", description = c("",
        "Generate Uniform random numbers", "and compute the proportion",
        "of points under the curve."))

ani.options(oopt)
```

MC.samplemean *Sample Mean Monte Carlo integration*

Description

Integrate a function from 0 to 1 using the Sample Mean Monte Carlo algorithm

Usage

```
MC.samplemean(FUN = function(x) x - x^2, n = ani.options("nmax"), col.rect = c("gray",
    "black"), adj.x = TRUE, ...)
```

Arguments

FUN	the function to be integrated
n	number of points to be sampled from the Uniform(0, 1) distribution
col.rect	colors of rectangles (for the past rectangles and the current one)

`adj.x`	should the locations of rectangles on the x-axis be adjusted? If `TRUE`, the rectangles will be laid side by side and it is informative for us to assess the total area of the rectangles, otherwise the rectangles will be laid at their exact locations.
`...`	other arguments passed to `rect`

Details

Sample Mean Monte Carlo integration can compute

$$I = \int_0^1 f(x)dx$$

by drawing random numbers x_i from Uniform(0, 1) distribution and average the values of $f(x_i)$. As n goes to infinity, the sample mean will approach to the expectation of $f(X)$ by Law of Large Numbers.

The height of the i-th rectangle in the animation is $f(x_i)$ and the width is $1/n$, so the total area of all the rectangles is $\sum f(x_i)1/n$, which is just the sample mean. We can compare the area of rectangles to the curve to see how close is the area to the real integral.

Value

A list containing

`x`	the Uniform random numbers
`y`	function values evaluated at `x`
`n`	number of points drawn from the Uniform distribtion
`est`	the estimated value of the integral

Note

This function is for demonstration purpose only; the integral might be very inaccurate when `n` is small.

`ani.options('nmax')` specifies the maximum number of trials.

Author(s)

Yihui Xie

See Also

`integrate`, `MC.hitormiss`

Examples

```
oopt = ani.options(interval = 0.2, nmax = ifelse(interactive(),
    50, 2))
par(mar = c(4, 4, 1, 1))

## when the number of rectangles is large, use border = NA
MC.samplemean(border = NA)$est

integrate(function(x) x - x^2, 0, 1)

## when adj.x = FALSE, use semi-transparent colors
MC.samplemean(adj.x = FALSE, col.rect = c(rgb(0, 0, 0, 0.3), rgb(1,
    0, 0)), border = NA)

## another function to be integrated
MC.samplemean(FUN = function(x) x^3 - 0.5^3, border = NA)$est

integrate(function(x) x^3 - 0.5^3, 0, 1)

## HTML animation page
saveHTML({
    ani.options(interval = 0.3, nmax = ifelse(interactive(), 50,
        2))
    MC.samplemean(n = 100, border = NA)
}, img.name = "MC.samplemean", htmlfile = "MC.samplemean.html",
    title = "Sample Mean Monte Carlo Integration", description = c("",
        "Generate Uniform random numbers", " and compute the average",
        "function values."))

ani.options(oopt)
```

---

moving.block *Cycle through an R object and plot each subset of elements*

Description

For a long numeric vector or matrix (or data frame), we can plot only a subset of its elements to take a closer look at its structure. With a moving "block" from the beginning to the end of a vector or matrix or any R objects to which we can apply `subset`, all elements inside the block are plotted as a line or scatter plot or any customized plots.

Usage

```
moving.block(dat = runif(100), block, FUN, ...)
```

Arguments

`dat`	a numeric vector or two-column matrix
`block`	block length (i.e. how many elements are to be plotted in each step)

FUN	a plot function to be applied to the subset of data
...	other arguments passed to FUN

Details

For a vector, the elments from `i + 1` to `i + block` will be plotted in the i-th step; similarly for a matrix or data frame, a (scatter) plot will be created from the `i + 1`-th row to `i + block`-th row.

However, this function is not limited to scatter plots or lines – we can customize the function FUN as we wish.

Value

NULL

Note

There will be `ani.options('nmax')` image frames created in the end. Ideally the relationship between `ani.options('nmax')` and `block` should follow this equality: `block = length(x) - ani.options('nmax') + 1` (replace `length(x)` with `nrow(x)` when `x` is a matrix). The function will compute `block` according to the equality by default if no block length is specified.

The three arguments `dat`, `i` and `block` are passed to FUN in case we want to customize the plotting function, e.g. we may want to annonate the x-axis label with `i`, or we want to compute the mean value of `dat[i + 1:block]`, etc. See the examples below to learn more about how to make use of these three arguments.

Author(s)

Yihui Xie

Examples

```
## (1) Brownian motion block length: 101 (i.e. 300-200+1)
oopt = ani.options(nmax = ifelse(interactive(), 200, 2), interval = 0.1)
# plot y = dat against x = i + 1:block customize xlab and ylab with
# 'i' and 'block' restrict ylim using the range of 'dat'
moving.block(dat = cumsum(rnorm(300)), FUN = function(..., dat = dat,
    i = i, block = block) {
    plot(..., x = i + 1:block, xlab = sprintf("block length = %d", block),
        ylim = range(dat), ylab = sprintf("x[%s:%s]", i + 1, i + block))
}, type = "o", pch = 20)


## (2) Word counts of Hu's speech (block = 10; length(HuSpeech) = 75)
## see any pattern in the President's speech?
ani.options(nmax = ifelse(interactive(), 66, 2), interval = 0.5)
moving.block(dat = HuSpeech, FUN = function(..., dat = dat, i = i, block = block) {
    plot(..., x = i + 1:block, xlab = "paragraph index", ylim = range(dat),
        ylab = sprintf("HuSpeech[%s:%s]", i + 1, i + block))
}, type = "o", pch = 20)
```

```
## (3) sunspot data: observe the 11-year cycles block = 11 years x 12
## months/year = 132 set interval greater than 0 if your computer
## really rocks!
ani.options(nmax = ifelse(interactive(), 2857, 2), interval = 0)
spt.att = tsp(sunspot.month)
# the time index (we need it to correctly draw the ticks of x-axis)
ts.idx = seq(spt.att[1], spt.att[2], 1/spt.att[3])
moving.block(dat = sunspot.month, block = 132, FUN = function(..., dat = dat,
    i = i, block = block) {
    plot(..., x = ts.idx[i + 1:block], xlab = sprintf("block length = %d",
        block), ylim = range(dat), ylab = sprintf("sunspot.month[%s:%s]",
        i + 1, i + block))
}, type = "o", pch = 20)


## (4) earth quake: order the data by 'depth' first see how the
## locations change as 'depth' increases
ani.options(nmax = ifelse(interactive(), 900, 2), interval = 0.01)
# compute the mean depth for each block of data
moving.block(quakes[order(quakes$depth), c("long", "lat")], FUN = function(...,
    dat = dat, i = i, block = block) {
    plot(..., xlab = sprintf("%s[%s:%s]", colnames(dat)[1], i + 1, i +
        block), ylab = sprintf("%s[%s:%s]", colnames(dat)[2], i + 1,
        i + block), xlim = range(dat[, 1]), ylim = range(dat[, 2]),
        main = sprintf("Mean Depth = %.3f", mean(sort(quakes$depth)[i +
            1:block])))
}, pch = 20, col = rgb(0, 0, 0, 0.5))

ani.options(oopt)
```

`mwar.ani` *Demonstration for "Moving Window Auto-Regression"*

Description

This function just fulfills a very naive idea about moving window regression using rectangles to denote the "windows" and move them, and the corresponding AR(1) coefficients as long as rough confidence intervals are computed for data points inside the "windows" during the process of moving.

Usage

```
mwar.ani(x, k = 15, conf = 2, mat = matrix(1:2, 2), widths = rep(1, ncol(mat)),
    heights = rep(1, nrow(mat)), lty.rect = 2, ...)
```

Arguments

x	univariate time-series (a single numerical vector); default to be sin(seq(0, 2 * pi, length = 50)) + r
k	an integer of the window width
conf	a positive number: the confidence intervals are computed as c(ar1 - conf*s.e., ar1 + conf*s.e.)
mat, widths, heights	arguments passed to layout to divide the device into 2 parts
lty.rect	the line type of the rectangles respresenting the moving "windows"
...	other arguments passed to points in the bottom plot (the centers of the arrows)

Details

The AR(1) coefficients are computed by arima.

Value

A list containing

phi	the AR(1) coefficients
L	lower bound of the confidence interval
U	upper bound of the confidence interval

Author(s)

Yihui Xie

References

Robert A. Meyer, Jr. Estimating coefficients that change over time. *International Economic Review*, 13(3):705-710, 1972.

See Also

arima

Examples

```
## moving window along a sin curve
oopt = ani.options(interval = 0.1, nmax = ifelse(interactive(), 50, 2))
par(mar = c(2, 3, 1, 0.5), mgp = c(1.5, 0.5, 0))
mwar.ani(lty.rect = 3, pch = 21, col = "red", bg = "yellow", type = "o")

## for the data 'pageview'
mwar.ani(pageview$visits, k = 30)

## HTML animation page
saveHTML({
    ani.options(interval = 0.1, nmax = ifelse(interactive(), 50, 2))
    par(mar = c(2, 3, 1, 0.5), mgp = c(1.5, 0.5, 0))
    mwar.ani(lty.rect = 3, pch = 21, col = "red", bg = "yellow", type = "o")
```

```
}, img.name = "mwar.ani", htmlfile = "mwar.ani.html", ani.height = 500,
    ani.width = 600, title = "Demonstration of Moving Window Auto-Regression",
    description = c("Compute the AR(1) coefficient for the data in the",
        "window and plot the confidence intervals. Repeat this step as the",
        "window moves."))

ani.options(oopt)
```

---

newton.method *Demonstration of the Newton-Raphson method for root-finding*

Description

This function provides an illustration of the iterations in Newton's method.

Usage

```
newton.method(FUN = function(x) x^2 - 4, init = 10, rg = c(-1, 10), tol = 0.001,
    interact = FALSE, col.lp = c("blue", "red", "red"), main, xlab, ylab, ...)
```

Arguments

`FUN`	the function in the equation to solve (univariate), which has to be defined without braces like the default one (otherwise the derivative cannot be computed)
`init`	the starting point
`rg`	the range for plotting the curve
`tol`	the desired accuracy (convergence tolerance)
`interact`	logical; whether choose the starting point by cliking on the curve (for 1 time) directly?
`col.lp`	a vector of length 3 specifying the colors of: vertical lines, tangent lines and points
`main, xlab, ylab`	titles of the plot; there are default values for them (depending on the form of the function `FUN`)
`...`	other arguments passed to `curve`

Details

Newton's method (also known as the Newton-Raphson method or the Newton-Fourier method) is an efficient algorithm for finding approximations to the zeros (or roots) of a real-valued function f(x).

The iteration goes on in this way:

$$x_{k+1} = x_k - \frac{FUN(x_k)}{FUN'(x_k)}$$

From the starting value x_0, vertical lines and points are plotted to show the location of the sequence of iteration values $x_1, x_2, \ldots$; tangent lines are drawn to illustrate the relationship between successive iterations; the iteration values are in the right margin of the plot.

Value

A list containing

root	the root found by the algorithm
value	the value of FUN(root)
iter	number of iterations; if it is equal to ani.options('nmax'), it's quite likely that the root is not reliable because the maximum number of iterations has been reached

Note

The algorithm might not converge – it depends on the starting value. See the examples below.

Author(s)

Yihui Xie

References

http://en.wikipedia.org/wiki/Newton's_method

See Also

optim

Examples

```
oopt = ani.options(interval = 1, nmax = ifelse(interactive(), 50, 2))
par(pch = 20)

## default example
xx = newton.method()
xx$root  # solution

## take a long long journey
newton.method(function(x) 5 * x^3 - 7 * x^2 - 40 * x + 100, 7.15, c(-6.2, 7.1))

## another function
ani.options(interval = 0.5)
xx = newton.method(function(x) exp(-x) * x, rg = c(0, 10), init = 2)

## does not converge!
xx = newton.method(function(x) atan(x), rg = c(-5, 5), init = 1.5)
xx$root  # Inf

## interaction: use your mouse to select the starting point
```

```
if (interactive()) {
    ani.options(interval = 0.5, nmax = 50)
    xx = newton.method(function(x) atan(x), rg = c(-2, 2), interact = TRUE)
}

## HTML animation pages
saveHTML({
    ani.options(nmax = ifelse(interactive(), 100, 2))
    par(mar = c(3, 3, 1, 1.5), mgp = c(1.5, 0.5, 0), pch = 19)
    newton.method(function(x) 5 * x^3 - 7 * x^2 - 40 * x + 100, 7.15, c(-6.2,
        7.1), main = "")
}, img.name = "newton.method", htmlfile = "newton.method.html", ani.height = 500,
    ani.width = 600, title = "Demonstration of the Newton-Raphson Method",
    description = "Go along with the tangent lines and iterate.")

ani.options(oopt)
```

`ObamaSpeech` *Word counts of a speech by the US President Obama*

Description

This data recorded the number of words in each paragraph of Barack Obama's speech in Chicago after winning the presidential election.

Format

int [1:59] 2 45 52 53 11 48 28 15 50 29 ...

Source

The full text of speech is at http://www.baltimoresun.com/news/nation-world/bal-text1105,0,5055673,full.story

Examples

```
## pattern: longer paragraph and shorter paragraph
plot(ObamaSpeech, type = "b", pch = 20, xlab = "paragraph index", ylab = "word count")
```

pageview — *Page views from Sep 21, 2007 to Dec 2, 2007 of Yihui's website*

Description

The data is collected by Awstats for the website http://yihui.name.

Format

A data frame with 73 observations on the following 5 variables.

day Date starts from Sep 21, 2007 to Dec 2, 2007.

visits number of visits: a new visit is defined as each new *incoming visitor* (viewing or browsing a page) who was not connected to the site during last *60 min*.

pages number of times a *page* of the site is viewed (sum for all visitors for all visits). This piece of data differs from "files" in that it counts only HTML pages and excludes images and other files.

files number of times a *page, image, file* of the site is viewed or downloaded by someone.

bandwidth amount of data downloaded by all *pages*, *images* and *files* within the site (units in MegaBytes).

Source

http://yihui.name

Examples

```
plot(pageview[, 1:2], type = "b", col = "red", main = "Number of Visits in Yihui's Web")
## partial auto-correlation
pacf(pageview$visits)
```

pdftk — *A wrapper for the PDF toolkit Pdftk*

Description

If the toolkit Pdftk is available in the system, it will be called to manipulate the PDF files (especially to compress the PDF files).

Usage

```
pdftk(input, operation = NULL, output, other.opts = "compress dont_ask")
```

Arguments

input	the path of the input PDF file(s)
operation	the operation to be done on the input (default to be NULL)
output	the path of the output (if missing and input is a scalar, output will be the same as input)
other.opts	other options (default to be 'compress dont_ask', i.e. compress the PDF files and do not ask the user for any input)

Details

This is a wrapper to call pdftk. The path of pdftk should be set via ani.options(pdftk = 'path/to/pdftk').

See the reference for detailed usage of pdftk.

Value

if ani.options('pdftk') is non-NULL, then this function returns the status of the operation (0 for success; see system); otherwise a warning will be issued

Author(s)

Yihui Xie

References

http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/

Examples

```
pdf("huge-plot.pdf")
plot(rnorm(50000))
dev.off()

## Windows
ani.options(pdftk = "D:/Installer/pdftk.exe")
pdftk("huge-plot.pdf", output = "huge-plot0.pdf")

## Linux (does not work??)
ani.options(pdftk = "pdftk")
pdftk("huge-plot.pdf", output = "huge-plot1.pdf")

ani.options(pdftk = NULL)

file.info(c("huge-plot.pdf", "huge-plot0.pdf", "huge-plot1.pdf"))["size"]
```

pollen *Synthetic dataset about the geometric features of pollen grains*

Description

There are 3848 observations on 5 variables. From the 1986 ASA Data Exposition dataset, made up by David Coleman of RCA Labs

Format

A data frame with 3848 observations on the following 5 variables.

RIDGE a numeric vector

NUB a numeric vector

CRACK a numeric vector

WEIGHT a numeric vector

DENSITY a numeric vector

Source

collected from Statlib Datasets Archive: http://lib.stat.cmu.edu/data-expo/

Examples

```
## some dense points in the center?
plot(pollen[, 1:2], pch = 20, col = rgb(0, 0, 0, 0.1))

## see demo('pollen', package = 'animation') for a 3D demo; truth is there!
```

price.ani *Demonstrate stock prices in animations*

Description

This function can display the frequencies of stock prices in a certain time span with the span changing.

Usage

```
price.ani(price, time, time.begin = min(time), span = 15 * 60, ..., xlab = "price",
    ylab = "frequency", xlim, ylim, main)
```

Arguments

`price`	stock prices
`time`	time corresponding to prices
`time.begin`	the time for the animation to begin (default to be the minimum `time`)
`span`	time span (unit in seconds; default to be 15 minutes)
`xlab, ylab, xlim, ylim, main`	they are passed to `plot` with reasonable default values
`...`	other arguments passed to `plot`

Value

invisible `NULL`

Author(s)

Yihui Xie

Examples

```
## see more examples in ?vanke1127
saveHTML({
    price.ani(vanke1127$price, vanke1127$time, lwd = 2)
}, img.name = "vanke1127", htmlfile = "vanke1127.html", title = "Stock prices of Vanke",
    description = c("Barplots", "of the stock prices of Vanke Co. Ltd", "on 2009/11/27"))
```

---

qpdf — *A wrapper for the PDF toolkit qpdf*

Description

If the tool qpdf is available in the system, it will be called to manipulate the PDF files (especially to compress the PDF files).

Usage

```
qpdf(input, output, options = "--stream-data=compress")
```

Arguments

`input`	the path of the input PDF file
`output`	the path of the output (if missing, `output` will be the same as `input`)
`options`	options for `qpdf` (default to be `'--stream-data=compress'`, i.e. compress the PDF files)

Details

This is a wrapper to call qpdf. The path of qpdf should be set via ani.options(qpdf = 'path/to/qpdf').

See the reference for detailed usage of qpdf.

Value

if ani.options('qpdf') is non-NULL, then this function returns the status of the operation (0 for success; see system); otherwise a warning will be issued

Author(s)

Yihui Xie

References

http://qpdf.sourceforge.net/

Examples

```
pdf("huge-plot.pdf")
plot(rnorm(50000))
dev.off()

## Windows
ani.options(qpdf = "D:/Installer/qpdf/bin/qpdf.exe")
qpdf("huge-plot.pdf", output = "huge-plot0.pdf")

## Linux
ani.options(qpdf = "qpdf")
qpdf("huge-plot.pdf", output = "huge-plot1.pdf")

ani.options(qpdf = NULL)

file.info(c("huge-plot.pdf", "huge-plot0.pdf", "huge-plot1.pdf"))["size"]
```

---

quincunx *Demonstration of the Quincunx (Bean Machine/Galton Box)*

Description

Simulates the quincunx with “balls” (beans) falling through several layers (denoted by triangles) and the distribution of the final locations at which the balls hit is denoted by a histogram; quincunx() is shows single layer, and quincunx2() is a two-stage version of the quincunx.

Usage

```
quincunx(balls = 200, layers = 15, pch.layers = 2, pch.balls = 19,
    col.balls = sample(colors(), balls, TRUE), cex.balls = 2)

quincunx2(balls = 200, layers = 15, pch.layers = 2, pch.balls = 19,
    col.balls = sample(colors(), balls, TRUE), cex.balls = 2)
```

Arguments

`balls`	number of balls
`layers`	number of layers
`pch.layers`	point character of layers; triangles (`pch = 2`) are recommended
`pch.balls, col.balls, cex.balls`	point character, colors and magnification of balls

Details

The bean machine, also known as the quincunx or Galton box, is a device invented by Sir Francis Galton to demonstrate the law of error and the normal distribution.

When a ball falls through a layer, it can either go to the right or left side with the probability 0.5. At last the location of all the balls will show us the bell-shaped distribution.

Value

A named vector: the frequency table for the locations of the balls. Note the names of the vector are the locations: 1.5, 2.5, ..., layers - 0.5.

Note

The maximum number of animation frames is controlled by `ani.options('nmax')` as usual, but it is strongly recommended that `ani.options(nmax = balls + layers -2)`, in which case all the balls will just fall through all the layers and there will be no redundant animation frames.

Author(s)

Yihui Xie, Lijia Yu, and Keith ORourke

References

http://vis.supstat.com/2013/04/bean-machine

See Also

`rbinom`

Examples

```
set.seed(123)
oopt = ani.options(nmax = ifelse(interactive(), 200 + 15 -
    2, 2), interval = 0.03)
freq = quincunx(balls = 200, col.balls = rainbow(200))
## frequency table
barplot(freq, space = 0)

## HTML animation page
saveHTML({
    ani.options(nmax = ifelse(interactive(), 200 + 15 -
        2, 2), interval = 0.03)
    quincunx(balls = 200, col.balls = rainbow(200))
}, img.name = "quincunx", htmlfile = "quincunx.html", ani.height = 500,
    ani.width = 600, single.opts = paste("'controls':",
        "['first', 'previous', 'play', 'next', 'last', 'loop', 'speed'],",
        "'delayMin': 0"), title = "Demonstration of the Galton Box",
    description = c("Balls", "falling through pins will show you the Normal",
        "distribution."))

ani.options(oopt)
set.seed(123)
oopt = ani.options(nmax = ifelse(interactive(), 200 + 15 -
    2, 2), interval = 0.03)
freq = quincunx2(balls = 200, col.balls = rainbow(200))

## frequency table
barplot(freq$top, space = 0)  # top layers
barplot(freq$bottom, space = 0)  # bottom layers

## HTML animation page
saveHTML({
    ani.options(nmax = ifelse(interactive(), 200 + 15 -
        2, 2), interval = 0.03)
    quincunx2(balls = 200, col.balls = rainbow(200))
}, img.name = "quincunx2", htmlfile = "quincunx2.html",
    ani.height = 500, ani.width = 600, single.opts = paste("'controls':",
        "['first', 'previous', 'play', 'next', 'last', 'loop', 'speed'],",
        "'delayMin': 0"), title = "Demonstration of the Galton Box",
    description = c("Balls", "falling through pins will show you the Normal",
        "distribution."))

ani.options(oopt)
```

Rosling.bubbles *The bubbles animation in Hans Rosling's Talk*

Description

In Hans Rosling's attractive talk "Debunking third-world myths with the best stats you've ever seen", he used a lot of bubble plots to illustrate trends behind the data over time. This function gives an imitation of those moving bubbles, besides, as this function is based on symbols, we can also make use of other symbols such as squares, rectangles, thermometers, etc.

Usage

```
Rosling.bubbles(x, y, data, type = c("circles", "squares", "rectangles", "stars",
   "thermometers", "boxplots"), bg, xlim = range(x), ylim = range(y), main = NULL,
    xlab = "x", ylab = "y", ..., grid = TRUE, text = 1:ani.options("nmax"),
    text.col = rgb(0, 0, 0, 0.5), text.cex = 5)
```

Arguments

`x, y`	the x and y co-ordinates for the centres of the bubbles (symbols). Default to be 10 uniform random numbers in [0, 1] for each single image frame (so the length should be 10 * `ani.options('nmax')`)
`type, data`	the type and data for symbols; see `symbols`. The default type is `circles`.
`bg, main, xlim, ylim, xlab, ylab, ...`	see `symbols`. Note that bg has default values taking semi-transparent colors.
`grid`	logical; add a grid to the plot?
`text`	a character vector to be added to the plot one by one (e.g. the year in Rosling's talk)
`text.col, text.cex`	color and magnification of the background text

Details

Suppose we have observations of n individuals over `ani.options('nmax')` years. In this animation, the data of each year will be shown in the bubbles (symbols) plot; as time goes on, certain trends will be revealed (like those in Rosling's talk). Please note that the arrangement of the data for bubbles (symbols) should be a matrix like A_{ijk} in which i is the individual id (from 1 to n), j denotes the j-th variable (from 1 to p) and k indicates the time from 1 to `ani.options('nmax')`.

And the length of `x` and `y` should be equal to the number of rows of this matrix.

Value

`NULL`.

Author(s)

Yihui Xie

References

http://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen.html

See Also

symbols

Examples

```
oopt = ani.options(interval = 0.1, nmax = ifelse(interactive(), 50, 2))

## use default arguments (random numbers); you may try to find the real
## data
par(mar = c(4, 4, 0.2, 0.2))
Rosling.bubbles()

## rectangles
Rosling.bubbles(type = "rectangles", data = matrix(abs(rnorm(50 * 10 *
    2)), ncol = 2))

## save the animation in HTML pages
saveHTML({
    par(mar = c(4, 4, 0.2, 0.2))
    ani.options(interval = 0.1, nmax = ifelse(interactive(), 50, 2))
    Rosling.bubbles(text = 1951:2000)
}, img.name = "Rosling.bubbles", htmlfile = "Rosling.bubbles.html", ani.height = 450,
    ani.width = 600, title = "The Bubbles Animation in Hans Rosling's Talk",
    description = c("An imitation of Hans Rosling's moving bubbles.",
        "(with 'years' as the background)"))

ani.options(oopt)
```

---

sample.cluster *Demonstration for the cluster sampling*

Description

Each rectangle stands for a cluster, and the simple random sampling without replacement is performed for each cluster. All points in the clusters being sampled will be drawn out.

Usage

```
sample.cluster(pop = ceiling(10 * runif(10, 0.2, 1)), size = 3, p.col = c("blue",
    "red"), p.cex = c(1, 3), ...)
```

Arguments

pop	a vector for the size of each cluster in the population.
size	the number of clusters to be drawn out.
p.col, p.cex	different colors / magnification rate to annotate the population and the sample
...	other arguments passed to rect to annotate the "clusters"

Value

None (invisible NULL).

Author(s)

Yihui Xie

See Also

sample, sample.simple, sample.ratio, sample.strat, sample.system

Examples

```
oopt = ani.options(nmax = ifelse(interactive(), 50, 2))
par(mar = rep(1, 4))
sample.cluster(col = c("bisque", "white"))

## HTML animation page
saveHTML({
    par(mar = rep(1, 4), lwd = 2)
    ani.options(nmax = ifelse(interactive(), 50, 2))
    sample.cluster(col = c("bisque", "white"))
}, img.name = "sample.cluster", htmlfile = "sample.html", ani.height = 350,
    ani.width = 500, title = "Demonstration of the cluster sampling",
    description = c("Once a cluster is sampled,", "all its elements will be chosen."))

ani.options(oopt)
```

---

sample.ratio *Demonstrate the ratio estimation in sampling survey*

Description

This function demonstrates the advantage of ratio estimation when further information (ratio) about x and y is available.

Usage

```
sample.ratio(X = runif(50, 0, 5), R = 1, Y = R * X + rnorm(X), size = length(X)/2,
   p.col = c("blue", "red"), p.cex = c(1, 3), p.pch = c(20, 21), m.col = c("black",
        "gray"), legend.loc = "topleft", ...)
```

Arguments

X	the X variable (ancillary)
R	the population ratio Y/X
Y	the Y variable (whose mean we what to estimate)

`size`	sample size
`p.col, p.cex, p.pch`	point colors, magnification and symbols for the population and sample respectively
`m.col`	color for the horizontal line to denote the sample mean of Y
`legend.loc`	legend location: topleft, topright, bottomleft, bottomright, ... (see `legend`)
`...`	other arguments passed to `plot.default`

Details

From this demonstration we can clearly see that the ratio estimation is generally better than the simple sample average when the ratio **R** really exists, otherwise ratio estimation may not help.

Value

A list containing

`X`	X population
`Y`	Y population
`R`	population ratio
`r`	ratio calculated from samples
`Ybar`	population mean of Y
`ybar.simple`	simple sample mean of Y
`ybar.ratio`	sample mean of Y via ratio estimation

Author(s)

Yihui Xie

See Also

`sample`, `sample.simple`, `sample.cluster`, `sample.strat`, `sample.system`

Examples

```
oopt = ani.options(interval = 2, nmax = ifelse(interactive(), 50, 2))

## observe the location of the red line (closer to the population mean)
res = sample.ratio()

## absolute difference with the true mean
matplot(abs(cbind(res$ybar.ratio, res$ybar.simple) - res$Ybar), type = "l")
legend("topleft", c("Ratio Estimation", "Sample Average"), lty = 1:2,
    col = 1:2)

## if the ratio does not actually exist:
sample.ratio(X = rnorm(50), Y = rnorm(50))
## ratio estimation may not be better than the simple average
```

```
## HTML animation page
saveHTML({
    par(mar = c(4, 4, 1, 0.5), mgp = c(2, 1, 0))
    ani.options(interval = 2, nmax = ifelse(interactive(), 50, 2))
    sample.ratio()
}, img.name = "sample.ratio", htmlfile = "sample.ratio.html", ani.height = 400,
    ani.width = 500, title = "Demonstration of the Ratio Estimation",
    description = c("Estimate the mean of Y, making use of the ratio",
        "Y/X which will generally improve the estimation."))

ani.options(oopt)
```

`sample.simple` *Demonstration for simple random sampling without replacement*

Description

The whole sample frame is denoted by a matrix (`nrow * ncol`) in the plane just for convenience, and the points being sampled are marked out (by red circles by default). Each member of the population has an equal and known chance of being selected.

Usage

```
sample.simple(nrow = 10, ncol = 10, size = 15, p.col = c("blue", "red"), p.cex = c(1,
    3))
```

Arguments

`nrow`	the desired number of rows of the sample frame.
`ncol`	the desired number of columns of the sample frame.
`size`	the sample size.
`p.col, p.cex`	different colors /magnification rate to annotate the population and the sample

Value

None (invisible `NULL`).

Author(s)

Yihui Xie

See Also

`sample`, `sample.ratio`, `sample.cluster`, `sample.strat`, `sample.system`

Examples

```
oopt = ani.options(nmax = ifelse(interactive(), 50, 2))
par(mar = rep(1, 4))
sample.simple()

## HTML animation page
saveHTML({
    par(mar = rep(1, 4), lwd = 2)
    ani.options(nmax = ifelse(interactive(), 50, 2))
    sample.simple()
}, img.name = "sample.simple", htmlfile = "sample.html",
    ani.height = 350, ani.width = 500, title = paste("Demonstration of",
        "the simple random sampling without replacement"),
    description = c("Each member of the population has an equal and",
        "known chance of being selected."))

ani.options(oopt)
```

---

sample.strat   *Demonstration for the stratified sampling*

Description

Each rectangle stands for a stratum, and the simple random sampling without replacement is performed within each stratum. The points being sampled are marked out (by red circles by default).

Usage

```
sample.strat(pop = ceiling(10 * runif(10, 0.5, 1)), size = ceiling(pop *
    runif(length(pop), 0, 0.5)), p.col = c("blue", "red"), p.cex = c(1, 3),
    ...)
```

Arguments

`pop`	a vector for the size of each stratum in the population.
`size`	a corresponding vector for the sample size in each stratum (recycled if necessary).
`p.col, p.cex`	different colors /magnification rate to annotate the population and the sample
`...`	other arguments passed to `rect` to annotate the “strata”

Value

None (invisible ‘`NULL`’).

Author(s)

Yihui Xie

See Also

sample, sample.simple, sample.cluster, sample.ratio, sample.system

Examples

```
oopt = ani.options(nmax = ifelse(interactive(), 50, 2))
par(mar = rep(1, 4), lwd = 2)

sample.strat(col = c("bisque", "white"))

## HTML animation page
saveHTML({
    par(mar = rep(1, 4), lwd = 2)
    ani.options(nmax = ifelse(interactive(), 50, 2))
    sample.strat(col = c("bisque", "white"))
}, img.name = "sample.strat", htmlfile = "sample.html", ani.height = 350,
    ani.width = 500, title = "Demonstration of the stratified sampling",
    description = c("Every rectangle stands for a stratum, and the simple",
        "random sampling without replacement is performed within each stratum."))

ani.options(oopt)
```

---

sample.system *Demonstration for the systematic sampling*

Description

The whole sample frame is denoted by a matrix (`nrow * ncol`) in the plane, and the sample points with equal intervals are drawn out according to a random starting point. The points being sampled are marked by red circles.

Usage

```
sample.system(nrow = 10, ncol = 10, size = 15, p.col = c("blue", "red"), p.cex = c(1,
    3))
```

Arguments

nrow	the desired number of rows of the sample frame.
ncol	the desired number of columns of the sample frame.
size	the sample size.
p.col, p.cex	different colors / magnification rate to annotate the population and the sample

Value

None (invisible `NULL`).

Author(s)

Yihui Xie

See Also

sample, sample.simple, sample.cluster, sample.ratio, sample.strat

Examples

```
oopt = ani.options(nmax = ifelse(interactive(), 50, 2))
par(mar = rep(1, 4), lwd = 2)

sample.system()

## HTML animation pages
saveHTML({
    ani.options(interval = 1, nmax = ifelse(interactive(), 30, 2))
    par(mar = rep(1, 4), lwd = 2)
    sample.system()
}, img.name = "sample.system", htmlfile = "sample.html", ani.height = 350,
    ani.width = 500, title = "Demonstration of the systematic sampling",
    description = "Sampling with equal distances.")

ani.options(oopt)
```

---

saveGIF *Convert images to a single animation file (typically GIF) using ImageMagick or GraphicsMagick*

Description

This function opens a graphical device (specified in `ani.options('ani.dev')`) first to generate a sequence of images based on `expr`, then makes use of the command `convert` in 'ImageMagick' to convert these images to a single animated movie (as a GIF or MPG file). An alternative software package is GraphicsMagick (use `convert = 'gm convert'`), which is smaller than ImageMagick.

Usage

```
saveGIF(expr, movie.name = "animation.gif", img.name = "Rplot", convert = "convert",
    cmd.fun, clean = TRUE, ...)

saveMovie(expr, movie.name = "animation.gif", img.name = "Rplot", convert = "convert",
    cmd.fun, clean = TRUE, ...)
```

Arguments

`expr`	an expression to generate animations; use either the animation functions (e.g. `brownian.motion()`) in this package or a custom expression (e.g. `for(i in 1:10) plot(runif(10), y`
`movie.name`	file name of the movie (with the extension)
`img.name`	file name of the sequence of images ('pure' name; without any format or extension)
`convert`	the command to convert images (default to be `convert` (i.e. use ImageMagick), but might be `imconvert` under some Windows platforms); can be `gm convert` in order to use GraphicsMagick; see the 'Note' section for details
`cmd.fun`	a function to invoke the OS command; by default `system`
`clean`	whether to delete the individual image frames
`...`	other arguments passed to `ani.options`, e.g. `ani.height` and `ani.width`, ...

Details

This function calls `im.convert` (or `gm.convert`, depending on the argument `convert`) to convert images to a single animation.

The advantage of this function is that it can create a single movie file, however, there are two problems too: (1) we need a special (free) software ImageMagick or GraphicsMagick; (2) the speed of the animation will be beyond our control, as the `interval` option is fixed. Other approaches in this package may have greater flexibilities, e.g. the HTML approach (see `saveHTML`).

See `ani.options` for the options that may affect the output, e.g. the graphics device (including the height/width specifications), the file extension of image frames, and the time interval between image frames, etc. Note that `ani.options('interval')` can be a numeric vector!

Value

The command for the conversion (see `im.convert`).

Note

See `im.convert` for details on the configuration of ImageMagick (typically for Windows users) or GraphicsMagick.

It is recommended to use `ani.pause()` to pause between animation frames in `expr`, because this function will only pause when called in a non-interactive graphics device, which can save a lot of time. See the demo `'Xmas2'` for example (`demo('Xmas2', package = 'animation')`).

`saveGIF` has an alias `saveMovie` (i.e. they are identical); the latter name is for compatibility to older versions of this package (< 2.0-2). It is recommended to use `saveGIF` to avoid confusions between `saveMovie` and `saveVideo`.

Author(s)

Yihui Xie

References

ImageMagick: http://www.imagemagick.org/script/convert.php; GraphicsMagick: http://www.graphicsmagick.org

See Also

Other utilities: gm.convert, im.convert; saveHTML; saveLatex; saveSWF; saveVideo

Examples

```
## make sure ImageMagick has been installed in your system
saveGIF({
    for (i in 1:10) plot(runif(10), ylim = 0:1)
})

## if the above conversion was successful, the option 'convert' should not be NULL
## under Windows
ani.options("convert")
## like 'C:/Software/LyX/etc/ImageMagick/convert.exe'

saveGIF({
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, movie.name = "brownian_motion.gif", interval = 0.1, nmax = 30, ani.width = 600,
    ani.height = 600)

## non-constant intervals between image frames
saveGIF({
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, movie.name = "brownian_motion2.gif", interval = runif(30, 0.01, 1), nmax = 30)
```

---

saveHTML *Insert animations into an HTML page*

Description

This function first records all the plots in the R expression as bitmap images, then inserts them into an HTML page and finally creates the animation using the SciAnimator library.

Usage

```
saveHTML(expr, img.name = "Rplot", global.opts = "", single.opts = "",
    navigator = ani.options("nmax") <= 100 && ani.options("interval") >=
        0.05, htmlfile = "index.html", ...)
```

Arguments

`expr` an R expression to be evaluated to create a sequence of images

`img.name` the filename of the images (the real output will be like 'img.name1.png', 'img.name2.png', ...); this name has to be different for different animations, since it will be used as the identifiers for each animation; make it as unique as possible; meanwhile, the following characters in `img.name` will be replaced by _ to make it a legal jQuery string:
!"#$%&'()*+,./:;?@[\]^`{|}~

`global.opts` a string: the global options of the animation; e.g. we can specify the default theme to be blue using $.fn.scianimator.defaults.theme = 'blue'; note these options must be legal JavaScript expressions (ended by ';')

`single.opts` the options for each single animation (if there are multiple ones in one HTML page), e.g. to use the dark theme and text labels for buttons:
'utf8': false, 'theme': 'dark'

or to remove the navigator panel (the navigator can affect the smoothness of the animation when the playing speed is extremely fast (e.g. `interval` less than 0.05 seconds)):
'controls': ['first', 'previous', 'play', 'next', 'last', 'loop',
'speed']

see the reference for a complete list of available options

`navigator` whether to show the navigator (like a progress bar); by default, the navigator is not shown for performance reasons when the number of images is greater than 100 or the time interval is smaller than 0.05

`htmlfile` the filename of the HTML file

`...` other arguments to be passed to `ani.options` to animation options such as the time interval between image frames

Details

It mainly uses the SciAnimator library, which is based on jQuery. It has a neat interface (both technically and visually) and is much easier to use or extend. Moreover, this function allows multiple animations in a single HTML page – just use the same HTML filename.

Optionally the source code and some session information can be added below the animations for the sake of reproducibility (specified by the option `ani.options('verbose')` – if `TRUE`, the description, loaded packages, the code to produce the animation, as well as a part of `sessionInfo()` will be written in the bottom of the animation; the R code will be highlighted using the SyntaxHighlighter library for better reading experience).

Value

The path of the HTML output.

Note

Microsoft IE might restrict the HTML page from running JavaScript and try to "protect your security" when you view the animation page, but this is not really a security problem.

When you want to publish the HTML page on the web, you have to upload the associated 'css' and 'js' folders with the HTML file as well as the images.

For saveHTML, ani.options('description') can be a character vector, in which case this vector will be pasted into a scalar; use '\n\n' in the string to separate paragraphs (see the first example below).

For the users who do not have R at hand, there is a demo in system.file('misc', 'Rweb', 'demo.html', package = 'an to show how to create animations online without R being installed locally. It depends, however, on whether the Rweb service can be provided for public use in a long period (currently we are using the Rweb at Tama University). See the last example below.

Author(s)

Yihui Xie

References

https://github.com/brentertz/scianimator

See Also

Other utilities: gm.convert, im.convert; saveGIF, saveMovie; saveLatex; saveSWF; saveVideo

Examples

```
## A quick and dirty demo
des = c("This is a silly example.\n\n", "You can describe it in more detail.",
    "For example, bla bla...")
saveHTML({
    par(mar = c(4, 4, 0.5, 0.5))
    for (i in 1:20) {
        plot(runif(20), ylim = c(0, 1))
        ani.pause()
    }
}, img.name = "unif_plot", imgdir = "unif_dir", htmlfile = "random.html",
    autobrowse = FALSE, title = "Demo of 20 uniform random numbers",
    description = des)



## we can merge another animation into the former page as long as
## 'htmlfile' is the same; this time I don't want the animation
## to autoplay, and will use text labels for the buttons (instead
## of UTF-8 symbols)
des = c("When you write a long long long long description, R will try to wrap the",
    "words automatically.", "Oh, really?!")
saveHTML({
    par(mar = c(4, 4, 0.5, 0.5))
    ani.options(interval = 0.5)
    for (i in 1:10) {
        plot(rnorm(50), ylim = c(-3, 3))
        ani.pause()
```

```
    }
}, img.name = "norm_plot", single.opts = "utf8: false", autoplay = FALSE,
    interval = 0.5, imgdir = "norm_dir", htmlfile = "random.html",
    ani.height = 400, ani.width = 600, title = "Demo of 50 Normal random numbers",
    description = des)



## use the function brownian.motion() in this package; this page
## is created in 'index.html' under the current working directory
des = c("Random walk of 10 points on the 2D plane:", "for each point (x, y),",
    "x = x + rnorm(1) and y = y + rnorm(1).")
saveHTML({
    par(mar = c(3, 3, 1, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8,
        cex.lab = 0.8, cex.main = 1)
    ani.options(interval = 0.05, nmax = ifelse(interactive(), 150,
        2))
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, img.name = "brownian_motion_a", htmlfile = "index.html", description = des)



## remove the 'navigator' (progress bar)
saveHTML({
    par(mar = c(3, 3, 1, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8,
        cex.lab = 0.8, cex.main = 1)
    ani.options(interval = 0.05, nmax = ifelse(interactive(), 150,
        2))
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, img.name = "brownian_motion_b", htmlfile = "index.html", navigator = FALSE,
    description = c("Random walk of 10 points on the 2D plane",
        "(without the navigation panel)"))


## use Rweb to create animations
if (interactive()) browseURL(system.file("misc", "Rweb", "demo.html",
    package = "animation"))
```

---

saveLatex — *Insert animations into a LaTeX document and compile it*

Description

Record animation frames and insert them into a LaTeX document with the `animate` package. Compile the document if an appropriate LaTeX command is provided.

Usage

```
saveLatex(expr, nmax, img.name = "Rplot", ani.opts, centering = TRUE,
    caption = NULL, label = NULL, pkg.opts = NULL, documentclass = "article",
```

```
latex.filename = "animation.tex", pdflatex = "pdflatex", install.animate = TRUE,
 overwrite = TRUE, full.path = FALSE, ...)
```

Arguments

`expr`	an expression to generate animations; use either the animation functions (e.g. `brownian.motion()`) in this package or a custom expression (e.g. `for(i in 1:10) plot(runif(10), y`
`nmax`	maximum number of animation frames (if missing and the graphics device is a bitmap device, this number will be automatically calculated); note that we do not have to specify `nmax` when using PDF devices.
`img.name`	basename of file names of animation frames; see the Note section for a possible adjustment on `img.name`
`ani.opts`	options to control the behavior of the animation (passed to the LaTeX macro `'\animategraphics'`; default to be `'controls,width=\linewidth'`)
`centering`	logical: whether to center the graph using the LaTeX environment \begin{center} and \end{center}
`caption, label`	caption and label for the graphics in the figure environment
`pkg.opts`	global options for the `animate` package
`documentclass`	LaTeX document class; if `NULL`, the output will not be a complete LaTeX document (only the code to generate the PDF animation will be printed in the console); default to be `article`, but we can also provide a complete statement like \documentclass[a5paper]{article}
`latex.filename`	file name of the LaTeX document; if an empty string `''`, the LaTeX code will be printed in the console and hence not compiled
`pdflatex`	the command for pdfLaTeX (set to `NULL` to ignore the compiling)
`install.animate`	copy the LaTeX style files '`animate.sty`' and '`animfp.sty`'? If you have not installed the LaTeX package `animate`, it suffices just to copy these to files.
`overwrite`	whether to overwrite the existing image frames
`full.path`	whether to use the full path (`TRUE`) or relative path (`FALSE`) for the animation frames; usually the relative path suffices, but sometimes the images and the LaTeX document might not be in the same directory, so `full.path = TRUE` could be useful; in the latter case, remember that you should never use spaces in the filenames or paths!
`...`	other arguments passed to the graphics device `ani.options('ani.dev')`, e.g. `ani.height` and `ani.width`

Details

This is actually a wrapper to generate a LaTeX document using R. The document uses the LaTeX package called `animate` to insert animations into PDF's. When we pass an R expression to this function, the expression will be evaluated and recorded by a grahpics device (typically `png` and `pdf`). At last, a LaTeX document will be created and compiled if an appropriate LaTeX command is provided. And the final PDF output will be opened with the PDF viewer set in `getOption('pdfviewer')` if `ani.options('autobrowse') == TRUE`.

Value

Invisible NULL

Note

This function will detect if it was called in a Sweave environment – if so, img.name will be automatically adjusted to prefix.string-label, and the LaTeX output will not be a complete document, but rather a single line like

```
\animategraphics[ani.opts]{1/interval}{img.name}{}{}
```

This automatic feature can be useful to Sweave users (but remember to set the Sweave option results=tex). See demo('Sweave_animation') for a complete example.

PDF devices are recommended because of their high quality and usually they are more friendly to LaTeX, but the size of PDF files is often much larger; in this case, we may set the option 'qpdf' or 'pdftk' to compress the PDF graphics output. To set the PDF device, use ani.options(ani.dev = 'pdf', ani.type = 'p

So far animations created by the LaTeX package **animate** can only be viewed with Acrobat Reader (Windows) or acroread (Linux). Other PDF viewers may not support JavaScript (in fact the PDF animation is driven by JavaScript). Linux users may need to install acroread and set options(pdfviewer = 'acroread').

Author(s)

Yihui Xie

References

To know more about the animate package, please refer to http://www.ctan.org/tex-archive/macros/latex/contrib/animate/. There are a lot of options can be set in ani.opts and pkg.opts.

See Also

Other utilities: gm.convert, im.convert; saveGIF, saveMovie; saveHTML; saveSWF; saveVideo

Examples

```
## brownian motion: note the 'loop' option in ani.opts and the
## careful settings in documentclass

saveLatex({
    par(mar = c(3, 3, 1, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8,
        cex.lab = 0.8, cex.main = 1)
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow",
        main = "Demonstration of Brownian Motion")
}, img.name = "BM", ani.opts = "controls,loop,width=0.95\\textwidth",
    latex.filename = ifelse(interactive(), "brownian_motion.tex", ""),
    interval = 0.1, nmax = 10, ani.dev = "pdf", ani.type = "pdf", ani.width = 7,
    ani.height = 7, documentclass = paste("\\documentclass{article}",
        "\\usepackage[papersize={7in,7in},margin=0.3in]{geometry}",
        sep = "\n"))
```

```
## the PDF graphics output is often too large because it is
## uncompressed; try the option ani.options('pdftk') or
## ani.options('qpdf') to compress the PDF graphics; see ?pdftk or
## ?qpdf and ?ani.options
```

---

saveSWF *Convert images to Flash animations*

Description

This function opens a graphical device first to generate a sequence of images based on `expr`, then makes use of the commands in SWFTools (`png2swf`, `jpeg2swf`, `pdf2swf`) to convert these images to a single Flash animation.

Usage

```
saveSWF(expr, swf.name = "animation.swf", img.name = "Rplot", swftools = NULL, ...)
```

Arguments

`expr`	an expression to generate animations; use either the animation functions (e.g. `brownian.motion()`) in this package or a custom expression (e.g. `for(i in 1:10) plot(runif(10), y`
`swf.name`	file name of the Flash file
`img.name`	the base file name of the sequence of images (without any format or extension)
`swftools`	the path of SWFTools, e.g. '`C:/swftools`'. This argument is to make sure that `png2swf`, `jpeg2swf` and `pdf2swf` can be executed correctly. If it is `NULL`, it should be guaranteed that these commands can be executed without the path; anyway, this function will try to find SWFTools from Windows registry even if it is not in the PATH variable.
`...`	other arguments passed to `ani.options`, e.g. `ani.height` and `ani.width`, ...

Value

An integer indicating failure (-1) or success (0) of the converting (refer to `system`).

Note

Please download and install the SWFTools before using this function: `http://www.swftools.org`

We can also set the path to SWF Tools by `ani.options(swftools = 'path/to/swftools')`.

`ani.options('ani.type')` can only be one of `png`, `pdf` and `jpeg`.

Also note that PDF graphics can be compressed using qpdf or Pdftk (if either one is installed and `ani.options('qpdf')` or `ani.options('pdftk')` has been set); see `qpdf` or `pdftk`.

Author(s)

Yihui Xie

See Also

Other utilities: gm.convert, im.convert; saveGIF, saveMovie; saveHTML; saveLatex; saveVideo

Examples

```
## from png to swf
saveSWF({
    par(mar = c(3, 3, 1, 1.5), mgp = c(1.5, 0.5, 0))
    knn.ani(test = matrix(rnorm(16), ncol = 2), cl.pch = c(16, 2))
}, swf.name = "kNN.swf", interval = 1.5, nmax = ifelse(interactive(), 40,
    2))

## from pdf (vector plot) to swf; can set the option 'pdftk' to compress
## PDF
saveSWF({
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, swf.name = "brownian.swf", interval = 0.2, nmax = 30, ani.dev = "pdf",
    ani.type = "pdf", ani.height = 6, ani.width = 6)
```

---

saveVideo *Convert a sequence of images to a video by FFmpeg*

Description

This function opens a graphics device to record the images produced in the code `expr`, then uses FFmpeg to convert these images to a video.

Usage

```
saveVideo(expr, video.name = "animation.mp4", img.name = "Rplot",
    ffmpeg = ani.options("ffmpeg"), other.opts = if (grepl("[.]mp4$",
        video.name)) "-pix_fmt yuv420p", ...)
```

Arguments

`expr`	the R code to draw (several) plots
`video.name`	the file name of the output video (e.g. ‘`animation.mp4`’ or ‘`animation.avi`’)
`img.name`	the file name of the sequence of images to be generated
`ffmpeg`	the command to call FFmpeg (e.g. `'C:/Software/ffmpeg/bin/ffmpeg.exe'` under Windows or ’avconv’ on some linux machines); note the full path of FFmpeg can be pre-specified in `ani.options('ffmpeg')`
`other.opts`	other options to be passed to `ffmpeg`, e.g. we can specify the bitrate as `other.opts = '-b 400k'` (The default `"-pix_fmt yuv420p"` is a work-around for a bug in some versions of ffmpeg.)
`...`	other arguments to be passed to `ani.options`

Details

This function uses system to call FFmpeg to convert the images to a single video. The command line used in this function is: `ffmpeg -y -r <1/interval> -i <img.name>%d.<ani.type> other.opts video.name`

where `interval` comes from `ani.options('interval')`, and `ani.type` is from `ani.options('ani.type')`. For more details on the numerous options of FFmpeg, please see the reference.

Some linux systems may use the alternate software 'avconv' instead of 'ffmpeg'. The package will attempt to determine which command is present and set `ani.options('ffmpeg')` to an appropriate default value. This can be overridden by passing in the `ffmpeg` argument.

Value

An integer indicating failure (-1) or success (0) of the converting (refer to system).

Note

There are a lot of possibilities in optimizing the video. My knowledge on FFmpeg is very limited, hence the default output by this function could be of low quality or too large. The file 'presets.xml' of WinFF might be a good guide: http://code.google.com/p/winff/.

Author(s)

Yihui Xie, based on an inital version by Thomas Julou <thomas.julou@gmail.com>

References

http://ffmpeg.org/documentation.html

See Also

Other utilities: gm.convert, im.convert; saveGIF, saveMovie; saveHTML; saveLatex; saveSWF

Examples

```
oopts = if (.Platform$OS.type == "windows") {
    ani.options(ffmpeg = "D:/Installer/ffmpeg/bin/ffmpeg.exe")
}
## usually Linux users do not need to worry about the ffmpeg path as long as
## FFmpeg or avconv has been installed

saveVideo({
    par(mar = c(3, 3, 1, 0.5), mgp = c(2, 0.5, 0), tcl = -0.3, cex.axis = 0.8,
        cex.lab = 0.8, cex.main = 1)
    ani.options(interval = 0.05, nmax = 300)
    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
}, video.name = "BM.mp4", other.opts = "-pix_fmt yuv420p -b 300k")
# higher bitrate, better quality

ani.options(oopts)
```

sim.qqnorm *Simulation of QQ plots for the Normal distribution*

Description

This demo shows the possible QQ plots created by random numbers generated from a Normal distribution so that users can get a rough idea about how QQ plots really look like.

Usage

```
sim.qqnorm(n = 20, last.plot = NULL, ...)
```

Arguments

`n`	integer: sample size
`last.plot`	an expression to be evaluated after the plot is drawn, e.g. `expression(abline(0, 1))` to add the diagonal line
`...`	other arguments passed to `qqnorm`

Details

When the sample size is small, it is hard to get a correct inference about the distribution of data from a QQ plot. Even if the sample size is large, usually there are outliers far away from the straight line. Therefore, don't overinterpret the QQ plots.

Value

`NULL`

Author(s)

Yihui Xie

See Also

`qqnorm`

Examples

```
oopt = ani.options(interval = 0.1, nmax = ifelse(interactive(), 100,
    2))
par(mar = c(3, 3, 2, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)

sim.qqnorm(n = 20, last.plot = expression(abline(0, 1)))

## HTML animation pages
saveHTML({
    par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
```

```
    ani.options(interval = 0.1, nmax = ifelse(interactive(), 100, 2))
    sim.qqnorm(n = 15, pch = 20, main = "")
}, img.name = "sim.qqnorm", htmlfile = "sim.qqnorm.html", ani.height = 500,
    ani.width = 500, title = "Demonstration of Simulated QQ Plots",
    description = c("This animation shows the QQ plots of random numbers",
        "from a Normal distribution. Does them really look like normally",
        "distributed?"))

ani.options(oopt)
```

---

vanke1127 *Stock prices of Vanke Co., Ltd on 2009/11/27*

Description

This is a sample of stock prices of the Vanke Co., Ltd on 2009/11/27.

Format

A data frame with 2831 observations on the following 2 variables.

time POSIXt: the time corresponding to stock prices

price a numeric vector: stock prices

Source

This data can be obtained from most stock websites.

Examples

```
tab.price = table(vanke1127$price)
plot(as.numeric(names(tab.price)), as.numeric(tab.price), type = 'h',
     xlab = 'price', ylab = 'frequency')

oopt = ani.options(interval = 0.5, loop = FALSE, title = 'Stock price of Vanke')

## a series of HTML animations with different time spans
saveHTML({
  price.ani(vanke1127$price, vanke1127$time, lwd = 2)
}, img.name = 'vanke_a', description = 'Prices changing along with time interval 15 min',
  htmlfile = "vanke1127_1.html")

saveHTML({
  price.ani(vanke1127$price, vanke1127$time, span = 30 * 60, lwd = 3)
}, img.name = 'vanke_b', description = 'Prices changing along with time interval 30 min',
  htmlfile = "vanke1127_2.html")

saveHTML({
  price.ani(vanke1127$price, vanke1127$time, span = 5 * 60, lwd = 2)
```

```
}, img.name = 'vanke_c', description = 'Prices changing along with time interval 5 min',
  htmlfile = "vanke1127_3.html")

## GIF animation
saveGIF(price.ani(vanke1127$price, vanke1127$time, lwd = 2),
        movie.name = 'price.gif', loop = 1)

ani.options(oopt)
```

---

vi.grid.illusion — *Visual illusions: Scintillating grid illusion and Hermann grid illusion*

Description

The two most common types of grid illusions are Hermann grid illusions and Scintillating grid illusions. This function provides illustrations for both illusions.

Usage

```
vi.grid.illusion(nrow = 8, ncol = 8, lwd = 8, cex = 3, col = "darkgray", type = c("s",
    "h"))
```

Arguments

nrow	number of rows for the grid
ncol	number of columns for the grid
lwd	line width for grid lines
cex	magnification for points in Scintillating grid illusions
col	color for grid lines
type	type of illusions: 's' for Scintillating grid illusions and 'h' for Hermann grid illusions

Details

A grid illusion is any kind of grid that deceives a person's vision.

This is actually a static image; pay attention to the intersections of the grid and there seems to be some moving points (non-existent in fact).

Value

NULL

Author(s)

Yihui Xie

References

http://vis.supstat.com/2013/03/make-visual-illusions-in-r

See Also

points, abline

Examples

```
## default to be Scintillating grid illusions
vi.grid.illusion()

## set wider lines to see Hermann grid illusions
vi.grid.illusion(type = "h", lwd = 22, nrow = 5, ncol = 5, col = "white")
```

---

vi.lilac.chaser *Visual Illusions: Lilac Chaser*

Description

Stare at the center cross for a few (say 30) seconds to experience the phenomena of the illusion.

Usage

```
vi.lilac.chaser(np = 16, col = "magenta", bg = "gray", p.cex = 7, c.cex = 5)
```

Arguments

np	number of points
col	color of points
bg	background color of the plot
p.cex	magnification of points
c.cex	magnification of the center cross

Details

Just try it out.

Value

NULL

Note

In fact, points in the original version of 'Lilac Chaser' are *blurred*, which is not implemented in this function.

Author(s)

Yihui Xie

References

http://vis.supstat.com/2013/03/make-visual-illusions-in-r

See Also

points

Examples

```
oopt = ani.options(interval = 0.05, nmax = 20)
par(pty = "s")
vi.lilac.chaser()

## HTML animation page; nmax = 1 is enough!
saveHTML({
    ani.options(interval = 0.05, nmax = 1)
    par(pty = "s", mar = rep(1, 4))
    vi.lilac.chaser()
}, img.name = "vi.lilac.chaser", htmlfile = "vi.lilac.chaser.html",
    ani.height = 480, ani.width = 480, title = "Visual Illusions: Lilac Chaser",
    description = c("Stare at the center cross for a few (say 30) seconds",
        "to experience the phenomena of the illusion."))

ani.options(oopt)
```

Index