

The animation Package

June 20, 2008

Type Package

Title Demonstrate Animations in Statistics

Version 1.0-1

Date 2008-06-20

Author Yihui Xie

Maintainer Yihui Xie <xieyihui@gmail.com>

Description This package consists of various functions for animations in statistics, covering many areas such as probability theory, mathematical statistics, multivariate statistics, nonparametric statistics, sampling survey, linear models, time series, computational statistics, data mining and machine learning. These functions might be of help in teaching statistics and data analysis.

Depends MASS

License GPL-2 | GPL-3

URL <http://animation.yihui.name>

R topics documented:

ani.news	2
ani.options	3
ani.start	5
ani.stop	6
animation-package	7
bisection.method	8
boot.iid	10
brownian.motion	12
buffon.needle	14
clt.ani	16
conf.int	17
cv.ani	19
cv.nfeaturesLDA	20

flip.coin	22
grad.desc	24
highlight.def	26
kfcv	27
kmeans.ani	28
knn.ani	30
lln.ani	32
mwar.ani	34
newton.method	36
pageview	38
sample.cluster	39
sample.simple	40
sample.strat	41
sample.system	43
saveMovie	44
saveSWF	46
tidy.source	47
vi.grid.illusion	48
vi.lilac.chaser	50
write.rss	51

Index **54**

<code>ani.news</code>	<i>Read news of package ‘animation’</i>
-----------------------	---

Description

Read news and changes in the package ‘animation’.

Usage

```
ani.news(...)
```

Arguments

... arguments passed to `file.show`.

Details

This function just makes use of `file.show` to display a file ‘NEWS’ in this package.

Value

None (invisible ‘NULL’).

Author(s)

Yihui Xie

See Also[file.show](#)**Examples**

```
ani.news()
```

```
ani.options
```

Set or Query Animation Parameters

Description

Set or query various parameters that control the behaviour of the animation, such as time interval, maximum frames, height and width, etc. This function is based on [options](#) to set an option `ani` which is a list containing the animation parameters.

Usage

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

Arguments

... arguments in `tag = value` form, or a list of tagged values. The tags must come from the animation parameters described below.

Value

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.

Animation Parameters

`interval` a positive number to set the time interval of the animation (unit in seconds).

`nmax` maximum number of steps for 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.

`ani.width`, `ani.height` width and height of image frames (unit in px); see graphics devices like [png](#), [jpeg](#), ...

`outdir` character: specify the output dir if we want to create HTML animation pages; default to be [tempdir](#).

`filename` character: name of the target HTML main file

`withprompt` character: prompt to display while using [ani.start](#) (restore with [ani.stop](#))

ani.type character: image format for animation frames, e.g. png, jpeg, ...

ani.dev function: the graphics device; e.g. (png, jpeg, ...)

title character: the title of animation

description character: a description about the animation

footer logical: if TRUE, write a foot part in the HTML page containing information such as date/time of creation; else don't write.

autobrowse logical: whether auto-browse the animation page immediately after it is created?

Note

Please note that `nmax` is usually equal to the number of animation frames (e.g. for `brownian.motion`) but not *always*! The reason is that sometimes there are 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.

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. All the parameters will affect the behaviour of HTML animations, but only `interval` will affect animations in windows graphics device.

Author(s)

Yihui Xie

References

<http://animation.yihui.name/animation:options>

See Also

[options](#)

Examples

```
## Not run:
# store the old option to restore it later
oopt = ani.options(interval = 0.05, nmax = 100, ani.dev = png, ani.type = "png")
ani.start()
opar = par(mar = c(3, 3, 2, 0.5), mgp = c(2, .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",)
par(opar)
ani.stop()
ani.options(oopt)
## End(Not run)
```

`ani.start`*Start the generation of an HTML animation page*

Description

Copy JavaScript file ‘FUN.js’ and CSS file ‘ANI.css’ to the same directory as the HTML animation page, create a directory ‘images’ and open a graphics device in this directory (the device is specified as `ani.dev` in `ani.options`). The prompt of the current R session is modified (by default ANI>).

Usage

```
ani.start(...)
```

Arguments

... arguments passed to `ani.options` to set animation parameters

Value

None (invisible ‘NULL’).

Note

After calling `ani.start`, either animation functions in this package or R script of your own can be used to generate & save animated pictures using proper graphics devices (specified as `ani.dev` in `ani.options`), then watch your animation by `ani.stop()`.

Note that former image files in the directory ‘images’ will be removed.

Author(s)

Yihui Xie

References

http://animation.yihui.name/animation:create_html_animation_page

See Also

`ani.options`, `ani.stop`

Examples

```
## Not run:  
  
# save the animation in HTML pages and auto-browse it  
ani.options(ani.width = 600, ani.height = 500, interval = 0.2)  
ani.start()  
boot.iid()  
ani.stop()  
  
## End(Not run)
```

ani.stop

Write the HTML animation page

Description

Write the HTML animation page, restore previous options such as `prompt` and close the graphical device opened in `ani.start`.

Usage

```
ani.stop()
```

Value

None (invisible ‘NULL’); a string will be printed in the console indicating where is the HTML file.

Note

The content of the HTML file completely depends on the parameters set in `ani.options`.

Author(s)

Yihui Xie

References

http://animation.yihui.name/animation:create_html_animation_page

See Also

`ani.options`, `ani.start`

Examples

```
## Not run:

# save the animation in HTML pages and auto-browse it
ani.options(ani.width = 600, ani.height = 500, interval = 0.2)
ani.start()
boot.iid()
ani.stop()

## End(Not run)
```

animation-package *Statistical Animations Using R*

Description

Various functions for animations in statistics which could probably aid in teaching statistics and data analysis.

Details

```
Package: animation
Type: Package
Version: 1.0
Date: 2008-06-20
License: GPL-2 | GPL-3
```

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/MPG animations are also available if necessary software has been installed.

Author(s)

Yihui Xie <<http://www.yihui.name>>

References

AniWiki: Animations in Statistics <http://animation.yihui.name>; created and maintained by Yihui Xie

Examples

```
## Not run:
#####
# (1) Animations in HTML pages
# create an animation page in the tempdir() and auto-browse it
# Brownian Motion
```

```

oopt = ani.options(interval = 0.05, nmax = 100, ani.dev = png,
  ani.type = "png",
  title = "Demonstration of Brownian Motion",
  description = "Random walk on the 2D plane: for each point
    (x, y), x = x + rnorm(1) and y = y + rnorm(1).")
ani.start()
opar = par(mar = c(3, 3, 2, 0.5), mgp = c(2, .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",)
par(opar)
ani.stop()
ani.options(oopt)

#####
# (2) Animations inside R windows graphics devices
# Bootstrapping
oopt = ani.options(interval = 0.3, nmax = 50)
boot.iid()
ani.options(oopt)

#####
# (3) GIF animations
oopt = ani.options(interval = 0, nmax = 100)
saveMovie(brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow"),
  interval = 0.05, outdir = getwd(), width = 600, height = 600)
ani.options(oopt)

#####
# (4) Flash animations
oopt = ani.options(nmax = 100, interval = 0)
saveSWF(buffon.needle(type = "S"), para = list(mar = c(3, 2.5, 1, 0.2),
  pch = 20, mgp = c(1.5, 0.5, 0)), dev = "pdf", swfname = "buffon.swf",
  outdir = getwd(), interval = 0.1)
ani.options(oopt)
## End(Not run)

```

bisection.method *Demonstration of the Bisection Method for Root-finding on an Interval*

Description

In mathematics, 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. This function gives a visual demonstration of this process of finding the root of an equation $f(x) = 0$.

Usage

```

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

```

Arguments

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

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

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

Author(s)

Yihui Xie

References

http://en.wikipedia.org/wiki/Bisection_method
http://animation.yihui.name/compstat:bisection_method

See Also

`deriv`, `uniroot`

Examples

```

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

## Not run:

# 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 end-points
bisection.method(f, c(-3, 5), interact = TRUE)

# HTML animation pages
ani.start(nmax = 50, ani.height = 400, ani.width = 600, interval = 1,
         title = "The Bisection Method for Root-finding on an Interval",
         description = "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.")
par(mar = c(4, 4, 1, 2))
bisection.method(main = "")
ani.stop()

## End(Not run)

```

boot.iid

Bootstrapping the i.i.d data

Description

Demonstrate bootstrapping for i.i.d data: use a sunflower scatter plot to illustrate the results of sampling, 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 = c("Bootstrapping the i.i.d data", "Density of bootstrap estimates"),
        ...)

```

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 (<i>m</i> -out-of- <i>n</i> bootstrap)
mat, widths, heights	arguments passed to <code>layout</code> 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 <code>sunflowerplot</code>

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 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'.

Author(s)

Yihui Xie

References

There are many references explaining the bootstrap and its variations. For a relatively complete one, you may just refer to:

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

http://animation.yihui.name/dmml:bootstrap_i.i.d

See Also

`sunflowerplot`

Examples

```

# bootstrap for 20 random numbers from U(0, 1)
opar = 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(interval = 0.5, nmax = 40)
# 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))

par(opar)

## Not run:

# save the animation in HTML pages
ani.options(ani.height = 500, ani.width = 600, outdir = getwd(),
           title = "Bootstrapping the i.i.d data",
           description = "This is a naive version of bootstrapping but
           may be useful for novices.")
ani.start()
par(mar = c(2.5, 4, 0.5, 0.5))
boot.iid(main = c("", ""), heights = c(1, 2))
ani.stop()

## End(Not run)

ani.options(oopt)

```

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

Description

Demonstrate Brownian motion (random walk) in a 2D scatterplot.

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 appearance of the scatterplot (title, points, etc), see points for details.
...	other arguments passed to plot.default

Details

Brownian motion, or random walk, can be regarded as the trace of some cumulative normal random numbers: the location of the next step is just “current location + random Gaussian numbers”, i.e.,

$$x_{k+1} = x_k + rnorm(1); \quad y_{k+1} = y_k + rnorm(1)$$

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

Value

None (invisible ‘NULL’).

Author(s)

Yihui Xie

References

http://animation.yihui.name/prob:brownian_motion

See Also

[rnorm](#)

Examples

```
# show an animation in (Windows/X Window...) a graphics device
# unless you have opened an invisible device like png(), pdf(), ...
oopt = ani.options(interval = 0.05, nmax = 150)
brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow",
  main = "Demonstration of Brownian Motion")
ani.options(oopt)

## Not run:
# create an HTML animation page
# store the old option to restore it later
oopt = ani.options(interval = 0.05, nmax = 100, ani.dev = png,
  ani.type = "png",
  title = "Demonstration of Brownian Motion",
  description = "Random walk on the 2D plane: for each point
  (x, y), x = x + rnorm(1) and y = y + rnorm(1).")
ani.start()
opar = par(mar = c(3, 3, 1, 0.5), mgp = c(2, .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")
par(opar)
ani.stop()
## End(Not run)
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. '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.

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 <code>layout</code> 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 <code>ylim</code> will be $(1 \pm \text{expand}) * \pi$.
type	an argument passed to <code>plot</code> when plotting the estimated π values (default to be lines).
...	other arguments passed to <code>plot</code> 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>.

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 dropping.

Value

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

Note

Note that `redraw` will affect the speed of the simulation (animation) to a great deal 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.

Author(s)

Yihui Xie

References

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

http://animation.yihui.name/prob:buffon_s_needle

Examples

```
# it takes several seconds if 'redraw = TRUE'
oopt = ani.options(nmax = 500, interval = 0)
opar = 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)

par(opar)
## Not run:

# create HTML animation page
ani.options(nmax = 100, interval = 0.1, ani.height = 500, ani.width = 600,
  outdir = getwd(), title = "Simulation of Buffon's Needle",
  description = "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.start()
par(mar = c(3, 2.5, 1, 0.2), pch = 20, mgp = c(1.5, 0.5, 0))
buffon.needle(type = "S")
ani.stop()

## End(Not run)

ani.options(oopt)
```

`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 $X_j, j = 1, 2, \dots, n$, and $n = 1, 2, \dots, nmax$, then the sample means are computed, and at last the density of these sample means is plotted as the sample size n increases, 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, col = c("bisque", "red", "black"),
       mat = matrix(1:2, 2), widths = rep(1, ncol(mat)),
       heights = rep(1, nrow(mat)), ...)
```

Arguments

<code>obs</code>	the number of sample points to be generated from the distribution
<code>FUN</code>	the function to generate n random numbers from a certain distribution
<code>col</code>	a vector of length 2 specifying the colors of the histogram and the density line
<code>mat, widths, heights</code>	arguments passed to <code>layout</code> to set the layout of the two graphs.
<code>...</code>	other arguments passed to <code>hist</code>

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

None.

Author(s)

Yihui Xie

References

E. L. Lehmann, *Elements of Large-Sample Theory*. Springer-Verlag, New York, 1999.

http://animation.yihui.name/prob:central_limit_theorem

See Also[hist](#), [density](#)**Examples**

```

oopt = ani.options(interval = 0.1, nmax = 150)
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)

## Not run:

# HTML animation page
ani.options(ani.height = 500, ani.width = 600, outdir = getwd(), nmax = 100,
            interval = 0.1, title = "Demonstration of the Central Limit Theorem",
            description = "This animation shows the distribution of the sample
            mean as the sample size grows.")
ani.start()
par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
clt.ani(type = "h")
ani.stop()
## End(Not run)
ani.options(oopt)

# other distributions: Chi-square with df = 5
f = function(n) rchisq(n, 5)
clt.ani(FUN = f)

```

conf.int

Demonstration of Confidence Intervals

Description

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

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 $N(0, 1)$

`c1` two different colors to annotate whether the confidence intervals cover the true mean (`c1[1]`: yes; `c1[2]`: no)
`...` other arguments passed to `plot`

Details

Intervals that cover the true parameter are denoted in color `c1[2]`, otherwise in color `c1[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

<code>level</code>	confidence level
<code>size</code>	sample size
<code>CI</code>	a matrix of confidence intervals for each sample
<code>CR</code>	coverage rate

Author(s)

Yihui Xie

References

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

http://animation.yihui.name/mathstat:confidence_interval

Examples

```
oopt = ani.options(interval = 0.1, nmax = 100)
# 90% interval
conf.int(0.90, main = "Demonstration of Confidence Intervals")

## Not run:

# save the animation in HTML pages
ani.options(ani.height = 400, ani.width = 600, outdir = getwd(), nmax = 100,
  interval = 0.15, title = "Demonstration of Confidence Intervals",
  description = "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.start()
par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
conf.int()
ani.stop()
```

```
## End(Not run)
ani.options(oopt)
```

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

Description

Simply 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. This function uses rectangles to illustrate these ‘parts’ and mark the test set & the training set with different colors.

Usage

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

Arguments

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

Details

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 `nmax` and `k`.

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

Author(s)

Yihui Xie

References

http://animation.yihui.name/dmml:k-fold_cross-validation

See Also

[kfcv](#)

Examples

```

oopt = ani.options(interval = 2, nmax = 10)
cv.ani(main = "Demonstration of the k-fold Cross Validation", bty = "l")

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

## Not run:

# save the animation in HTML pages
ani.options(ani.height = 400, ani.width = 600, outdir = getwd(), interval = 2,
  nmax = 10, title = "Demonstration of the k-fold Cross Validation",
  description = "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.start()
par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0), tcl = -0.3)
cv.ani(bty = "l")
ani.stop()

## End(Not run)
ani.options(oopt)

```

cv.nfeaturesLDA	<i>Cross-validation to find the optimum number of features (variables) in LDA</i>
-----------------	---

Description

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

Details

The procedure is like this:

- Split the whole data randomly into k folds:
 - For the number of features $g = 1, 2, \dots, g_{max}$, choose g features that have the largest discriminatory power (measured by the F-statistic in ANOVA):
 - * For the fold i ($i = 1, 2, \dots, k$):
 - 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")`.

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

References

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

http://animation.yihui.name/da:biostat:select_features_via_cv

See Also

[kfcv](#), [cv.ani](#), [lda](#)

Examples

```

op = par(pch = 19, mar = c(3, 3, 0.2, 0.7), mgp = c(1.5, 0.5, 0))
cv.nfeaturesLDA()
par(op)

## Not run:

# save the animation in HTML pages
oopt = ani.options(ani.height = 480, ani.width = 600, outdir = getwd(),
  interval = 0.5, nmax = 10,
  title = "Cross-validation to find the optimum number of features in LDA",
  description = "This animation has provided an illustration of the process of
  finding out the optimum number of variables using k-fold cross-validation
  in a linear discriminant analysis (LDA).")
ani.start()
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()
ani.stop()
ani.options(oopt)

## End(Not run)

```

 flip.coin

Probability in flipping coins

Description

In the first class of learning probability theory, we usually begin with flipping coins or tossing dice. This function provides a simulation to such a process and computes the frequency 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 `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://animation.yihui.name/prob:flipping_coins

See Also

[ani.start](#), [ani.stop](#)

Examples

```
oopt = ani.options(interval = 0.2, nmax = 100)
# 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")
```

```
## Not run:

# HTML animation page
ani.options(ani.height = 500, ani.width = 600, outdir = getwd(), interval = 0.2,
            nmax = 50, title = "Probability in flipping coins",
            description = "This animation has provided a simulation of flipping coins,
            which might be helpful in understanding the concept of probability.")
ani.start()
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))
ani.stop()
## End(Not run)
ani.options(oopt)
```

grad.desc

Gradient Descent Algorithm for the 2D Case

Description

This function has provided 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, len = 50,
          interact = FALSE)
```

Arguments

FUN	the objective function to be minimized; contains only two independent variables (variable names do not need to be 'x' and 'y')
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 $F(x_{i+1})$
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?

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 (`nmax` in `control`) has not been reached.

Value

A list containing

<code>par</code>	the solution for the local minimum
<code>value</code>	the value of the objective function corresponding to <code>par</code>
<code>iter</code>	the number of iterations; if it is equal to <code>control\$nmax</code> , it's quite likely that the solution is not reliable because the maximum number of iterations has been reached
<code>gradient</code>	the gradient function of the objective function; it is returned by <code>deriv</code>
<code>persp</code>	a function to make the perspective plot of the objective function; can accept further arguments from <code>persp</code> (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` (see the help file)!

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

Author(s)

Yihui Xie

References

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

http://animation.yihui.name/compstat:gradient_descent_algorithm

See Also

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

Examples

```

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

## Not run:

# 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$persp(col = "lightblue", theta = 30, phi = 30)
# or
ani.options(interval = 0, nmax = 200)
f2 = function(x, y) sin(1/2 * x^2 - 1/4 * y^2 + 3) *
  cos(2 * x + 1 - exp(y))
xx = grad.desc(f2, c(-2, -2, 2, 2), c(-1, 0.5),
  gamma = 0.1, tol = 1e-04)
# click your mouse to select a start point
xx = grad.desc(f2, c(-2, -2, 2, 2), interact = TRUE,
  tol = 1e-04)
xx$persp(col = "lightblue", theta = 30, phi = 30)

# HTML animation pages
ani.options(ani.height = 500, ani.width = 500, outdir = getwd(), interval = 0.3,
  nmax = 50, title = "Demonstration of the Gradient Descent Algorithm",
  description = "The arrows will take you to the optimum step by step.")
ani.start()
grad.desc()
ani.stop()

## End(Not run)
ani.options(oopt)

```

highlight.def

Create R definition file for the software Highlight

Description

The default definition file for R in Highlight is somewhat incomplete, and this function is to dynamically generate such a file according to packages in the search path.

Usage

```
highlight.def(file = "r.lang")
```

Arguments

file the path of the output definition file.

Details

First all the functions are listed out by `ls`; then some constants and operators are removed from this long list; at last these characters are written into the ‘file’.

Value

None.

Author(s)

Yihui Xie

References

Highlight by Andre Simon: <http://www.andre-simon.de/>

See Also

`ls`, `cat`

Examples

```
# generate the definition file in getwd()
highlight.def()

# include functions in package 'animation'
library(animation)
highlight.def()
```

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

Description

Compute sample sizes for k -fold cross-validation.

Usage

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

Arguments

<code>k</code>	number of groups.
<code>N</code>	total sample size.

Details

If N/k is an integer, the sample sizes are k ' N/k 's ($N/k, N/k, \dots$), 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 K-Means Cluster Algorithm

Description

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

Usage

```
kmeans.ani(x = matrix(runif(100), ncol = 2,
  dimnames = list(NULL, c("X1", "X2"))), centers = 3, pch = 1:3,
  col = 1:3, hints = c("Move centers!", "Find cluster?"))
```

Arguments

<code>x</code>	A numerical 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 <i>only</i> 2 columns.
<code>centers</code>	Either the number of clusters or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in <code>x</code> is chosen as the initial centres.

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.
hints	Two text strings indicating the steps of k-means clustering: move the center or find the cluster membership?

Details

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

For practical applications please refer to [kmeans](#).

Note that `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 $2 * nmax$ 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

References

Hartigan, J. A. and Wong, M. A. (1979). A K-means clustering algorithm. *Applied Statistics* **28**, 100-108.

http://animation.yihui.name/mvstat:k-means_cluster_algorithm

See Also

[kmeans](#)

Examples

```
#set larger 'interval' if the speed is too fast
oopt = ani.options(interval = 2, nmax = 50)
op = par(mar = c(3, 3, 1, 1.5), mgp = c(1.5, 0.5, 0))
kmeans.ani()

ani.options(nmax = 50)
# 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?
ani.options(nmax = 50)
kmeans.ani(x, centers = 3)

par(op)
## Not run:

# create HTML animation page
ani.options(ani.height = 480, ani.width = 480, outdir = getwd(), interval = 2,
           nmax = 50, title = "Demonstration of the K-means Cluster Algorithm",
           description = "Move! Average! Cluster! Move! Average! Cluster! ...")
ani.start()
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)
ani.stop()

## End(Not run)
ani.options(oopt)

```

knn.ani

Demonstrate kNN classification algorithm on the 2D plane

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 <i>ignored</i> if <code>interact = TRUE</code> ; see <code>interact</code> below.
cl	factor of true classifications of training set
k	number of neighbours considered.

<code>interact</code>	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 <code>test</code> .
<code>tt.col</code>	a vector of length 2 specifying the colors for the training data and test data.
<code>cl.pch</code>	a vector specifying symbols for each class
<code>dist.lty, dist.col</code>	the line type and color to annotate the distances
<code>knn.col</code>	the color to annotate the k-nearest neighbour points using a polygon

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 $4 * \min(\text{ani.options}("nmax"), \text{nrow}(\text{test}))$ 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.
http://animation.yihui.name/dmml:k-nearest_neighbour_algorithm

See Also

[knn](#)

Examples

```
## a binary classification problem
oopt = ani.options(interval = 2, nmax = 10)
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)

## Not run:
# an interactive demo: choose the test set by mouse-clicking
ani.options(nmax = 5)
knn.ani(interact = TRUE)

ani.options(ani.height = 500, ani.width = 600, outdir = getwd(), nmax = 10,
           interval = 2, title = "Demonstration for kNN Classification",
           description = "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.start()
par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0))
knn.ani()
ani.stop()

## End(Not run)
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

<code>FUN</code>	a function to generate random numbers from a certain distribution: <code>function(n, mu)</code>
<code>mu</code>	population mean; passed to <code>FUN</code>
<code>np</code>	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
<code>pch</code>	symbols for points; see Details
<code>col.poly</code>	the color of the polygon to annotate the range of sample means
<code>col.mu</code>	the color of the horizontal line which denotes the population mean
<code>...</code>	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

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

http://animation.yihui.name/prob:law_of_large_numbers

Examples

```
oopt = ani.options(interval = 0.01, nmax = 150)
lln.ani(pch = ".")

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

## Not run:
```

```
# save the animation in HTML pages
ani.options(ani.height = 480, ani.width = 600, outdir = getwd(), nmax = 100,
  interval = 0.1, title = "Demonstration of the Law of Large Numbers",
  description = "The sample mean approaches to the population mean as
  the sample size n grows.")
ani.start()
par(mar = c(3, 3, 1, 0.5), mgp = c(1.5, 0.5, 0))
ltn.ani(cex = 0.6)
ani.stop()

## End(Not run)
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 $\sin(\text{seq}(0, 2 * \pi, \text{length} = 50)) + \text{rnorm}(50, \text{sd} = 0.2)$
k	an integer of the window width
conf	a positive number: the confidence intervals are computed as $[\text{ar1} - \text{conf} * \text{s.e.}, \text{ar1} + \text{conf} * \text{s.e.}]$
mat, widths, heights	arguments passed to layout to divide the device into 2 parts
lty.rect	the line type of the rectangles representing 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.

http://animation.yihui.name/ts:moving_window_ar

See Also

[arima](#)

Examples

```
# moving window along a sin curve
oopt = ani.options(interval = 0.1, nmax = 50)
op = 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'
ani.options(interval = 0.1, nmax = 50)
data(pageview)
mwar.ani(pageview$visits, k = 30)

par(op)
## Not run:

# HTML animation page
ani.options(ani.height = 500, ani.width = 600, outdir = getwd(), nmax = 50,
  title = "Demonstration of Moving Window Auto-Regression",
  description = "Compute the AR(1) coefficient for the data in the
  window and plot the confidence intervals. Repeat this step as the
  window moves.")
ani.start()
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')
ani.stop()

## End(Not run)
ani.options(oopt)
```

 newton.method

Demonstration of the Newton-Raphson Method for Root-finding

Description

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 $f(x)$. 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

<code>FUN</code>	the function in the equation to solve (univariate)
<code>init</code>	the starting point
<code>rg</code>	the range for plotting the curve
<code>tol</code>	the desired accuracy (convergence tolerance)
<code>interact</code>	logical; whether choose the starting point by clicking on the curve (for 1 time) directly?
<code>col.lp</code>	a vector of length 3 specifying the colors of: vertical lines, tangent lines and points
<code>main, xlab, ylab</code>	titles of the plot; there are default values for them (depending on the form of the function <code>FUN</code>)
<code>...</code>	other arguments passed to curve

Details

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, \dots ; 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 control\$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 = 50)
op = 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))

## Not run:

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

```

```

2), interact = TRUE)

# HTML animation pages
ani.options(ani.height = 500, ani.width = 600, outdir = getwd(), nmax = 100,
  interval = 1, title = "Demonstration of the Newton-Raphson Method",
  description = "Go along with the tangent lines and iterate.")
ani.start()
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 = "")
ani.stop()

## End(Not run)
par(op)
ani.options(oopt)

```

pageview

Data of page view from Sep 21, 2007 for Yihui's website

Description

Page view data for Yihui's website from Sep 21, 2007 to a recent date.

Usage

```
data(pageview)
```

Format

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

day Date starts from Sep 21, 2007 to a recent date.

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).

Details

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

Source

```

http://www.yihui.name/cgi-bin/awstats/awstats.pl?month=10&year=2007&output=main&config=yihuiname&framename=index

```

Examples

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

sample.cluster

Demonstration for cluster sampling

Description

Every 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 <code>rect</code> to annotate the “clusters”

Value

None (invisible ‘NULL’).

Author(s)

Yihui Xie

References

Cochran, W G (1977) *Sampling Techniques*, Wiley, ISBN 0-471-16240-X
http://animation.yihui.name/samp:cluster_sampling

See Also

[sample](#)

Examples

```

oopt = ani.options(interval = 1, nmax = 30)
op = par(mar = rep(1, 4))
sample.cluster(col = c("bisque", "white"))
par(op)
## Not run:

# HTML animation page
ani.options(ani.height = 350, ani.width = 500, outdir = getwd(), nmax = 30,
            interval = 1, title = "Demonstration of the cluster sampling",
            description = "Once a cluster is sampled, all its elements will be
            chosen.")
ani.start()
par(mar = rep(1, 4), lwd = 2)
sample.cluster(col = c("bisque", "white"))
ani.stop()

## End(Not run)
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

ReferencesCochran, W G (1977) *Sampling Techniques*, Wiley, ISBN 0-471-16240-X<http://animation.yihui.name/samp:srswr>**See Also**[sample](#)**Examples**

```

oopt = ani.options(interval = 1, nmax = 30)
op = par(mar = rep(1, 4))
sample.simple()
par(op)
## Not run:

# HTML animation page
ani.options(ani.height = 350, ani.width = 500, outdir = getwd(), nmax = 30,
            interval = 1,
            title = "Demonstration of the simple random sampling without replacement",
            description = "Each member of the population has an equal and known chance
              of being selected.")
ani.start()
par(mar = rep(1, 4), lwd = 2)
sample.simple()
ani.stop()

## End(Not run)
ani.options(oopt)

```

sample.strat

*Demonstration for stratified sampling***Description**

Every 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

References

Cochran, W G (1977) *Sampling Techniques*, Wiley, ISBN 0-471-16240-X

See Also

[sample](#)

Examples

```
oopt = ani.options(interval = 1, nmax = 30)
op = par(mar = rep(1, 4), lwd = 2)
sample.strat(col = c("bisque", "white"))
par(op)
## Not run:

# HTML animation page
ani.options(ani.height = 350, ani.width = 500, outdir = getwd(), nmax = 30,
  interval = 1, title = "Demonstration of the stratified sampling",
  description = "Every rectangle stands for a stratum, and the simple
  random sampling without replacement is performed within each stratum.")
ani.start()
par(mar = rep(1, 4), lwd = 2)
sample.strat(col = c("bisque", "white"))
ani.stop()

## End(Not run)
ani.options(oopt)
```

sample.system *Demonstration for 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

References

Cochran, W G (1977) *Sampling Techniques*, Wiley, ISBN 0-471-16240-X

See Also

[sample](#)

Examples

```
oopt = ani.options(interval = 1, nmax = 30)  
op = par(mar = rep(1, 4), lwd = 2)  
sample.system()  
par(op)  
## Not run:  
  
# HTML animation page
```

```

ani.options(ani.height = 350, ani.width = 500, outdir = getwd(), nmax = 30,
            interval = 1, title = "Demonstration of the systematic sampling",
            description = "Sampling with equal distances.")
ani.start()
par(mar = rep(1, 4), lwd = 2)
sample.system()
ani.stop()

## End(Not run)
ani.options(oopt)

```

saveMovie

Convert Images to A Single Animated Movie

Description

This function opens a graphical device 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 (in formats such as GIF and MPG, etc).

Usage

```

saveMovie(expr, interval = 1, moviename = "movie", movietype = "gif",
          loop = 0, dev = png, filename = "Rplot", fmt = "%03d",
          outdir = tempdir(), para = par(no.readonly = TRUE), ...)

```

Arguments

<code>expr</code>	an expression to generate animations; use either the animation functions (e.g. <code>brownian.motion()</code>) in this package or a custom expression (e.g. <code>for(i in 1:10) plot(runif(10), ylim = 0:1)</code>).
<code>interval</code>	duration between animation frames (unit in seconds)
<code>moviename</code>	file name of the movie (base only, without extension)
<code>movietype</code>	format of the movie (‘gif’, ‘mpg’, ...; as long as it’s supported by ImageMagick)
<code>loop</code>	iterations of the movie; set iterations to zero to repeat the animation an infinite number of times, otherwise the animation repeats itself up to <code>loop</code> times (N.B. for GIF only!)
<code>dev</code>	a function for a graphical device such as <code>png</code> , <code>jpeg</code> and <code>bmp</code> , etc.
<code>filename</code>	file name of the sequence of images (‘pure’ name; without any format or extension)
<code>fmt</code>	a C-style string formatting command, such as <code>%3d</code>
<code>outdir</code>	the directory for the movie frames and the movie itself
<code>para</code>	a list: the graphics parameters to be set before plotting; passed to <code>par</code>
<code>...</code>	other arguments passed to the graphical device, such as <code>height</code> and <code>width</code> , ...

Details

The convenience of this function is that it can create a single movie file, however, two drawbacks are obvious too: (1) we need a special (free) software ImageMagick; (2) the speed of the animation cannot be conveniently controlled, as we have specified a fixed `interval`. So just go ahead with this function to create your movies if you don't mind these two points.

Value

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

Note

Please make sure ImageMagick has been installed in your system: <http://www.imagemagick.org>

Author(s)

Yihui Xie

References

<http://www.imagemagick.org/script/convert.php>

<http://animation.yihui.name/animation:start>

See Also

[saveSWF](#), [system](#), [png](#), [jpeg](#) and [bmp](#)

Examples

```
## make sure ImageMagick has been installed in your system
## Not run:

saveMovie(for(i in 1:10) plot(runif(10), ylim = 0:1), loop = 1)
oopt = ani.options(interval = 0.05, nmax = 100)
saveMovie(brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow"),
          width = 600, height = 600)
ani.options(oopt)
## End(Not run)
```

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 ‘SWF Tools’ (`png2swf`, `jpeg2swf`, `pdf2swf`) to convert these images to a single Flash animation.

Usage

```
saveSWF(expr, interval = 1, swfname = "movie.swf",
        dev = c("png", "jpeg", "pdf"), filename = "Rplot", fmt = "%03d",
        outdir = tempdir(), swftools = NULL, para = par(no.readonly = TRUE), ...)
```

Arguments

<code>expr</code>	an expression to generate animations; use either the animation functions (e.g. <code>brownian.motion()</code>) in this package or a custom expression (e.g. <code>for(i in 1:10) plot(runif(10), ylim = 0:1)</code>).
<code>interval</code>	duration between animation frames (unit in seconds)
<code>swfname</code>	file name of the Flash file
<code>dev</code>	character: the graphics device to be used. Three choices are available: png , jpeg and pdf , etc.
<code>filename</code>	file name of the sequence of images (‘pure’ name; without any format or extension)
<code>fmt</code>	a C-style string formatting command, such as ‘%3d’
<code>outdir</code>	the directory for the animation frames and the Flash file
<code>swftools</code>	the path of ‘SWF Tools’, e.g. ‘C:/swftools’. This argument is to make sure that <code>png2swf</code> , <code>jpeg2swf</code> and <code>pdf2swf</code> can be executed correctly. If it is <code>NULL</code> , it should be guaranteed that these commands can be executed without the path.
<code>para</code>	a list: the graphics parameters to be set before plotting; passed to <code>par</code>
<code>...</code>	other arguments passed to the graphical device, such as <code>height</code> and <code>width</code> , ...

Value

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

Note

Please download the SWF Tools before using this function: <http://www.swftools.org>

Author(s)

Yihui Xie

References

http://animation.yihui.name/animation:start#create_flash_animations

See Also

[saveMovie](#), [system](#), [png](#), [jpeg](#), [pdf](#)

Examples

```
## Not run:
oopt = ani.options(interval = 0, nmax = 50)
# from png
saveSWF(knn.ani(test = matrix(rnorm(16), ncol = 2),
  cl.pch = c(16, 2)), 1.5, dev = "png", para = list(mar = c(3,
  3, 1, 1.5), mgp = c(1.5, 0.5, 0)), swfname = "kNN.swf")

# from pdf (vector plot!)
ani.options(interval = 0, nmax = 50)
saveSWF(brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow"),
  0.2, "brownian.swf", "pdf", fmt = "")

ani.options(oopt)
## End(Not run)
```

tidy.source

'Tidy up' R code

Description

Actually this function has nothing to do with code optimization; it just returns parsed source code.

Usage

```
tidy.source(source = "clipboard", ...)
```

Arguments

source	a string: location of the source code
...	other arguments passed to cat , e.g. file

Details

This function helps the user to tidy up his source code in a sense that necessary indents and spaces will be added, etc. See [parse](#).

Value

None (invisible 'NULL'). 'Clean' code will be returned to the console unless the output is redirected by 'sink'.

Author(s)

Yihui Xie

Referenceshttp://animation.yihui.name/animation:misc#tidy_up_r_source**See Also**[parse](#), [cat](#)**Examples**

```
## tidy up the source code of image demo
x = file.path(system.file(package = "graphics"), "demo", "image.R")
# to console
tidy.source(x)
# to a file
tidy.source(x, file = tempfile())
## check the original code here and see the difference
file.show(x)

## if you've copied R code into the clipboard
## Not run:
tidy.source("clipboard")
## End(Not run)
```

vi.grid.illusion *Visual Illusions: Scintillating grid illusion and Hermann grid illusion*

Description

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

<code>col</code>	color for grid lines
<code>type</code>	type of illusions: 's' for Scintillating grid illusions and 'h' for Hermann grid illusions

Details

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

None.

Note

In fact there isn't any animation!

Author(s)

Yihui Xie

References

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

http://animation.yihui.name/animation:misc#visual_illusions

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

None.

Note

In fact, points in the original version of ‘Lilac Chaser’ are *blurred*, which is not implemented in this function. If you have any idea, please contact me.

Author(s)

Yihui Xie

References

http://en.wikipedia.org/wiki/Lilac_chaser
http://animation.yihui.name/animation:misc#lilac_chaser

See Also

[points](#)

Examples

```

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

## Not run:

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

## End(Not run)
par(op)
ani.options(oopt)

```

write.rss

Create RSS feed from a CSV data file

Description

An RSS feed is essentially just an XML file, thus the creation is easy just with `cat` to write some tags into a text file. The elements of an item in an RSS feed usually contains 'title', 'link', 'author', 'description', 'pubDate', 'guid', and 'category', etc, which are stored in the CSV data file.

Usage

```

write.rss(file = "feed.xml", entry = "rss.csv", xmlver = "1.0",
  rssver = "2.0", title = "What's New?",
  link = "http://R.yihui.name",
  description = "Animated Statistics Using R",
  language = "en-us", copyright = "Copyright 2007, Yihui Xie",
  pubDate = Sys.time(), lastBuildDate = Sys.time(),
  docs = "http://R.yihui.name",
  generator = "Function write.rss() in R package animation",
  managingEditor = "xieyihui[at]gmail.com",
  webMaster = "xieyihui[at]gmail.com",
  maxitem = 10, ...)

```

Arguments

`file` the path of the output file (RSS feed); passed to `cat`

entry	the input CSV file, containing elements for items in the RSS feed (with tag names in the header); read.csv
xmlver	version of XML
rssver	version of RSS
title	The name of the channel. It's how people refer to your service. If you have an HTML website that contains the same information as your RSS file, the title of your channel should be the same as the title of your website.
link	The URL to the HTML website corresponding to the channel.
description	Phrase or sentence describing the channel.
language	The language the channel is written in.
copyright	Copyright notice for content in the channel.
pubDate	The publication date for the content in the channel.
lastBuildDate	The last time the content of the channel changed.
docs	A URL that points to the documentation for the format used in the RSS file.
generator	A string indicating the program used to generate the channel.
managingEditor	Email address for person responsible for editorial content.
webMaster	Email address for person responsible for technical issues relating to channel.
maxitem	Maximum number of items to be written into the feed.
...	other elements for the channel, e.g. image, cloud, etc.

Details

The items of the RSS feed are stored in the file 'entry', and the many arguments above are just for the channel information.

Value

None. Only a message indicating where the RSS was created.

Note

As the argument `file` is passed to `cat`, you may specify it as an empty string "" so that the result will be printed to the standard output connection, the console unless redirected by 'sink'.

Note the order of items in the CSV file: newer items are added to the end of the file. But this order will be *reversed* in the RSS file!

Author(s)

Yihui Xie

References

Read <http://cyber.law.harvard.edu/rss/rss.html> for the specification of RSS.

See Also

[cat](#), [read.csv](#)

Examples

```
# create rss feed from a sample file in 'animation'  
# to getwd()  
write.rss(entry = system.file("js", "rss.csv", package = "animation"))  
  
## Not run:  
  
# Read entries from the internet  
write.rss(entry = "http://r.yihui.name/news/rss.csv")  
  
## End(Not run)
```

Index

*Topic **IO**

ani.news, 1
highlight.def, 25
tidy.source, 46
write.rss, 50

*Topic **arith**

kfcv, 26

*Topic **classif**

cv.ani, 18
cv.nfeaturesLDA, 19
kfcv, 26
knn.ani, 29

*Topic **cluster**

kmeans.ani, 27

*Topic **datasets**

pageview, 37

*Topic **device**

animation-package, 6
saveMovie, 43
saveSWF, 45

*Topic **distribution**

clt.ani, 15
conf.int, 16
flip.coin, 21
lln.ani, 31
sample.cluster, 38
sample.simple, 39
sample.strat, 40
sample.system, 42

*Topic **dplot**

animation-package, 6
bisection.method, 7
boot.iid, 9
brownian.motion, 11
buffon.needle, 13
clt.ani, 15
conf.int, 16
cv.nfeaturesLDA, 19
flip.coin, 21

grad.desc, 23

lln.ani, 31

mwar.ani, 33

newton.method, 35

*Topic **dynamic**

animation-package, 6

bisection.method, 7

boot.iid, 9

brownian.motion, 11

buffon.needle, 13

clt.ani, 15

conf.int, 16

cv.ani, 18

cv.nfeaturesLDA, 19

flip.coin, 21

grad.desc, 23

kmeans.ani, 27

knn.ani, 29

lln.ani, 31

mwar.ani, 33

newton.method, 35

sample.cluster, 38

sample.simple, 39

sample.strat, 40

sample.system, 42

saveMovie, 43

saveSWF, 45

vi.grid.illusion, 47

vi.lilac.chaser, 49

*Topic **hplot**

buffon.needle, 13

cv.ani, 18

flip.coin, 21

kmeans.ani, 27

knn.ani, 29

*Topic **iplot**

knn.ani, 29

*Topic **math**

buffon.needle, 13

- *Topic **misc**
 - ani.options, 2
 - write.rss, 50
 - *Topic **multivariate**
 - cv.nfeaturesLDA, 19
 - kmeans.ani, 27
 - *Topic **nonparametric**
 - boot.iid, 9
 - *Topic **optimize**
 - bisection.method, 7
 - grad.desc, 23
 - newton.method, 35
 - *Topic **package**
 - animation-package, 6
 - *Topic **ts**
 - mwar.ani, 33
 - *Topic **utilities**
 - ani.start, 4
 - ani.stop, 5
 - saveMovie, 43
 - saveSWF, 45
- abline, 48
- ani.news, 1
- ani.options, 2, 2, 4–6, 14, 15, 17, 24, 32
- ani.start, 3, 4, 4–6, 22
- ani.stop, 3, 4, 5, 5, 22
- animation-package, 6
- arma, 33, 34
- bisection.method, 7
- bmp, 43, 44
- boot.iid, 3, 9
- brownian.motion, 3, 11
- buffon.needle, 3, 13
- cat, 26, 46, 47, 50–52
- clt.ani, 15
- conf.int, 16
- contour, 24
- curve, 8, 35
- cv.ani, 3, 18, 20, 27
- cv.nfeaturesLDA, 19
- density, 16
- deriv, 9, 24
- file.show, 2
- flip.coin, 3, 21
- grad.desc, 23
- highlight.def, 25
- hist, 15, 16
- jpeg, 3, 43–46
- kfcv, 18–20, 26
- kmeans, 28
- kmeans.ani, 3, 27
- knn, 30
- knn.ani, 3, 29
- layout, 10, 13, 15, 33
- lda, 20
- lln.ani, 31
- ls, 26
- mwar.ani, 33
- newton.method, 35
- optim, 24, 36
- options, 2, 3
- pageview, 37
- par, 43, 45
- parse, 46, 47
- pdf, 45, 46
- persp, 24
- plot, 13, 17, 18
- plot.default, 11
- png, 3, 43–46
- points, 11, 22, 32, 33, 48, 49
- read.csv, 51, 52
- rect, 38, 41
- rnorm, 12
- sample, 38, 40–42
- sample.cluster, 38
- sample.simple, 39
- sample.strat, 40
- sample.system, 42
- saveMovie, 43, 46
- saveSWF, 44, 45
- shapiro.test, 15
- sunflowerplot, 10
- system, 44–46
- tempdir, 3

tidy.source, [46](#)

uniroot, [9](#)

vi.grid.illusion, [47](#)

vi.lilac.chaser, [49](#)

write.rss, [50](#)

x11, [6](#)