Journal of Statistical Software

April 2013, Volume 53, Issue 1. *http://www.jstatsoft.org/*

animation: An R Package for Creating Animations and Demonstrating Statistical Methods

Yihui Xie
Iowa State University

Abstract

Animated graphs that demonstrate statistical ideas and methods can both attract interest and assist understanding. In this paper we first discuss how animations can be related to some statistical topics such as iterative algorithms, random simulations, (re)sampling methods and dynamic trends, then we describe the approaches that may be used to create animations, and give an overview to the R package **animation**, including its design, usage and the statistical topics in the package. With the **animation** package, we can export the animations produced by R into a variety of formats, such as a web page, a GIF animation, a Flash movie, a PDF document, or an MP4/AVI video, so that users can publish the animations fairly easily. The design of this package is flexible enough to be readily incorporated into web applications, e.g., we can generate animations online with **Rweb**, which means we do not even need R to be installed locally to create animations. We will show examples of the use of animations in teaching statistics and in the presentation of statistical reports using `Sweave` or **knitr**. In fact, this paper itself was written with the **knitr** and **animation** package, and the animations are embedded in the PDF document, so that readers can watch the animations in real time when they read the paper (the Adobe Reader is required).

Animations can add insight and interest to traditional static approaches to teaching statistics and reporting, making statistics a more interesting and appealing subject.

Keywords: animation, statistical demonstration, simulation, web application, reproducible research, R.

1. Introduction

The range of statistical methods, models and computer implementations now available is very wide. Both students in statistics, and application area specialists who find that they need to master and use statistical methodology, require effective means of entrance into this extensive body of ideas, theories, and practical computer implementations. Effective visual demonstrations can be useful aids in the mastering of concepts. Statistical analysis uses computer algorithms to process data, with the user then required to interpret results. Even with a good training in mathematics, it can be hard to understand the motivation for a statistical algorithm, or how it works in processing data.

An obvious application is the simulation of limiting distributions. Not only is it possible to illustrate the central limit theorem (CLT); one can show how the CLT fails if the observations come from a population with infinite variance. This can be done moreover, both in a simulation (sampling from a theoretical distribution) and resampling (sampling from an empirical distribution) context. The bootstrap sampling distribution of the mean is a simple use of sampling from an empirical distribution.

Technological advances in data capture and deployment have led to large increases in the size and complexity of data sets. This creates new challenges those who work on the analysis of such data. Often, as for example in the analysis of gene expression data, the challenges have substantial statistical novelty. Consider, as an example, methods that select, on the basis of a measure of discriminatory power, a few genes from among a large group of genes, in order to discriminate between previously assigned groups. Simulation, and associated animations, can be an excellent tool for conveying the process of selection.

Traditional static printed statistical reports have severe limitations, for conveying results with some degree of complexity to an audience that is not trained to understand the statistical ideas. Here, an active animated way to present results can help greatly. The human visual cortex is arguably the most powerful computing system to which we have access. Visualization puts information in a form that uses the power of this computing system. By virtue of our visual system, we may be able to quickly understand a complicated method or result, or at least a simplified case. Wender and Muehlboeck (2003) showed the advantages of animations in teaching statistical topics like least squares and type I & II errors. Velleman and Moore (1996) discussed a set of animation ideas that can be effective for a student to learn concepts actively. Animation can be helpful only if it is used appropriately; Fisher (2010) mentioned several successful applications of animations such as GapMinder (Rosling and Johansson 2009), but also argued animation could be very bad when used poorly – it is usually good for presentation but not for exploration.

The **animation** package (Xie 2013) was designed mainly for two purposes:

1. to demonstrate ideas, concepts, theories and methods in statistics through animations;
2. to provide utilities to export animations to other formats (e.g., GIF, Flash and various video formats) or write animations into other files such as HTML pages and `Sweave` (Leisch 2002) documents;

There has been active research in the area of dynamic and interactive graphics for decades, which is mainly focused on data analysis. The most similar project to the **animation** package in the literature might be the GASP (Globally Accessible Statistical Procedures, available at `http://www.stat.sc.edu/rsrch/gasp/`), which is a very early collection of statistical routines for data analysis and statistical education. Most of these routines were based on Java Applets and HTML forms using CGI to handle user's input. These techniques have become rare nowadays over the Internet, and as we know, it is much more convenient to use R to do statistical computing and graphics than Java today.

In terms of conveying statistical ideas, there are two similar packages to the **animation** package: the **TeachingDemos** package (Snow 2012) and the **rpanel** package (Bowman, Crawford, Alexander, and Bowman 2007), both of which are based on **tcltk** (Dalgaard 2001) to create graphical user interfaces (GUI) for interactive graphics. There are a number of other sophisticated packages focused on building GUI's, e.g., **RGtk2** (Lawrence and Temple Lang 2010) and **gWidgets** (Verzani 2007, 2012). At the same time, several R packages are especially designed for interactive statistical graphics, such as **iplots** (Urbanek and Wichtrey 2011) and **playwith** (Andrews 2010) – basic operations like brushing and identifying can be done in these packages. Moreover, there are also standalone dynamic and interactive graphics systems like **GGobi** (Cook and Swayne 2007) with the corresponding R package **rggobi** (Temple Lang, Swayne, Wickham, and Lawrence 2012). This list of GUI-related packages and dynamic graphics systems is still far from exhaustive, however, we will see significant distinctions between them and the **animation** package. As mentioned before, most of them are focused on data analysis, whereas the **animation** package puts more emphasis on illustrating statistical ideas – these illustrations may or may not involve with data. At least so far the package has been mainly dealing with statistical methods and theories for pedagogical purposes. Another major difference is, the **animation** package has flexible utilities to export R animations to other formats, among which we need to point out two formats especially: the first one is the HTML pages – we can record the animation frames in R and embed images into an HTML page to create an animation; this process does not depend on any third-party software and the output is like a common movie player (with navigation buttons), so it is an ideal tool to record a whole plotting process and demonstrate later to the audience; the other format is the animation in PDF documents, which is essentially created by the LaTeX (Lamport 1994) package **animate** (Grahn 2010). With this approach, we can easily insert animations into `Sweave` or other literate programming documents, producing "reproducible animations" in our reports or papers.

This paper provides both a conceptual framework on how animations can be effectively related to statistical theories and ideas, and an introduction on the approaches to implement animations, in the R language environment (R Development Core Team 2012). Before we move on, we need to point out that this paper was written in a reproducible document compiled by the **knitr** package (Xie 2012) with animations created by the **animation** package automatically; later we will see that we can actually watch the animations in this document with the Adobe Reader. There was an editorial in the *Journal of Computational and Graphical Statistics* (JCGS) by Levine, Tierney, Wickham, Sampson, Cook, and van Dyk (2010), encouraging authors to submit supplements containing animations, interactive 3D graphics and movies, and finally "embed these dynamic graphics in the online articles themselves". Apparently the **animation** package can be a helpful tool for these authors.

2. Statistics and animations

An animation is a rapid display of a sequence of images of 1D, 2D or 3D artwork or model positions in order to create an illusion of movement. It is an optical illusion of motion due to the phenomenon of persistence of vision, and can be created and demonstrated in a number of ways.

Animations can assist in a number of areas. For example, we can monitor the steps of K-Means cluster algorithm in multivariate statistics, simulate the approximation of the sampling

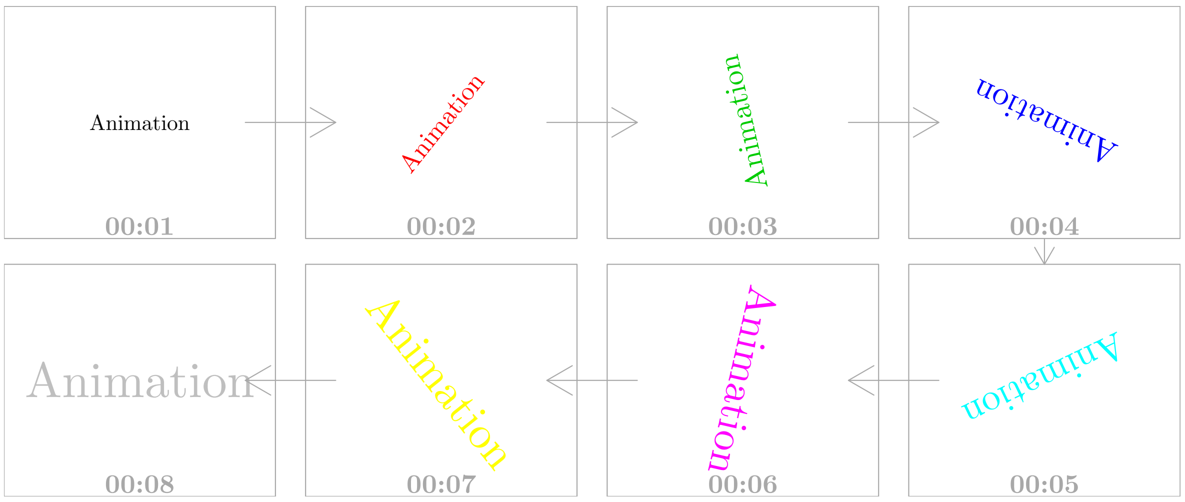


Figure 1: An illustration of a sequence of images: the basic constitution of an animation. See `demo("wordrotation", package = "animation")` for the real animation.

distribution of $\bar{X}_n$ to normal distribution as the sample size n increases, or simply show the dynamic trends of the time-series data as time passes by, etc. The common characteristic of these examples is that they all involve with multiple steps and we may use a *sequence* of images to illustrate the process. The sequence of images forms the base of an animation. Figure 1 is a primitive illustration of how a rotation in an animation is constructed based on a sequence of images.

After a review of some common statistical theories and applications, we think there are generally four types of scenarios as follows in which animations may play a useful and interesting role.

2.1. Iterative algorithms

Iterative algorithms are widely used to derive estimates. Animation can help us monitor the detailed progress, inserting pauses as may be needed for the readers to follow the changes.

Figure 2 is a demonstration of the "gradient descent algorithm" for bivariate functions $z = f(x, y)$. Generally, to minimize a real-valued differentiable function $F(\mathbf{x})$ ($\mathbf{x}$ can be a vector), one can start with a guess $\mathbf{x}_0$ for a local minimum of F, and consider the sequence $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, ... such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \nabla F(\mathbf{x}_n),\ n = 0, 1, 2, \cdots$$

$\nabla F(\cdot)$ is the gradient of the function F. This algorithm has an intuitive interpretation: at each iteration, move the current location of the minimum along the negative gradient by an amount of γ. Specifically, for a bivariate function, we know the gradient at a point is orthogonal to the contour line going through this point. Note R is almost immediately ready for an animation: we can draw the contour plot by `contour()`, calculate the gradient by `deriv()` and use arrows to indicate the process of the iterations, which is the way Figure 2 was constructed. The left animation was set to automatically begin to play when the page is loaded (otherwise we have to click on the animation to play), and we can see the arrows are

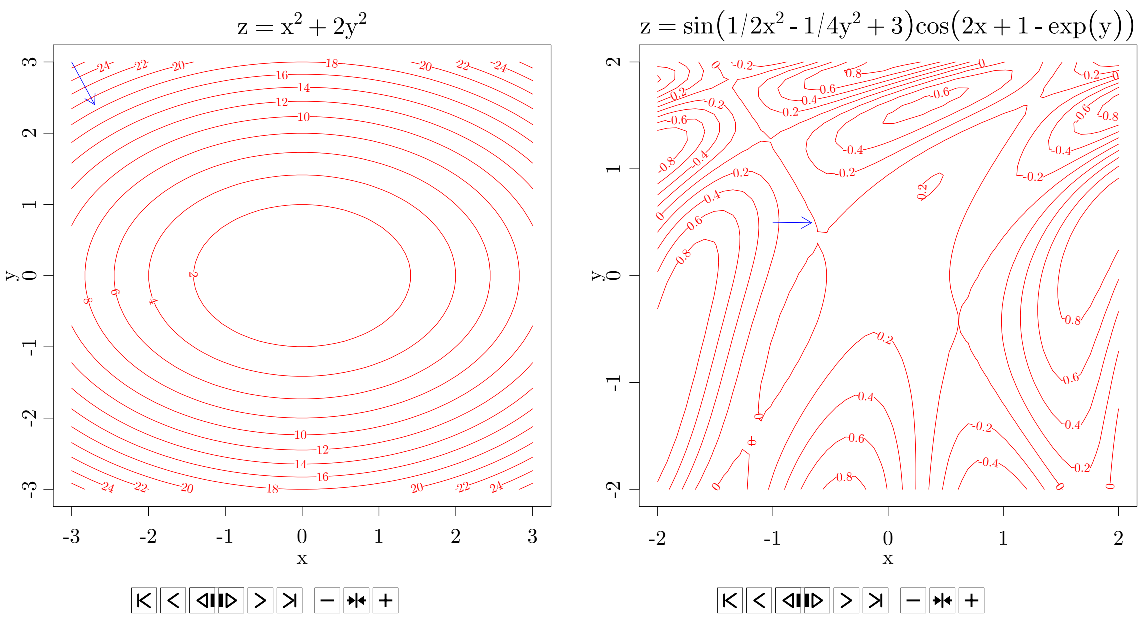


Figure 2: The gradient descent algorithm applied to two bivariate objective functions with different step lengths. The arrows indicate the direction of iterations. See the function `grad.desc()` in the **animation** package.

moving in directions which are orthogonal to the contour lines. Since $z = x^2 + 2y^2$ is a trivial example, we will not be surprised if the arrows finally reach the global minimum $(0, 0)$.

One condition for the gradient descent algorithm to work is that the step length γ has to be small enough. The right animation in Figure 2 can demonstrate the importance of this condition especially when the objective function is complicated, e.g., it has multiple local minima. We need to be careful in choosing the initial guess and the step length in this case. In the animation, the step length is 0.3, which is a large value; although the arrows went to the direction of a local minimum in the beginning, it could not really reach that minimum. At the 12th iteration, the arrow suddenly “jumped” away, because γ was too large for it to approach the minimum. After wiggling for a while near the local minimum, the arrows moved to another area and tried to get close to the local minimum there. This clearly gives us an impression on how the algorithm performs under various conditions; we can further change the location of the initial guess or the value of the step length to improve the process.

There are still a large amount of iterative algorithms that can be animated in this way such as the Newton-Raphson method or the bisection method for finding the roots of an equation $f(x) = 0$, the Gibbs sampling algorithm, and the k-Means clustering algorithm and so on. The iterative nature of these methods makes them suitable candidates for animations, due to the duality between the computationally step-by-step results and the graphically step-by-step animations.

2.2. Random numbers and simulations

We often need to generate random numbers to do simulations, either to validate whether our theories can behave well in a simulated scenario, or to compute estimates based on random numbers. For example, we already know from the CLT that the standardized sample mean

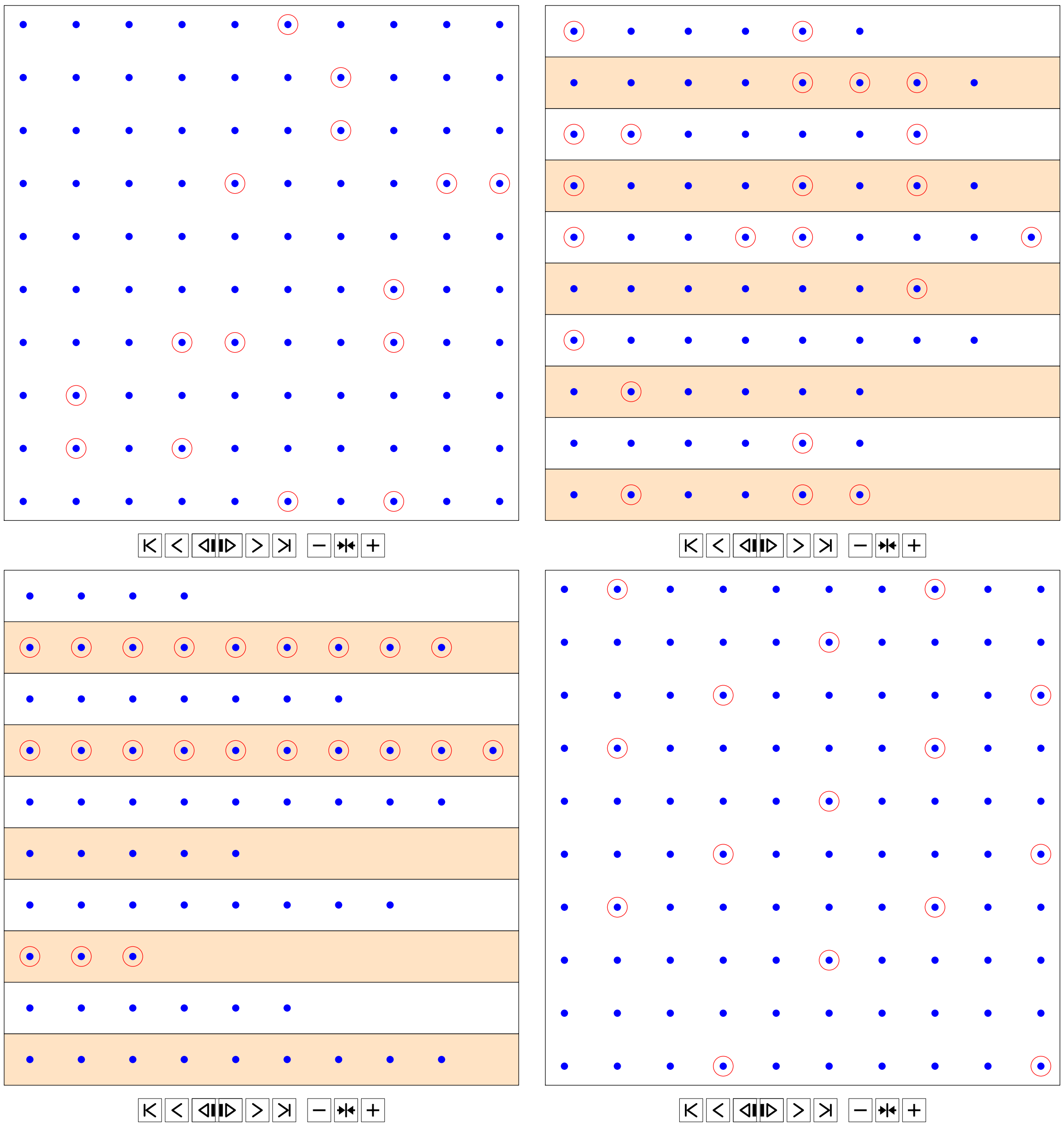


Figure 3: Four basic types of sampling methods: (1) top-left: the simple random sampling without replacement; (2) top-right: stratified sampling – simple random sampling within each stratum with possibly different sample sizes, and the strata are denoted by colored bands; (3) bottom-left: cluster sampling – the population consist of several clusters and we sample the clusters instead of individual points; and (4) bottom-right: systematic sampling – randomly decide the first sample point and find the rest of samples by a fixed increment (e.g., sample every 6th element). The solid dots represent the population, and the circles mark the samples. See functions `sample.simple()`, `sample.strat()`, `sample.cluster()` and `sample.system()` in the **animation** package.

$\sqrt{n}(\bar{X}_n - \mu)/\sigma$ has a limiting standard normal distribution under proper conditions, so we may check how perfectly the distribution of the sample mean approximates to the normal

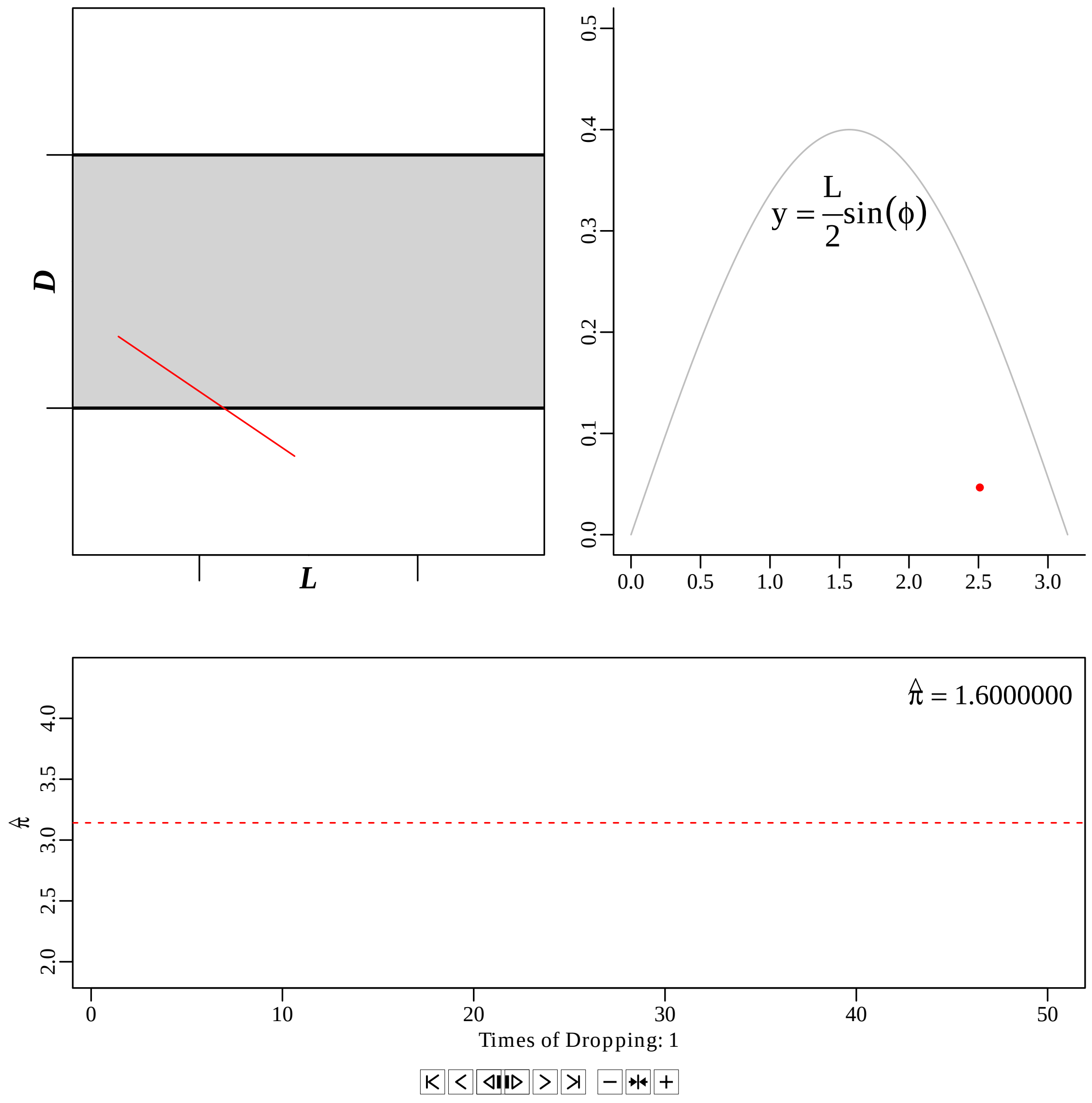


Figure 4: Simulation of Buffon's Needle: (1) top-left: simulation of randomly dropping needles with length L: the crossing angle is ϕ and the distance between parallel lines is D; (2) top-right: corresponding point pairs (ϕ, y), where y is the distance from the center of the needle to the nearest parallel line; the needle crosses the lines if and only if $y \leq \frac{L}{2}\sin(\phi)$ for a given ϕ – this is translated in the plot as "if and only if the point falls below the sine curve", and we know the probability of this event is $\left(\int_0^\pi \frac{L}{2}\sin(\phi)d\phi\right) / \left(\pi\frac{D}{2}\right) = \frac{2L}{\pi D}$ which can be approximated by the proportion of points below the curve; (3) bottom: values of $\hat{\pi}$ calculated from the above simulations. Note we only dropped the needle for 50 times in this example; in general this is not sufficient to obtain a reliable estimate of π. See the function `buffon.needle()` in the **animation** package.

distribution as the sample size n increases to a very large number (theoretically infinity). On the other hand, we can also demonstrate effects of departure from theories, e.g., the behavior of the sample mean when the population variance is infinite, which we will discuss later.

Many numerical experiments, concepts and theorems in probability theory and mathematical statistics can be expressed in the form of animations, including the classical Buffon's Needle (Ramaley 1969), the Law of Large Numbers (LLN), the CLT, the Brownian Motion, coin tosses, confidence intervals and so on. The basic idea is to display the result graphically each time we get a random sample. For instance, in the LLN, we show how the sample means are distributed as we increase the sample size, i.e., for a specific sample size n, we generate m batches of random numbers with each batch of size n, compute the m sample means and plot them; finally we can observe that the sample means will approach to the population mean.

Some elementary animations can be applied to the subject of survey sampling (Cochran 1977) to demonstrate basic sampling methods, such as the simple random sampling, stratified sampling, and systematic sampling, etc (see Figure 3), and one slightly advanced animation in survey sampling is about the ratio estimation – we can show the ratio estimate of the population mean tends to be better than a simple sample average.

The above ideas can be extended to fit into a broader context – to show the sampling variation. This is not well-recognized by the general audience. We often draw conclusions based on a specific dataset and ignore the variation behind the data-generating mechanism. Wild, Pfannkuch, Regan, and Horton (2011) elaborated this point with a rich collection of illustrations. Animations can help the audience build up the "desired habit of mind" advocated there, i.e., whenever I see [this dataset], I remember [the possible variation behind it]. Later we will see Example 1 in Section 4 which demonstrates the "desired habit of reading QQ plots".

Figure 4 is a demonstration of the classical Buffon's Needle problem for approximating the value of π. As we drop more and more needles on the plane, the approximation will be closer to the true value of π in general. This demonstration not only shows the estimate of π each time the needle was dropped in a cumulative manner, but also draws the actual scenario (dropping needles) and the corresponding mathematical interpretation to assist understanding.

2.3. Resampling methods

Instead of sampling from a theoretical distribution, which is usually a common case in Section 2.2, there are also a lot of statistical methods that focus on sampling from the empirical distribution. This is often referred to as "resampling" in the literature. Typical methods include cross-validation, bootstrapping, and permutation methods (Efron and Tibshirani 1994), etc.

Again, we may use animations to investigate the resampling effects if we keep on sampling from the empirical distribution and demonstrate the corresponding output each time we have obtained a new sample X^*. For example, under certain conditions, the standardized sample mean will converge in distribution to a normal random variable (Lahiri 2006), so we may generate several bootstrap replications of the sample mean and examine its convergence to normality as the sample size grows, or failing to converge when the original sample comes from a distribution with infinite variance; this is similar to the LLN case in Section 2.2. In the bootstrap literature, perhaps the moving blocks bootstrap is a better candidate for animations: each time we have selected some blocks of sample points, we can mark these blocks up one by one to emphasize the block structure, then show the output based on these sample blocks. There are also inappropriate uses of resampling, e.g., the jackknife estimate the standard error of the median, which has proved to be inconsistent; or the sampling distribution of extreme quantiles such as the maximum (Miller 1964). These phenomena can

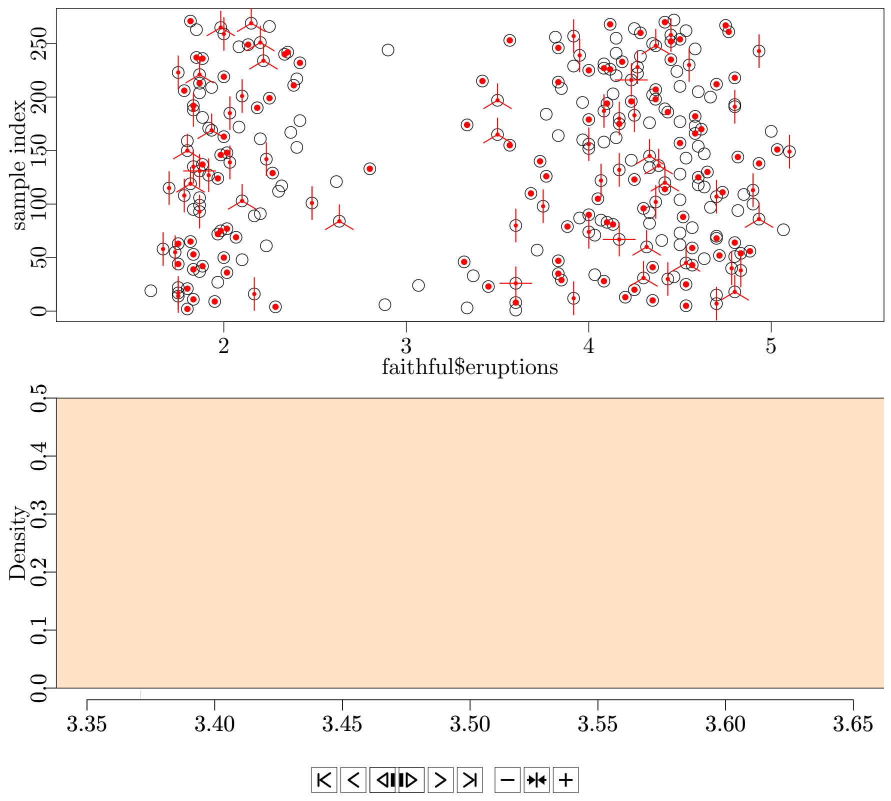


Figure 5: Bootstrapping for the variable `eruptions` of the `faithful` data: the original data is denoted by open circles, while the resampled data is marked by red points with "leaves"; the number of leaves in the sunflower scatter plot means how many times these points are picked out, as bootstrap methods always sample *with* replacement. Note the x-axis in the top plot matches the x-axis in the bottom histogram of the bootstrapped sample mean. See the function `boot.iid()` in the **animation** package.

be demonstrated graphically, as the simplest case in Figure 5 will show.

Figure 5 is an illustration of the resampling nature of bootstrapping, using the sunflower scatter plot. Each animation frame shows one replication of bootstrapping, and the density curve indicates the distribution of the bootstrapping sample mean; the short ticks on the x-axis of the histogram denote the value of the current sample mean. We keep on resampling from the original data and get a number of bootstrapping samples, from which we can make inference about the population mean.

2.4. Dynamic trends

The idea of animations for dynamic trends is perhaps the most natural one. For example, we can be show the values of a time series as time passes by. This is usually not enough in practical applications, so we need to characterize patterns in the data and illustrate their changes via an animation. The definition of the trends in data can be quite flexible, e.g., the relationship between income and expenditure (say, the regression coefficient) in a time span, or the cluster membership of all countries in the world using two socioeconomic indicators as cluster variables. An influential product for this type of animations is Google's "Motion

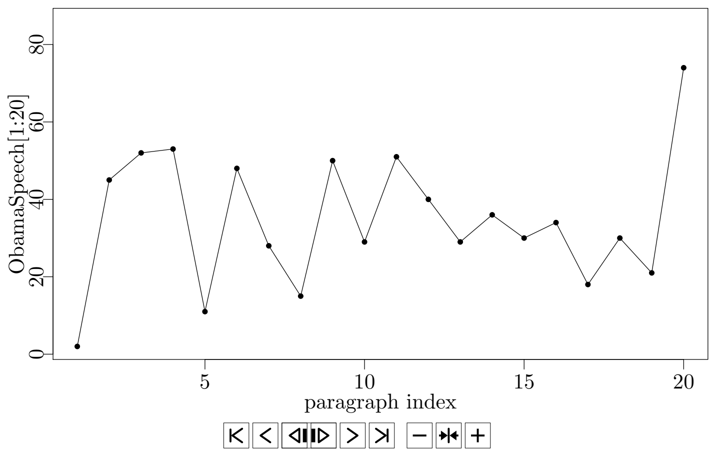


Figure 6: Obama's acceptance speech in Chicago: The word counts from the 1st to the 59th paragraph. The animation can emphasize a pattern along with time – longer paragraphs and shorter paragraphs often appear alternatively. See the dataset `ObamaSpeech` in this package.

Chart" (originally from **Gapminder**), which is often used to show the changes of two or three variables over time using the so-called "bubble chart".

Dynamic trends can be observed more clearly with the help of animations, and sometimes we can draw preliminary conclusions from the animation, e.g., there may be a "breakpoint" in a certain year because we feel a sudden change in the animation. After this kind of exploratory data analysis, we can do formal modeling and hypothesis testing such as the Chow breakpoint test (Chow 1960) to justify these findings.

Figure 6 is a simple example illustrating a pattern in Obama's acceptance speech in Chicago in 2008; we can watch the data in successive chunks.

In an even more general sense, the meaning of "dynamic" does not have to involve with a time variable – we may as well extend this "time" variable to any variables as long as they can be sorted, which essentially means conditioning on a variable. In that case, animations can be created along with the ascending or descending values of this conditioning variable.

3. Design and contents

The R package **animation** (Xie 2013) was initially written based on the above ideas in statistics, and later the package was also developed to be a comprehensive collection of tools to create animations using R. In this section, we first explain the basic schema to create an animation in R, then we introduce the utilities to export animations, and finally we give a brief overview of existing statistical topics covered by this package, as well as the demos in the package. The package is currently available on the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=animation`.

3.1. The basic schema

The approach to create animations in R with the **animation** package is fairly straightforward – just draw plots one by one with a pause between these plots. Below is the basic schema for all the animation functions in the package:

```
  library("animation")
  oopt <- ani.options(interval = 0.2, nmax = 10)
  for (i in 1:ani.options("nmax")) {
    ## draw your plots here, then pause for a while with
    ani.pause()
  }
  ani.options(oopt)
```

The functions `ani.pause()` and `ani.options()` are in the **animation** package; the function `ani.pause()`, based on `Sys.sleep()`, was used to insert pause in the animation when it is played in an *interactive* graphics device (typically a window device), because it is simply a waste of time to pause under a non-interactive off-screen device when we cannot actually see the plots; `ani.options()` can be used to specify many animation options. For example, `ani.options("interval")` sets the time interval to pause between animation frames, and `ani.options("nmax")` is used to set the maximum number of iterations or repetitions. Then we use a loop to draw plots one by one with a pause in each step. In the end we restore the animation options. We can see from below how to use these options to design a simple demonstration for the Brownian motion, which is included in this package as the function `brownian.motion()`:

```
R> brownian.motion

function (n = 10, xlim = c(-20, 20), ylim = c(-20, 20), ...)
{
    x = rnorm(n)
    y = rnorm(n)
    for (i in seq_len(ani.options("nmax"))) {
        dev.hold()
        plot(x, y, xlim = xlim, ylim = ylim, ...)
        text(x, y)
        x = x + rnorm(n)
        y = y + rnorm(n)
        ani.pause()
    }
}
<environment: namespace:animation>
```

The basic idea is to keep on adding an increment in the horizontal and vertical direction respectively, and these increments are independently and identically distributed as $N(0, 1)$. Brownian motion is a model describing the random drifting of particles suspended in a fluid. It is important to set the axes limits (`xlim` and `ylim`) as fixed – not only for this particular animation, but also for other animations in which the range of the coordinates is changing – otherwise we can hardly perceive the motion correctly without a frame of reference. Figure 7 is an output of this function.

Note that although the for-loop is straightforward to create an animation, there are also many cases in which a while-loop becomes useful. Consider, for instance, the gradient descent

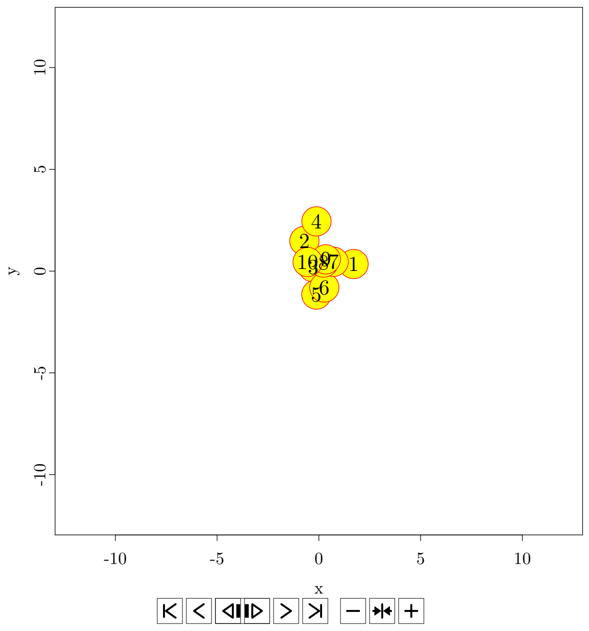


Figure 7: A demonstration of the Brownian motion. Points are moving randomly on the plane; in each step, their locations change by i.i.d. random numbers from $N(0,1)$.

algorithm: there are multiple conditions to break the loop, one of which is the absolute value of changes between two successive iterations (e.g., $|f(x_{k+1}) - f(x_k)| < \delta$). In this case, `ani.options("nmax")` might be smaller than the exact number of animation frames in the output.

3.2. Tools for exporting animations

Many software packages can produce animations, however, as for statistics, the power of statistical computations and graphics plays a more crucial role than animation technologies. The R language is a natural choice as the infrastructure of this **animation** package, since it provides flexible computing routines and graphical facilities, among which those low-level plotting commands are especially useful for generating basic elements of animations (Murrell 2005).

We can watch animations in R's screen devices: the Windows graphics devices (under Windows) or X Window System graphics devices (under Linux) or Mac OS X Quartz devices (under Mac OS X). This looks like a natural choice to play animations, but there are two drawbacks:

1. Each time we want to replay the animation, R has to go through all the computations again and redraw the plots; besides, the animations can only be watched in R.

2. Before R 2.14.0, R's screen devices lack the capability of double buffering, so the animations can be flickering on the screen; the performance is better under Windows than Linux and Mac OS (after R 2.14.0, we can use the new functions `dev.hold()` and `dev.flush()` so the plots will not flicker).

Therefore we need to export animations from R for better performance, portability and to save

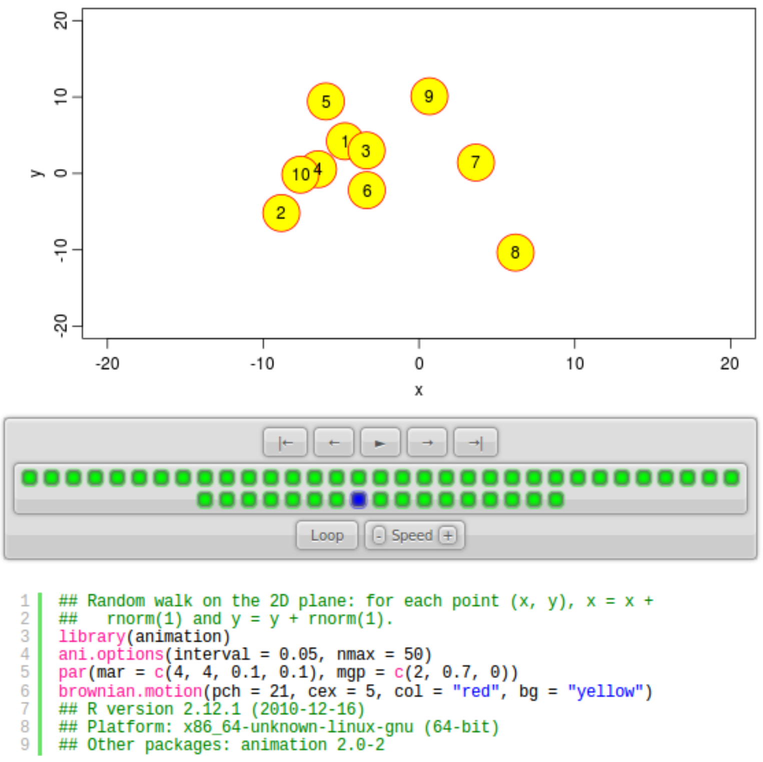
Function	Software dependence	Viewer	Interface
`saveHTML()`	**SciAnimator** (JavaScript)	Any web browser	Very flexible
`saveLatex()`	(pdf)LaTeX	Adobe Reader	flexible
`saveGIF()`	**GraphicsMagick/ImageMagick**	Most image viewers	Relies on the viewer
`saveSWF()`	**SWFTools**	Flash players	Relies on the viewer
`saveVideo()`	**FFmpeg**	Movie players	Relies on the viewer

Table 1: The exporting utilities in the **animation** package.

time. The **animation** package provides five approaches to export animations in other formats: HTML pages, LaTeX/PDF animations, GIF animations, Flash animations, and videos. Table 1 is an overview of these utilities. We have also provided an "animation recorder" to capture all the changes in the graphics device, and this can be helpful when we only use low-level plotting commands to create animations.

HTML pages

We can use the function `saveHTML()` to create an HTML page containing the animations. Figure 8 shows the interface of the animation; basically it is just like a movie player – it has buttons to play/stop the animation and navigate through the images, and we can change the loop mode as well as the speed. The animation player is based on a JavaScript library named **SciAnimator** (Ertz 2011); the bottom part of Figure 8 shows the code to produce the animation



```
## Random walk on the 2D plane: for each point (x, y), x = x +
##   rnorm(1) and y = y + rnorm(1).
library(animation)
ani.options(interval = 0.05, nmax = 50)
par(mar = c(4, 4, 0.1, 0.1), mgp = c(2, 0.7, 0))
brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
## R version 2.12.1 (2010-12-16)
## Platform: x86_64-unknown-linux-gnu (64-bit)
## Other packages: animation 2.0-2
```

Figure 8: The interface of the animation in the HTML page (under the **Firefox** browser).

with some session information for the sake of reproducibility. The R code is automatically added by `saveHTML()` and highlighted by the JavaScript library **SyntaxHighlighter** for better readability. See the help page of `saveHTML()` for download addresses of these libraries; they are built-in with the **animation** package so the users do not need to manually install them.

The usage of `saveHTML()` is:

```
  saveHTML(expr, img.name = "Rplot", global.opts = "", single.opts = "", ...)
```

We only need to provide an R expression `expr` which can produce a sequence of images (it does not have to use a loop); `img.name` specifies the base filename of images to be created, and other arguments can be used to tune more details of the animation (see the help page). This function will record the images using R's off-screen graphics devices (`png()` by default; specified in `ani.options("ani.dev")`), and insert them into the HTML page by JavaScript. Most of the relevant options can be specified by `ani.options()`, e.g., the width and height of images; the options related to the player can be found in the reference of **SciAnimator**. This function allows multiple animations to be embedded into the same HTML file – we only need to specify the option `ani.options("htmlfile")` to be the same each time we call `saveHTML()`.

Below is the code corresponding to Figure 8. Remember the first argument of `saveHTML()` is an R expression to create plots, and the curly brackets `{}` in the code is used to wrap several lines of R code into one expression to be passed to the argument `expr`:

```
R> saveHTML({
+    ani.options(interval = 0.05, nmax = 50)
+    par(mar = c(4, 4, .1, 0.1), mgp = c(2, 0.7, 0))
+    brownian.motion(pch = 21, cex = 5, col = "red", bg = "yellow")
+  }, img.name = "bm_plot", ani.height = 300, ani.width = 550,
+    title = "Demonstration of the Brownian Motion",
+    description = c("Random walk on the 2D plane: for each point",
+      "(x, y), x = x + rnorm(1) and y = y + rnorm(1)."))
```

We have built a website (`http://animation.yihui.name/`) based on the function `saveHTML()` to show the animations online, so that they can be watched by the general audience.

In fact we can easily extend this function for web applications, because what `saveHTML()` essentially does is to write HTML and JavaScript in files. These code can also be written to users' web browsers as the response from a web server. We have succeeded in getting **Rweb** (Banfield 1999) to work with this function, so that the user only needs to submit R code in a web browser to the **Rweb** server, and the server will generate the images and return an animation page to the user. This make it possible for users to create animations without R locally installed. There is a demo file in the package that can show this idea (it depends on the availability of the **Rweb** server):

```
R> system.file("misc", "Rweb", "demo.html", package = "animation")
```

PDF animations via LaTeX

The LaTeX package **animate** by Grahn (2010) is capable of inserting animations into a PDF document, and the animations are also driven by JavaScript. This can be quite helpful, yet is not widely used in our papers or reports. There was an editorial in the JCGS by Levine *et al.* (2010) encouraging authors to take the advantage of animations and movies when publishing electronic papers; the function `saveLatex()` in the **animation** package is one tool to fulfill this mission. This function can be used either as a standalone utility to create PDF animations, or in a `Sweave` document to insert animations dynamically. It will automatically detect if it is called in `Sweave` and output LaTeX code accordingly.

The usage of `saveLatex()` is similar to `saveHTML()`, except that it involves with several LaTeX-related arguments:

```
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, ...)
```

Again, the most important argument is `expr` (an R expression), and the rest of arguments are optional. This will first open a graphical device (specified in `ani.options("ani.dev")`) to record all the plots created by `expr`, then write out the LaTeX code based on the syntax of the **animate** package, e.g., `\animategraphics[controls]{5}{Rplot}{1}{50}`, which can produce an animation from a plot file "`Rplot.pdf`" using the pages from 1 to 50 (or from `Rplot1.png`, ..., `Rplot50.png`, depending the graphics device we use), and the time interval between animation frames is $1/5 = 0.2$ seconds. See the documentation of the **animate** package for details.

The `saveLatex()` function can also be used with `Sweave` directly. There is a demo in this package, `demo("Sweave_animation", package = "animation")`, which is a complete example to show how to use this function with `Sweave` (the `Sweave` source document is `system.file("misc", "Sweave_animation.Rnw", package = "animation")`).

Generally we recommend using PDF graphics because of the high quality, but the file size can easily get too large, since R's `pdf()` device does not support compression of PDF graphics before 2.14.0. To solve this problem, we wrote two wrappers, `qpdf()` and `pdftk()`, for two additional software packages **QPDF** (Berkenbilt 2013) and **PDFtk** (Steward 2013) respectively; if either of them is available in the system, the PDF graphics will be compressed before being processed to animations.

GIF animations, Flash animations and videos

Both JavaScript and LaTeX are common tools, so users can easily create the HTML and PDF animations, and we still have other three approaches: `saveGIF()`, `saveSWF()` and `saveVideo()`, which rely on three less common third-party software packages **GraphicsMagick** (**GraphicsMagick** Group 2013)/**ImageMagick** (**ImageMagick** Studio LLC 2012), **SWFTools** (Kramm *et al.* 2012), and **FFmpeg** (**FFmpeg** Team 2013), respectively. These functions are merely wrappers to these tools, i.e., they will use the function `system()` to call these external tools to convert R images to other animation formats. Specifically, `saveGIF()` calls the external command `gm convert` (**GraphicsMagick**) or `convert` (**ImageMagick**) to create GIF animations; `saveSWF()` calls `png2swf`, `jpeg2swf` or `pdf2swf` in **SWFTools** to convert images to Flash animations; `saveVideo()` calls the command `ffmpeg` in the **FFmpeg** library to convert images to a video (some common formats include Ogg, MP4 and AVI). For these functions to work, they have to know the paths of the tools, which can be set up in the function arguments.

All of the three functions share a similar usage to `saveHTML()` or `saveLatex()`, and the difference is that we may need to specify the location of these tools especially under Windows (using arguments `convert`, `swftools` and `ffmpeg` in three functions respectively):

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

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

saveVideo(expr, video.name = "animation.mp4", img.name = "Rplot",
  ffmpeg = "ffmpeg", other.opts = "", clean = FALSE, ...)
```

A large amount of work was done for Windows to search for the location of these software packages automatically, so that Windows users do not have to be familiar with command lines. The paths of these tools can also be pre-specified in `ani.options()`.

The animation recorder

There is a difficulty in exporting the animations made purely by low-level plotting commands, because the off-screen graphics devices (e.g., the `png()` device) can only capture high-level plotting changes as new image files. For instance, even if we keep on adding points to a plot and it seems to be changing on screen, but when we put the code under an off-screen device, we will only get one image in the end. To solve this problem, we can use the function `recordPlot()` in the **grDevices** package to record all the changes as a list, then replay them using `replayPlot()`. There are two functions `ani.record()` and `ani.replay()` based on these functions in this package, which can act as an "animation recorder". Here is an example that shows the a one-dimensional random walk by adding points on an empty plot; the changes are recorded and replayed:

```
R> oopt <- ani.options(nmax = 50, interval = 0.1)
R> x <- cumsum(rnorm(n = ani.options("nmax")))
R> ani.record(reset = TRUE)
R> par(bg = "white", mar = c(4, 4, 0.1, 0.1))
R> plot(x, type = "n")
R> for(i in 1:length(x)) {
+    points(i, x[i])
+    ani.record()
+  }
R> ani.replay()
R> ani.options(oopt)
```

We can also replay inside a `saveXXX()` function introduced before to export the animation, e.g., `saveHTML(ani.replay())`.

Other R packages

There are a few other R packages to export animations; for example, the **SVGAnnotation** package by Nolan and Temple Lang (2012) can create a simple SVG (scalable vector graphics) animation using the function `animate()` to animate scatter plots like the Google Motion Chart, and the **R2SWF** package by Qiu and Xie (2011) can create a Flash animation by converting bitmap images to a SWF (ShockWave Flash) file based on the **swfmill** library (Turing *et al.* 2011). These packages can be run on their own, i.e., they are not simply command line wrappers for third-party software packages. There is yet another package named **swfDevice** (Bracken 2010) to create Flash animations, which has not been released to CRAN but looks useful as well (currently on R-Forge).

3.3. Topics in statistics

This package contains a large variety of functions for animations in statistics, covering topics in several areas such as probability theory, mathematical statistics, multivariate statistics, nonparametric statistics, sampling survey, linear models, time series, computational statistics, data mining and machine learning, etc. These functions might be of help both in teaching statistics and in data analysis.

As of version 2.0-7, there are nearly 30 functions related to statistics in this package:

`bisection.method()` The bisection method for root-finding on an interval.

`boot.iid()` Bootstrapping for i.i.d. data (demonstrate the idea of sampling with replacement).

`boot.lowess()` Fit LOWESS curves based on resampled data.

`brownian.motion()`, `BM.circle()`, `g.brownian.motion()` Different demonstrations of the Brownian motion on the 2D plane.

`buffon.needle()` Simulation of the Buffon's needle problem to approximate π.

`clt.ani()` Demonstration of the central limit theorem (also shows the goodness-of-fit to the normal distribution as the sample size increases).

`conf.int()` Demonstration of the concept of confidence intervals.

`cv.ani()` Demonstration of the concept of cross-validation.

`flip.coin()` Simulation of flipping coins.

`grad.desc()` The gradient descent algorithm.

`kmeans.ani()` The k-means cluster algorithm (show the changes of cluster labels and the move of cluster centers).

`knn.ani()` The k-nearest neighbors classification algorithm.

`least.squares()` Demonstration of least squares estimation (show how the residual sum of squares changes when the coefficients change).

`lln.ani()` The law of large numbers.

`MC.hitormiss(),MC.samplemean()` Two approaches of the Monte Carlo integration.

`mwar.ani()` The moving window auto-regression.

`newton.method()` The Newton-Raphson method for root-finding.

`quincunx()` Demonstration of the Quincunx (i.e., the Bean Machine) to show the normal distribution coming from the falling beans.

`sample.cluster()` The cluster sampling.

`sample.simple()` The simple random sampling without replacement.

`sample.strat()` The stratified sampling.

`sample.system()` The systematic sampling.

`sample.ratio()` The demonstration of advantages of the ratio estimation in survey sampling.

`sim.qqnorm()` The simulation of QQ plots to show how they look like if the data really comes from the normal distribution.

3.4. Demos

Demos are an important part of this package. They can be divided into two groups: some are purely for entertaining purposes, and the others can demonstrate additional capabilities and applications of this package. Here we introduce a selective subset of these demos.

For the `saveXXX()` functions in Section 3.2, we can extend them in flexible ways. For example, we do not have to restrict ourselves by the common R graphics devices; we can use arbitrary devices, as long as we follow the "naming template" for image files. This is explained in detail in the options `use.dev` and `img.fmt` in the help page of `ani.options()`. The `demo("rgl_animation")` shows how to capture the 3D plots created by the **rgl** package (Adler and Murdoch 2012); see Figure 9 for the animation, which was embedded in this paper using the **knitr** package. This demo is actually a quick summary of the 1986 ASA Data Expo – looking for patterns in a synthetic dataset. The original plot seems to be nothing but a point cloud, but there is actually a word "EUREKA" hidden in it; we can clearly see them by zooming into the points. A similar demo is `demo("ggobi_animation")` – it shows how to record plots in **GGobi** and create an HTML animation page accordingly.

Another demo shows that we can even download images from the Internet, and wrap them into an animation with `saveHTML()`; see `demo("flowers")`. This again indicates the flexibility of the **animation** package. To learn how these two demos work, we can take a look at the source code:

```
R> file.show(system.file("demo", "rgl_animation.R", package = "animation"))
R> file.show(system.file("demo", "flowers.R", package = "animation"))
```

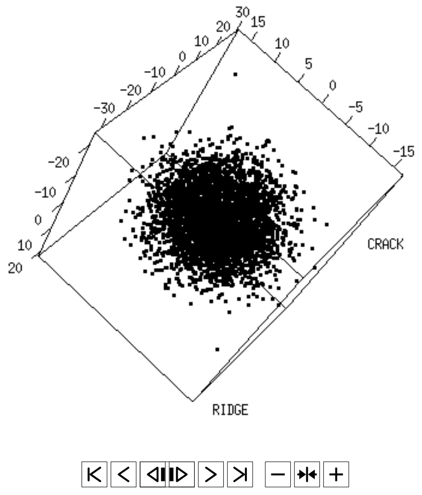


Figure 9: A 3D animation demo (created by the **rgl** package): the classical `pollen` data. The hidden letters will emerge as we gradually zoom into the point cloud.

Some entertaining demos include `demo("game_of_life")` (three types of the famous “Game of Life”), `demo("hanoi")` (the recursive algorithm of the classical game “Tower of Hanoi”), and `demo("fire")` (a simulation of the fire flames) and so on. Finally, we can also demonstrate interesting datasets in animations, e.g., `demo("CLEvsLAL")` is a “playback” of an NBA game between Cavaliers and Lakers on Dec 25, 2009; at each time point, we can clearly see the location of the current player, and whether he made the goal or not.

4. Examples

In this section we give three examples illustrating the application of animations to both statistical theories and data analysis. The first example shows we can construct simple but convincing animations to explain how to interpret QQ plots reasonably; the second example quickly shows how the sample mean behaves when the conditions for CLT are not satisfied; the third one gives the process of cross-validation.

Example 1: QQ plots for different sample sizes

QQ plots are often used to check the normality of residuals in linear regressions. In practice, we may over-interpret these plots – departure of the points from a straight line does not necessarily mean a severe contradiction with normality. Figure 10 shows several QQ plots for different batches of random numbers which are really from the standard normal distribution, however, we can clearly see that few of them really fall on the diagonal (i.e., the line $y = x$) in the left plot, and there are almost always a few points in the tails which are far away from the diagonal in the right plot. The animation gives the readers an impression of “deviation” because the diagnal is fixed in all frames while the points are “wiggling” around it, and the animation can also hold more plots than a static $m \times n$ display of individual plots.

These two animations correspond to the sample size 20 and 100 respectively. When the sample size is small, QQ plots might give us a fairly high type I error rate (reject normality even if

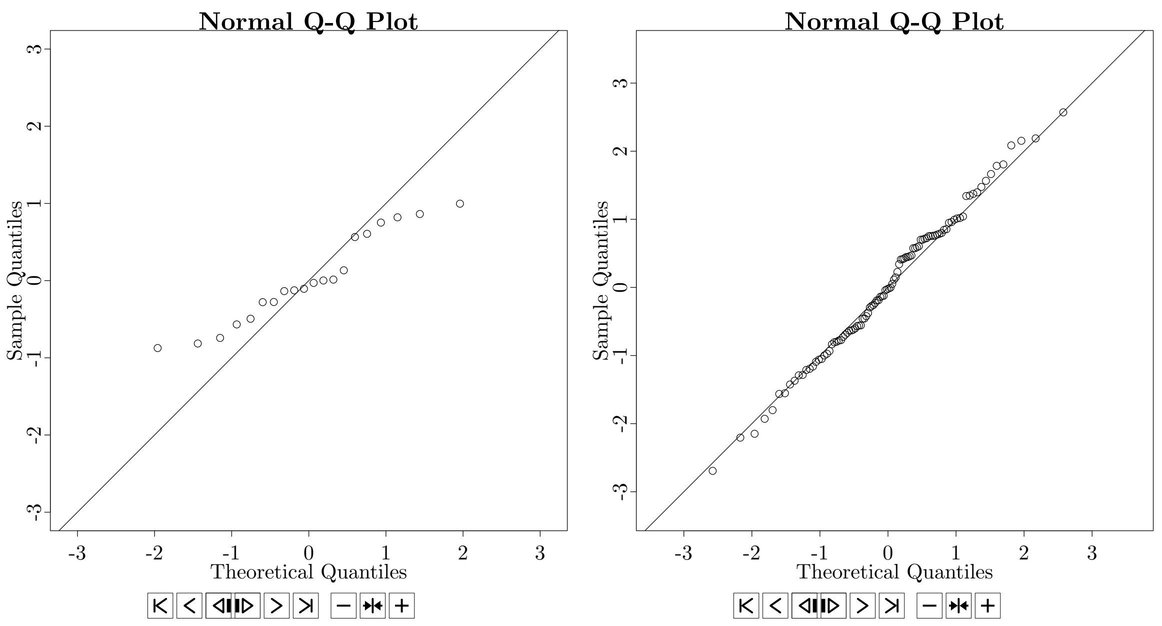


Figure 10: The QQ plot for random numbers from $N(0,1)$ with a sample size 20 (left) and 100 (right) respectively. The points may not strictly stick to the diagonal even they are really from the normal distribution.

the sample really comes from the normal distribution); on the other hand, we should not pay too much attention to "outliers" in the tails in QQ plots when the sample size is large.

To see how quickly and easily we can write such an animation function, we may look at the source code of the function `sim.qqnorm()`, which created the animations in Figure 10. It is basically a loop creating QQ plots with `qqnorm()` based different batches of random numbers:

```
R> sim.qqnorm

function (n = 20, last.plot = NULL, ...)
{
    for (i in 1:ani.options("nmax")) {
        dev.hold()
        qqnorm(rnorm(n), ...)
        eval(last.plot)
        ani.pause()
    }
}
<environment: namespace:animation>
```

Example 2: Limiting distribution of the sample mean

The function `clt.ani()` was designed to demonstrate the central limit theorem, i.e., to illustrate the limiting distribution of $\bar{X}_n$ as n increases. In the animation, the density of $\bar{X}_n$ is estimated from a number of simulated mean values based on a given sample size n.

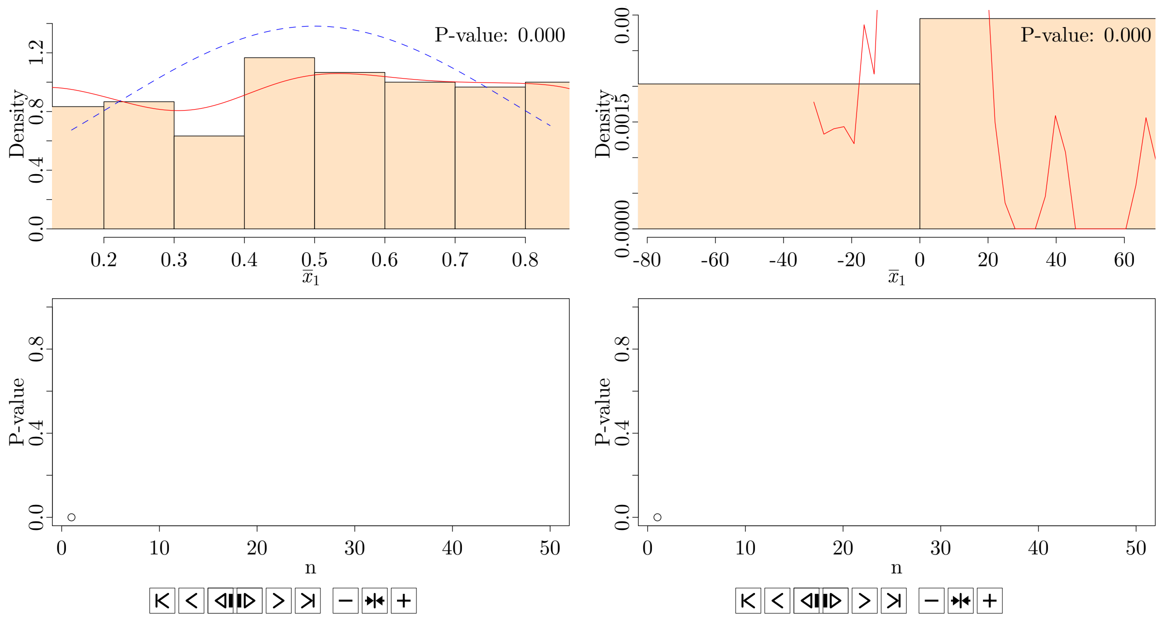


Figure 11: The normality of the sample mean $\bar{X}_n$ as the sample size n increases from 1 to 50 with $X \sim Unif(0,1)$ (left) and $Cauchy(0,1)$ (right); the dashed line is the theoretical limiting density curve. The conditions for CLT are satisfied in the left plot, and we see the p values will no longer be systematically small as the sample size grows; but the right plot can never give us a bell-shaped density, because the variance of the Cauchy distribution does not exist.

Figure 11 is a comparison for the distributions of the sample means when $n = 50$. The population distributions in the left and right animations are the uniform and Cauchy distributions respectively. We know the latter one does not satisfy the condition of finite variance in CLT, hence it should not give us any bell-shaped density curves no matter how large the sample size is, which is true according to the right animation in Figure 11. The upper plot is the histogram overlaid with a density curve to show the distribution of $\bar{X}_n$ (the dashed line denotes the theoretical limiting distribution), and the lower plot shows the corresponding p values from the Shapiro-Wilk test of normality – if normality holds, the p values should be uniformly distributed, so if p values are systematically small, normality can be questionable. The demonstration of CLT is an extremely old idea, and there exists a large number of demos, but very few of them emphasized the goodness-of-fit to normality – usually they merely show how the density curve of the sample mean can become bell-shaped, but "being bell-shaped" alone could be far from normality. Therefore we especially added the theoretical limiting density curve as well as the p values in order that readers have an idea about the departure from the theoretical normality. For example, we know for $Unif(0,1)$, $\bar{X}_n \xrightarrow{d} N(\frac{1}{2}, \frac{1}{12n})$, so we can draw the limiting density curve to compare with the empirical density curve of $\bar{X}_n$. Note we used p values only as one measure of goodness-of-fit, and large p values do not necessarily indicate normality (we just cannot reject it).

Example 3: Classification of gene expression data

It is very common that the gene expression data is high-dimensional with variables far more than observations. We may hope to use as few variables as possible to build predictive models, since too many variables can bring the problem of overfitting. The main ideas in this example are borrowed from Maindonald and Braun (2007, pp. 400), but here we only illustrate the process of cross-validation and the corresponding results.

Suppose we want to find out the optimum number (say, less than $g_{\max} = 10$) of features using the linear discriminant analysis (LDA) with a 5-fold cross-validation. The procedure is:

- Split the whole data randomly into 5 folds:
  - For the number of features $g = 1, 2, \cdots, 10$, choose g features that have the largest discriminatory power individually (measured by the F statistic in ANOVA of one feature against the cancer types):
    - For the fold i ($i = 1, 2, \cdots, 5$): Train a LDA model without the i-th fold data, and predict with the i-th fold for a proportion of correct predictions $\bar{p}_{gi}$;
  - Average the 5 proportions to get an average rate of correct prediction $\bar{p}_{g\cdot}$;
- Determine the optimum number of features as $\arg\max_{g} \bar{p}_{g\cdot}$.

This procedure was implemented in the function `cv.nfeaturesLDA()` in the **animation** package, and we use the datasets `Golub` and `golubInfo` from the **hddplot** package (Maindonald 2012). The goal is to correctly classify three cancer types (`allB`, `allT` and `aml`) using as less variables as possible from all the 7129 features; the sample size is 72, and we want to restrict the maximal number of features to be 10.

In Figure 12, points with sizes proportional to the rates of correct predictions are moving from bottom to top and left to right, which demonstrates the process of k-fold cross-validation (1 to k on the y-axis) and the increasing number of features (1 to $g_{\max}$ on the x-axis), respectively. For a fixed number g, we denote the rate of correct predictions p_{gi} for $i = 1, 2, \cdots, k$ one by one in the vertical direction, and then move to the next number $g + 1$. The average rates are computed and denoted at the bottom of the graph; points that marked by "?" denote unknown p_{gi}, i.e., they have not been computed yet. In the end, we only have to find out the number g corresponding to the biggest point in the bottom. The results are as follows:

```
R> set.seed(130)
R> library("hddplot")
R> data("Golub")
R> data("golubInfo")
R> ani.options(nmax = 10)
R> par(mar = c(3, 3, 0.2, 0.7), mgp = c(1.5, 0.5, 0))
R> res <- cv.nfeaturesLDA(t(Golub), cl = golubInfo$cancer, k = 5,
+    cex.rg = c(0, 3), pch = 19)
R> res$accuracy
```

```
          1     2     3     4     5     6     7     8     9    10
Fold1 0.500 0.429 0.500 0.857 0.929 0.929 0.929 0.857 0.857 0.857
Fold2 0.467 0.533 0.600 0.800 0.733 0.800 0.800 0.867 0.733 0.733
Fold3 0.714 0.714 0.786 0.714 0.714 0.714 0.714 0.571 0.786 0.643
Fold4 0.357 0.429 0.500 0.571 0.571 0.571 0.571 0.571 0.571 0.571
Fold5 0.533 0.600 0.600 0.667 0.467 0.533 0.667 0.533 0.867 0.867
```

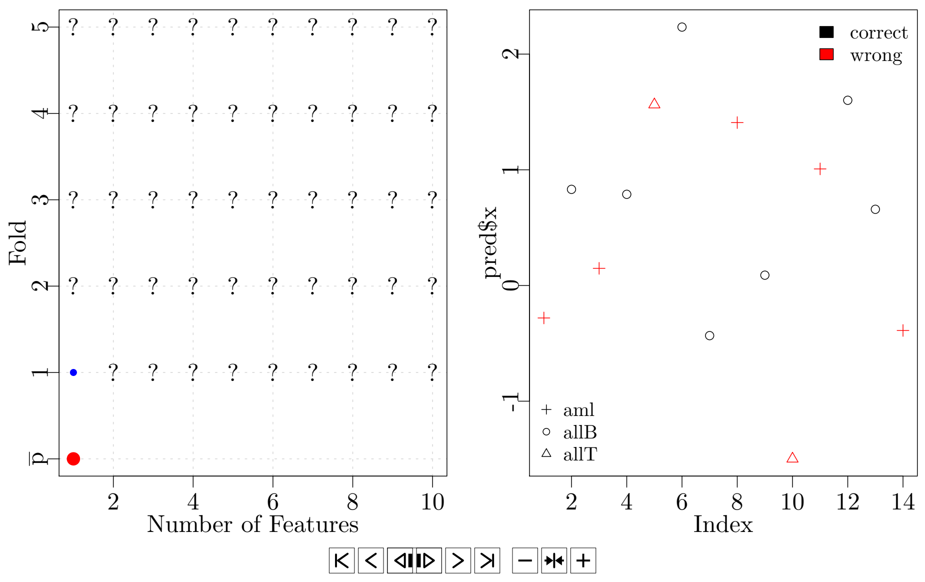


Figure 12: An illustration of the 5-fold cross-validation for finding the optimum number of gene features based on LDA. The left plot shows the correct rate (denoted by the size of dots); the right plot shows the test data on the first 2 discriminant axes (only 1 axis when $g = 1$ so the first 5 frames are different with latter frames); correct predictions and misclassified cases are marked by different colors.

```
R> res$optimum
```

```
[1] 9
```

The maximum prediction accuracy is achieved for 9 features (`res$optimum`); the matrix `res$accuracy` shows the prediction accuracy for $g = 1, 2, \cdots, 10$ in the column and fold 1 to 5 in the 5-fold cross-validation. In fact, this example is more a "slide show" than a real animation, but it can be a helpful tool to explain the procedure.

5. Conclusions

The goal of the S language was "to turn ideas into software, quickly and faithfully" (Chambers 1998); for the **animation** package, the goal is "to turn ideas into animations (quickly and faithfully)". This package won the John M. Chambers Statistical Software Award (`http://stat-computing.org/awards/jmc/`) in 2009.

In Section 2, we have reviewed the close relationship between animations and statistics in certain aspects. We can use animations to illustrate every single step of iterative algorithms, show all the results from simulations or resampling methods in detail, and characterize dynamic trends in the data.

Section 3 introduced the design and contents of this package. The programming schema for the animation functions is quite simple but effective, and animations made from this schema can help us better understand statistical methods and concepts, and gain insights into data. This package contains nearly 30 demonstrations for statistical topics, as well as more than

20 demos for other topics such as computer games or novel applications of some functions in this package.

The examples in Section 4 have shown the advantage of animations in both demonstrating statistical theories and results of data analysis: the procedures illustrated step by step are fairly clear, and the results can be observed on the fly. In traditional static printed reports and papers, these dynamic information cannot be conveyed conveniently.

We have also introduced the reproducibility of animations in this paper, and the **animation** package has functions to incorporate with reproducibility. For example, the `saveHTML()` function can write the R code as well as the R session information into an HTML page, so that users can copy and paste the code to reproduce the animations; `saveLatex()` can be used in the `Sweave` environment to insert animations into LaTeX documents, which usually will be compiled to PDF documents, so that we can watch animations in the PDF documents. The source file of this paper can be found under `system.file("articles", "jss725.Rnw", package = "animation")` and this paper can be reproduced by:

```
R> install.packages(c("hddplot", "rgl", "tikzDevice", "animation", "knitr"))
R> library("knitr")
R> knit("jss725.Rnw")
```

Xie and Cheng (2008) is an introductory article for this package in *R News*, and since then, there have been a large amount of improvements in terms of usability, consistency, documentation, new topics in statistics, and additional utilities. Currently we have included comprehensive utilities for exporting animations in R, and in future the main direction is to contribute more functions related to statistics. Depending on the demand of users, we may also consider developing a GUI for this package using, for example, the **gWidgets** package.

Acknowledgments

I'm grateful to John Maindonald for his generous help on the manuscript and suggestions to this package ever since early 2008. I'd also like to thank Di Cook and the anonymous reviewers for the helpful comments. Besides, I appreciate all the suggestions and feedback from the users, without whom this package would not have achieved its current quality and flexibility (see `news(package = "animation")`). I thank the authors of open-source software packages **ImageMagick**, **GraphicsMagick**, **SWFTools**, **FFmpeg**, the LaTeX package **animate**, the JavaScript libraries **SciAnimator** and **SyntaxHighlighter**, **QPDF** and **PDFtk**. Finally, the 2009 John M. Chambers Statistical Software Award was a substantial encouragement to the development of this package, and I thank Dr. Chambers as well as the award committee. This work has been partly supported by the National Science Foundation grant DMS0706949.

References

Adler D, Murdoch D (2012). ***rgl**: 3D Visualization Device System (OpenGL).* R package version 0.92.880, URL `http://CRAN.R-project.org/package=rgl`.

Andrews F (2010). ***playwith**: A GUI for Interactive Plots Using **GTK+**.* R package version 0.9-53, URL `http://CRAN.R-project.org/package=playwith`.

Banfield J (1999). "**Rweb**: Web-Based Statistical Analysis." *Journal of Statistical Software*, **4**(1), 1–15. URL `http://www.jstatsoft.org/v04/i01/`.

Berkenbilt J (2013). ***QPDF**: A Content-Preserving PDF Transformation System.* URL `http://qpdf.sourceforge.net/`.

Bowman A, Crawford E, Alexander G, Bowman RW (2007). "**rpanel**: Simple Interactive Controls for R Functions Using the **tcltk** Package." *Journal of Statistical Software*, **17**(9), 1–18. URL `http://www.jstatsoft.org/v17/i09/`.

Bracken C (2010). ***swfDevice**: R Graphics Device for SWF (Flash) Output.* R package version 0.1/r41, URL `http://R-Forge.R-project.org/projects/swfdevice/`.

Chambers JM (1998). *Programming with Data: A Guide to the S Language.* Springer-Verlag, New York.

Chow GC (1960). "Tests of Equality Between Sets of Coefficients in Two Linear Regressions." *Econometrica*, pp. 591–605.

Cochran WG (1977). *Sampling Techniques.* 3rd edition. John Wiley & Sons.

Cook D, Swayne DF (2007). *Interactive and Dynamic Graphics for Data Analysis with R and **GGobi**.* Springer-Verlag, New York.

Dalgaard P (2001). "A Primer on the R-Tcl/Tk Package." *R News*, **1**(3), 27–31. URL `http://CRAN.R-project.org/doc/Rnews/`.

Efron B, Tibshirani R (1994). *An Introduction to the Bootstrap.* Chapman & Hall/CRC, Boca Raton.

Ertz B (2011). ***SciAnimator**: Scientific Image Animator Plugin for **jQuery**.* URL `http://brentertz.github.com/scianimator/`.

FFmpeg Team (2013). ***FFmpeg**.* URL `http://FFmpeg.org/`.

Fisher D (2010). "Animation for Visualization: Opportunities and Drawbacks." In J Steele, N Iliinsky (eds.), *Beautiful Visualization: Looking at Data Through the Eyes of Experts*, pp. 329–351. O'Reilly.

Grahn A (2010). *The **animate** Package.* URL `http://tug.ctan.org/pkg/animate`.

GraphicsMagick Group (2013). ***GraphicsMagick** Image Processing System.* URL `http://www.GraphicsMagick.org/`.

ImageMagick Studio LLC (2012). ***ImageMagick**: Convert, Edit, and Compose Images.* URL `http://www.ImageMagick.org/`.

Kramm M, *et al.* (2012). ***SWFTools**: SWF Manipulation and Generation Utilities.* URL `http://www.SWFTools.org/`.

Lahiri SN (2006). "Bootstrap Methods: A Review." In J Fan, H Koul (eds.), *Frontiers in Statistics*, pp. 231–256. Imperial College Press.

Lamport L (1994). *LaTeX: A Document Preparation System.* 2nd edition. Addison-Wesley, Reading.

Lawrence M, Temple Lang D (2010). "**RGtk2**: A Graphical User Interface Toolkit for R." *Journal of Statistical Software*, **37**(8), 1–52. URL `http://www.jstatsoft.org/v37/i08/`.

Leisch F (2002). "Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *COMPSTAT 2002 – Proceedings in Computational Statistics*, pp. 575–580. Physica-Verlag, Heidelberg.

Levine RA, Tierney L, Wickham H, Sampson E, Cook D, van Dyk DA (2010). "Editorial: Publishing Animations, 3D Visualizations, and Movies in JCGS." *Journal of Computational and Graphical Statistics*, **19**(1), 1–2.

Maindonald J (2012). ***hddplot**: Use Known Groups in High-Dimensional Data to Derive Scores for Plots.* R package version 0.55, URL `http://CRAN.R-project.org/package=hddplot`.

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

Miller RG (1964). "A Trustworthy Jackknife." *The Annals of Mathematical Statistics*, **35**(4), 1594–1605.

Murrell P (2005). *R Graphics.* Chapman & Hall/CRC, Boca Raton.

Nolan D, Temple Lang D (2012). "Interactive and Animated Scalable Vector Graphics and R Data Displays." *Journal of Statistical Software*, **46**(1), 1–88. URL `http://www.jstatsoft.org/v46/i01/`.

Qiu Y, Xie Y (2011). ***R2SWF**: Convert R Graphics to Flash Animations.* R package version 0.3-0, URL `http://CRAN.R-project.org/package=R2SWF`.

Ramaley JF (1969). "Buffon's Noodle Problem." *The American Mathematical Monthly*, **76**(8), 916–918.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org/`.

Rosling H, Johansson C (2009). "**Gapminder**: Liberating the x-Axis from the Burden of Time." *Statistical Computing and Statistical Graphics Newsletter*, **20**(1), 4–7.

Snow G (2012). ***TeachingDemos**: Demonstrations for Teaching and Learning.* R package version 2.8, URL `http://CRAN.R-project.org/package=TeachingDemos`.

Steward S (2013). ***PDFtk**: The PDF Toolkit.* URL `http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/`.

Temple Lang D, Swayne D, Wickham H, Lawrence M (2012). ***rggobi**: Interface between R and **GGobi**.* R package version 2.1.18, URL `http://CRAN.R-project.org/package=rggobi`.

Turing D, *et al.* (2011). ***swfmill***. URL http://swfmill.org/.

Urbanek S, Wichtrey T (2011). ***iplots**: iPlots – Interactive Graphics for R*. R package version 1.1-4, URL http://CRAN.R-project.org/package=iplots.

Velleman PF, Moore DS (1996). "Multimedia for Teaching Statistics: Promises and Pitfalls." *The American Statistician*, pp. 217–225.

Verzani J (2007). "An Introduction to **gWidgets**." *R News*, **7**(3), 26–33. URL http://CRAN.R-project.org/doc/Rnews/.

Verzani J (2012). ***gWidgets**: **gWidgets** API for Building Toolkit-Independent, Interactive GUIs*. R package version 0.0-50, URL http://CRAN.R-project.org/package=gWidgets.

Wender KF, Muehlboeck JS (2003). "Animated Diagrams in Teaching Statistics." *Behavior Research Methods*, **35**(2), 255–258.

Wild CJ, Pfannkuch M, Regan M, Horton N (2011). "Towards More Accessible Conceptions of Statistical Inference." *Journal of the Royal Statistical Society A*, **174**(2), 247–295.

Xie Y (2012). ***knitr**: A General-Purpose Package for Dynamic Report Generation in R*. R package version 0.6.3, URL http://CRAN.R-project.org/package=knitr.

Xie Y (2013). ***animation**: A Gallery of Animations in Statistics and Utilities to Create Animations*. R package version 2.2, URL http://CRAN.R-project.org/package=animation.

Xie Y, Cheng X (2008). "**animation**: A Package for Statistical Animations." *R News*, **8**(2), 23–27. URL http://CRAN.R-project.org/doc/Rnews/.

Affiliation:

Yihui Xie
Department of Statistics & Statistical Laboratory
Iowa State University
102 Snedecor Hall
Ames, IA 50011, United States of America
E-mail: xie@yihui.name
URL: http://yihui.name/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 53, Issue 1 *Submitted:* 2011-02-12
April 2013 *Accepted:* 2012-07-02