animation: A Package for Statistical Animations

by Yihui Xie and Xiaoyue Cheng

The mastery of statistical ideas has always been a challenge for many who study statistics. The power of modern statistical software, taking advantage of advances in theory and in computer systems, both adds to those challenges and offers new opportunities for creating tools that will assist learners. The **animation** package (Xie, 2008) uses graphical and other animations to communicate the results of statistical simulations, giving meaning to abstract statistical theory. From our own experience, and that of our classmates, we are convinced that such experimentation can assist the learning of statistics. Often, practically minded data analysts try to bypass much of the theory, moving quickly into practical data analysis. For such individuals, use of animation to help understand simulations may make their use of statistical methods less of a "black box".

Introduction

Animation is the rapid display of a sequence of 2D or 3D images to create an illusion of movement. Persistence of vision creates the illusion of motion. Figure 1 shows a succession of frames that might be used for a simple animation – here an animation that does not serve any obvious statistical purpose.

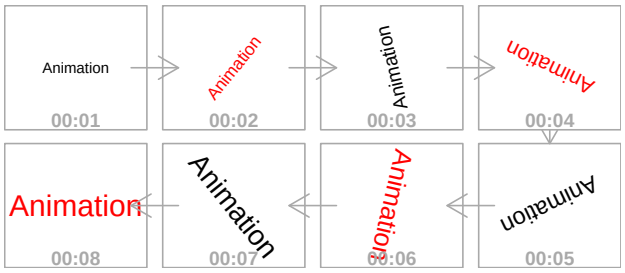

Figure 1: This sequence of frames, displayed in quick succession, creates an illusion of movement.

An animation consists of multiple image frames, which can be designed to correspond to the successive steps of an algorithm or of a data analysis. Thus far, we have sought to:

- build a gallery of statistical animations, including some that have a large element of novelty;
- provide convenient functions to generate animations in a variety of formats including HTML/JS (JavaScript), GIF/MPEG and SWF (Flash);
- make the package extensible, allowing users to create their own animations.

Design and features

It is convenient to create animations using the R environment. The code below illustrates how the functions in the **animation** package produce animations. In this case, the output is similar to the animation on the web page obtained by going to `http://animation.yihui.name/animation:start#method`, as indicated in the first line of the following code:

```
# animation:start#method
clr = rainbow(360)
for (i in 1:360) {
    plot(1, ann = F, type = "n", axes = F)
    text(1, 1, "Animation", srt = i, col =
        clr[i], cex = 7 * i/360)
    Sys.sleep(0.01)
}
```

High-level and low-level plotting commands, used along with various graphical devices described in Murrell (2005), make it straightforward to create and display animation frames. Within an R session, any available graphics device (for 'Windows', 'Linux', 'MacOS X', etc) can be used for display. Any R graphics system can be used (**graphics**, **grid**, **lattice**, **ggplot**, etc).

Image formats in which animations can be saved (and shared) include, depending on the computer platform, `png()`, `jpeg()` and `pdf()`. For display separately from an R session, possible formats include (1) HTML and JS, (2) GIF/MPEG, or (3) Flash. All that is needed is (1) a browser with JavaScript enabled, (2) an image viewer (for GIF) / a media player (for MPEG), or (3) a Flash viewer for the relevant animation format.

The basic schema for all animation functions in the package is:

```
ani.fun <- function(args.for.stat.method,
    args.for.graphics, ...) {
    {stat.calculation.for.preparation.here}
    i = 1
    while (i <= ani.options("nmax") &
        other.conditions.for.stat.method) {
        {stat.calculation.for.animation}
        {plot.results.in.ith.step}
        # pause for a while in this step
        Sys.sleep(ani.options("interval"))
        i = i + 1
    }
```

```
    # (i - 1) frames produced in the loop
    ani.options("nmax") = i - 1
    {return.something}
}
```

As the above code illustrates, an animation consists of successive plots with pauses inserted as appropriate. The function `ani.options()`, modeled on the function `options()`, is available to set or query animation parameters. For example, `nmax` sets the maximum number of steps in a loop, while `interval` sets the duration for the pause in each step. The following demonstrates the use of `ani.options()` in creating an animation of Brownian Motion. (As with all code shown in this paper, a comment on the first line of code has the name of the web page, in the root directory `http://animation.yihui.name`, where the animation can be found.)

```
# prob:brownian_motion
library(animation)
# 100 frames of brownian motion
ani.options(nmax = 100)
brownian.motion(pch = 21, cex = 5,
    col = "red", bg = "yellow")

# faster: interval = 0.02
ani.options(interval = 0.02)
brownian.motion(pch = 21, cex = 5,
    col = "red", bg = "yellow")
```

Additional software is not required for HTML/JS, as the pages are generated by writing pure text (HTML/JS) files. The functions `ani.start()` and `ani.stop()` do all that is needed. Creation of animations in GIF/MPEG or SWF formats requires third-party utilities such as ImageMagick[1] or SWF Tools[2]. We can use `saveMovie()` and `saveSWF()` to create GIF/MPEG and Flash animations, respectively.

The four examples that now follow demonstrate the four sorts of animations that are supported.

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

##########################################
# (2) Animations in HTML pages: create an
# animation page for the Brownian Motion in
# the tempdir() and auto-browse it
oopt = ani.options(interval=0.05, nmax=100,
  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).", outdir = tempdir())
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")
par(opar)
ani.stop()
ani.options(oopt)

##########################################
# (3) GIF animations (require ImageMagick!)
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 (require SWF Tools!)
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)
```

Additionally, users can write their custom functions that generate animation frames, then use `ani.start()`, `ani.stop()`, `saveMovie()` and `saveSWF()` as required to create animations that can be viewed outside R.[3] The following commands define a function `jitter.ani()` that demonstrates the jitter effect when plotting discrete variables. This helps make clear the randomness of jittering.

```
# animation:example_jitter_effect
# function to show the jitter effect
jitter.ani <- function(amount = 0.2, ...) {
    x = sample(3, 300, TRUE, c(.1, .6, .3))
    y = sample(3, 300, TRUE, c(.7, .2, .1))
    i = 1
    while (i <= ani.options("nmax")) {
        plot(jitter(x, amount = amount),
            jitter(y, amount = amount),
            xlim = c(0.5, 3.5),
            ylim = c(0.5, 3.5),
            xlab = "jittered x",
            ylab = "jittered y",
            panel.first = grid(), ...)
        points(rep(1:3, 3), rep(1:3, each =
            3), cex = 3, pch = 19)
        Sys.sleep(ani.options("interval"))
        i = i + 1
```

[1] Use the command `convert` to convert multiple image frames into single GIF/MPEG files.
[2] The utilities `png2swf`, `jpeg2swf` and `pdf2swf` can convert PNG, JPEG and PDF files into Flash animations
[3] Very little change to the code is needed in order to change to a different output format.

```
    }
}

# HTML/JS
ani.options(nmax = 30, interval = 0.5,
    title = "Jitter Effect")
ani.start()
par(mar = c(5, 4, 1, 0.5))
jitter.ani()
ani.stop()

# GIF
ani.options(interval = 0)
saveMovie(jitter.ani(), interval = 0.5,
    outdir = getwd())

# Flash
saveSWF(jitter.ani(), interval = 0.5,
    dev = "png", outdir = getwd())
```

Animation examples

Currently, the **animation** package has about 20 simulation options. The website `http://animation.yihui.name` has several HTML pages of animations. These illustrate many different statistical methods and concepts, including the K-Means cluster algorithm (`mvstat:k-means_cluster_algorithm`), the Law of Large Numbers (`prob:law_of_large_numbers`), the Central Limit Theorem (`prob:central_limit_theorem`), and several simulations of famous experiments including Buffon's Needle (`prob:buffon_s_needle`).

Two examples will now be discussed in more detail – coin flips and the *k-nearest neighbor algorithm* (*k*NN).

Example 1 (Coin Flips) Most elementary courses in probability theory use coin flips as a basis for some of the initial discussion. While teachers can use coins or dice for simulations, these quickly become tedious, and it is convenient to use computers for extensive simulations. The function `flip.coin()` is designed for repeated coin flips (die tosses) and for displaying the result. Code for a simple simulation is:

```
# prob:flipping_coins
flip.coin(faces = c("Head", "Stand", "Tail"),
    type = "n", prob = c(0.45, 0.1, 0.45),
    col =c(1, 2, 4))
```

Suppose the possible results are: 'Head', 'Tail', or just 'Stand' on the table (which is rare but not impossible), and the corresponding probabilities are 0.45, 0.1, and 0.45. Figure 2 is the final result of a simulation. The frequencies (0.42, 0.14, and 0.44) are as close as might be expected to the true probabilities.

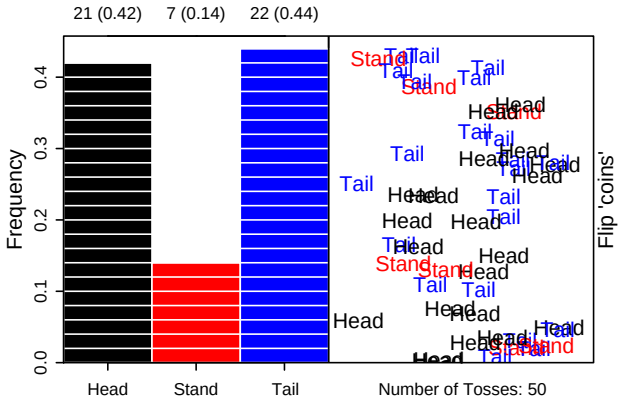

Figure 2: Result of flipping a coin 50 times. The right part of the graph shows the process of flipping, while the left part displays the result.

Example 2 (*k*-Nearest Neighbor Algorithm) The *k*-nearest neighbor algorithm is among the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors.

Basically the *k*NN algorithm has four steps:

1. locate the test point (for which we want to predict the class label);
2. compute the distances between the test point and all points in the training set;
3. find the *k* shortest distances and the corresponding training set points;
4. classify by majority vote.

The function `knn.ani()` gives an animated demonstration (i.e. the four steps above) of classification using the *k*NN algorithm. The following command uses the default arguments[4] to show the animation:

```
# dmml:k-nearest_neighbour_algorithm
knn.ani()
```

Figure 3 is a two-dimensional demonstration of the *k*NN algorithm. Points with unknown classes are marked as '?'. The algorithm is applied to each test point in turn, assigning the class ○ or △.

Our initial motivation for writing this package came in large part from the experience of students in a machine learning class. We noticed that when the lecturer described the *k*-NN algorithm, mentioning also *k*-fold cross-validation, many students confused the two '*k*'s. We therefore designed animations that were intended to help students understand the basic

[4]Note that the arguments of most animation functions have defaults.

ideas behind various machine learning algorithms. Judging from the response of the audience, our efforts proved effective.

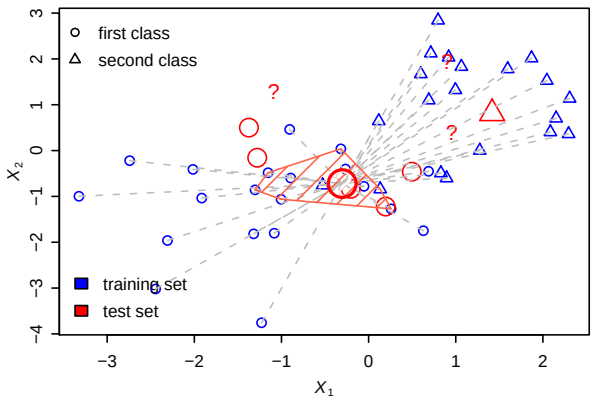

Figure 3: Demonstration of the *k*-nearest neighbor algorithm. The training and test set are denoted with different colors, and the different classes are denoted by different symbols. For each test point, there will be four animation frames. In this plot there are 8 circles and 2 triangles; hence the class '○' is assigned.

There are many other demonstrations in this package. Figure 4 is the third frame of a demonstration for *k*-fold cross-validation (`cv.ani()`). In that animation, the meaning of *k* is quite clear. No-one would confuse it with the *k* in *k*NN even when these two topics are talked about together. (Web page: `dmml:k-fold_cross-validation`)

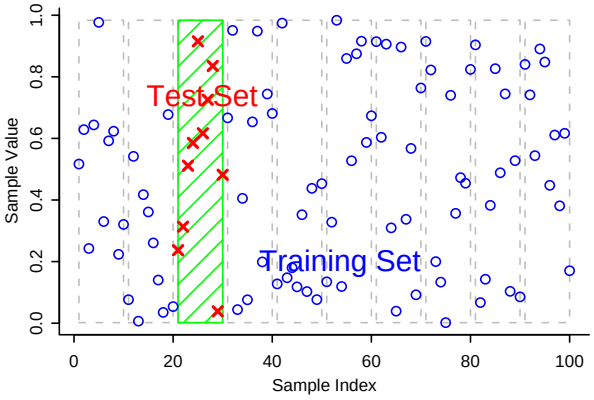

Figure 4: An animation frame from the demonstration of *k*-fold cross-validation. First, points are partitioned into 10 subsets. Each subset will be used in turn as the test set. In the animation, the subsets are taken in order from left to right.

Other examples that are on the website include the Newton-Raphson method (see Figure 5 and web page `compstat:newton_s_method`) and the Gradient Descent algorithm (see Figure 6 and web page: `compstat:gradient_descent_algorithm`).

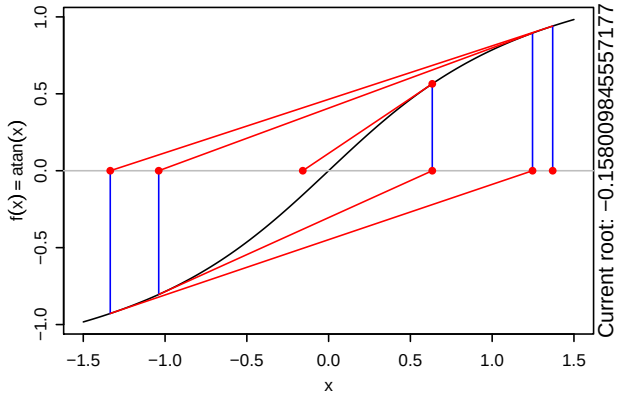

Figure 5: This is the fifth animation frame from the demonstration of the Newton-Raphson method for root-finding. The tangent lines will "guide" us to the final solution.

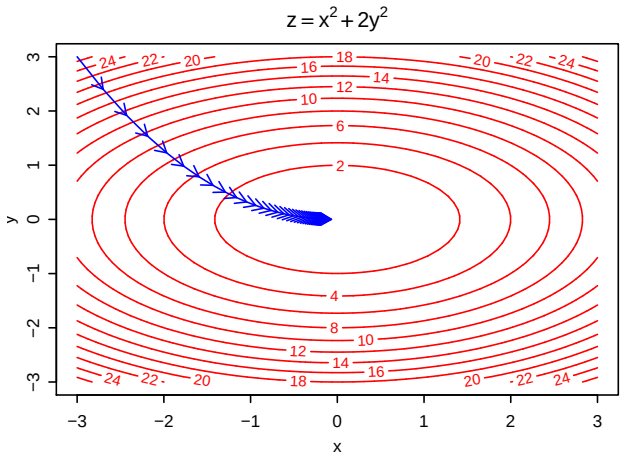

Figure 6: The final frame of a demonstration of the Gradient Descent algorithm. The arrows move from the initial solution to the final solution.

Summary

The R **animation** package is designed to convert statistical ideas into animations that will assist the understanding of statistical concepts, theories and data analysis methods. Animations can be displayed both inside R and outside R.

Further development of this package will include: (1) addition of further animations; (2) development of a graphical user interface GUI (using `Tcl/Tk` or `gWidgets`, etc), for the convenience of users.

Acknowledgment

We'd like to thank John Maindonald for his invaluable suggestions for the development of the **animation** package.

Bibliography

P. Murrell. *R Graphics*. Chapman & Hall/CRC, 2005.

Y. Xie. *animation: Demonstrate Animations in Statistics*, 2008. URL `http://animation.yihui.name`. R package version 1.0-1.

Yihui Xie
School of Statistics, Room 1037, Mingde Main Building, Renmin University of China, Beijing, 100872, China
`xieyihui@gmail.com`

Xiaoyue Cheng
School of Statistics, Room 1037, Mingde Main Building, Renmin University of China, Beijing, 100872, China
`chengxiaoyue@gmail.com`