A NEW SET OF SOUND COMMANDS FOR R; SONIFICATION OF THE HMC ALGORITHM

Matthias Heymann and Mark Hansen
Statistics Research, Bell Labs, Murray Hill, New Jersey 07974
www.MatthiasHeymann.de/mathematics.html
mail@MatthiasHeymann.de

Key Words: sonification, Hybrid Monte Carlo algorithm, R sound package, `wav` files.

Abstract:

We describe a new set of commands for the programming language R that we developed to work with sound samples and `wav`-files. The combination of these tools with the large number of statistical commands in the standard R package is an ideal environment for our efforts to sonify datasets.

As an example, we show how we use these commands to tune a parameter in the Hybrid Monte Carlo algorithm. Musicians will also enjoy the great possibilities for creating new sounds and sound effects on a very basic level and without any restrictions.

1. Introduction

The work of a statistician is dominated by dealing with large amounts of data from which one wants to derive useful information. In most cases intuitive graphical methods are used, but in higher-dimensional datasets these methods become more difficult to handle. We are looking at ways of "sonifying" data to develop tools that can help us understand graphics or that can even substitute graphical methods in certain situations.

For example, Speeth sonified in [1] seismograms by simply sending the recorded signal (with rescaled time axis) to the speaker to distinguish earthquakes from atomic explosions. In [2] Chambers, Mathews and Moore introduced a mapping technique that mapped a three-dimensional dataset to pitch, timbre and amplitude of a tone. Finally, Hermann, Hansen and Ritter used in [3] the sonification method described in this paper to tune parameters of the McMC algorithm. See [4] for other examples.

One of the computer languages used most by statisticians is R [5], an open source implementation of SPLUS [6]. Since sonification is not the usual way of treating datasets, the standard R package doesn't provide any commands for using sound or for dealing with `wav`-files, the standard file format for uncompressed sound files on PCs. So at the beginning of our research we had to implement a new set of commands that would let us try all our ideas without any restrictions and additional effort.

The sound package with full documentation can be found on the R development website [7].

2. The Sound Package for R

Let us start with some background on digital sound sampling and a description of how we handle sound samples in R. Then we will list some of the most important commands in the sound package.

2.1 Background on Digital Sound

Sound is oscillating air pressure that can be measured by our eardrums and interpreted by our brain as voices, music or simply as noise. Using a microphone and an AD (analog to digital) converter, the air pressure is measured in fixed time intervals and stored in the computer memory or, for example, on audio CDs. To play the sound sample again, a DA (digital to analog) converter is used to convert these values back into an analog voltage signal. This signal is sent to a loudspeaker and then translated into the movement of a cone in the speaker. The cone moves the surrounding air and reproduces the recorded sound. An illustration of the sampling process can be seen in Figure 1.

The quality of the recording depends on the goodness of the discretized air pressure curve which in

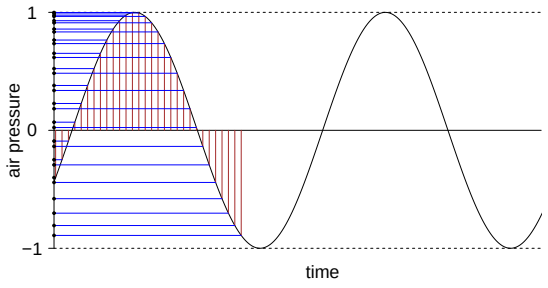

Figure 1: The principle of sound sampling.

turn depends on two parameters: the sampling rate and the number of bits per sample.

The sampling rate is the number of measurements per second. Sound samples with lower sampling rates need less memory but also have a worse quality because they cannot capture the high frequencies of the sound spectrum. In common audio CDs the standard sampling rate is 44,100 samples/second, high-end DAT-recorders even use 48,000 samples/second. Note that the word "sample" stands both for the whole sound recording (typically if they are not longer than a couple of seconds) and for single values in the discretization of the waveform.

The number of bits per sample (8 or 16) describes the exactness with which the measurements are recorded. Eight-bit sampling uses only half the memory but is a lot more noisy because the waveform is discretized to a range of only $2^8 = 256$ different values instead of $2^{16} = 65536$ in 16 bit sampling.

2.2 The Structure of a Sample Object

Since the structure of `wav`-files (the standard file format for uncompressed sound files on PCs) is based on this way to describe sound, we chose an equivalent description for our implementation of sound into R. The basic object of our sound package is a variable of the new class "Sample," that is a list with slots for

- the sampling rate
- the number of bits per sample
- the waveform matrix itself. This is a matrix with one or two rows (for mono or stereo sounds), each row representing one channel. The rows are sequences of values in $[-1, 1]$ that discretize the waveform of the sound.

Internally the components of the waveform matrix are saved as doubles, independent of the bits parameter. This principle ensures a higher quality for all manipulations of the sound. The bits parameter is only used when the sample is played or saved onto disk. As we will see later, these doubles can take values outside the interval $[-1, 1]$, but before the sample is saved or played it needs to be rescaled again. Otherwise this will result in loss of data and cracks in the sound.

Since these three basic parts of the Sample object are the same as used in a standard `wav`-file, Sample objects and `wav`-files are equivalent. That means that if a `wav`-file is loaded from disk, converted to a Sample object and then saved again, the new `wav`-file will be the same as the original one.

2.3 The New Commands of the Sound Package

The following section gives a brief overview of the most important commands of the sound package. It is assumed that the package is installed and loaded with the command `library(sound)`. The package and complete documentation can be found at [7].

2.3.1 Create, load, save and play Sample objects

Sample objects can be created with the command

```
sample <- as.Sample(sound,rate,bits)
```

where `sound` is the waveform matrix of the sample. Alternatively, one can use

```
sample <- loadSample(filename)
```

to load a `wav`-file on the hard disk and convert it into a Sample object. Similarly, one can save a Sample object as a `wav`-file, using

```
saveSample(sample,filename).
```

Since `wav`-files and Sample objects are equivalent, most commands of the package accept both filenames and sample objects as arguments. For example,

```
play(x)
```

will play either the given Sample object `x` in the R memory or the `wav`-file with the name `x`. When applied to a Sample object, the play command saves it as a temporary `wav`-file before it uses the standard media player (or a player of your choice) to play that file.

Finally, the commands `Sine`, `Square`, `SawTooth` (each with the parameters frequency and duration), `Noise` and `Silence` provide the programmer with some basic waveforms.

2.3.2 Access the basic parameters

Because it is often inconvenient to deal with samples that have different parameters (rate or bits) or a different number of channels, you can read these values with

```
rate(sample), bits(sample), channels(sample)
```

and convert the sample to another one with a new parameter with

```
newsample <- setRate(sample,newrate)
```

and similar commands for the number of bits and channels. The command `sound(sample)` returns the waveform matrix of a Sample object.

2.3.3 Manipulate sounds

The most basic sound manipulation is to play two different samples at a time. Physically this means simply adding the two corresponding waveforms, and therefore in our implementation of this process we chose the syntax

```
newsample <- sample1 + sample2.
```

If the two initial samples have different parameters, the algorithm first calls the commands mentioned in paragraph 2.3.2 to change these parameters to the higher sampling rate, number of bits per seconds and number of channels of the two samples.

Similarly, the product of two samples is defined. This makes sense, for example, if you multiply a sample with a sine tone with very low frequency $f \lessapprox 20$ Hz. This lets the amplitude (loudness) of the first sample oscillate with frequency $2f$.

Here the programmer has to take care that the range of the new waveform is contained in the interval $[-1, 1]$. This can be achieved by multiplying the sample with a scalar (`const*sample`) or by using the `normalize` command that determines the right scaling factor so that the full range $[-1, 1]$ is used.

Other common ways to modify samples are

```
appendSample(sample1,sample2)
```

that appends two Sample objects (where again filenames are accepted also) and commands to cut certain parts out of a sample:

```
cutSample(sample,time1,time2),
sample[n1:n2].
```

While the first version uses seconds as the unit for the start and end times, the second one is more precise and uses the indices of the columns in the waveform matrix. The command

```
noSilence(sample,level)
```

cuts off silence at the beginning and at the end of the sample,

```
reverse(sample)
```

returns the sample played backwards, and the command

```
pitch(sample,semitones)
```

changes the pitch of a sample by rescaling the time axis of the waveform by the factor $2^{\text{semitones}/12}$. Note that pitching a sample 12 semitones up (= 1 octave) means to play it with double speed (and therefore with doubled frequencies). The command

```
newsample <- panorama(sample,pan)
```

narrows the panorama of a stereo sample by adding a part of the right channel to the left and vice versa, with `pan` giving the strength of this effect.

Useful commands for actually working with sound samples are `print` and `plot`. `print(sample)` prints the most important information about a Sample object on the screen, such as rate, bits, channels, memory usage and duration; `plot(sample)` plots the waveform, which can be an important tool when you want to analyze the results of your sonification routines.

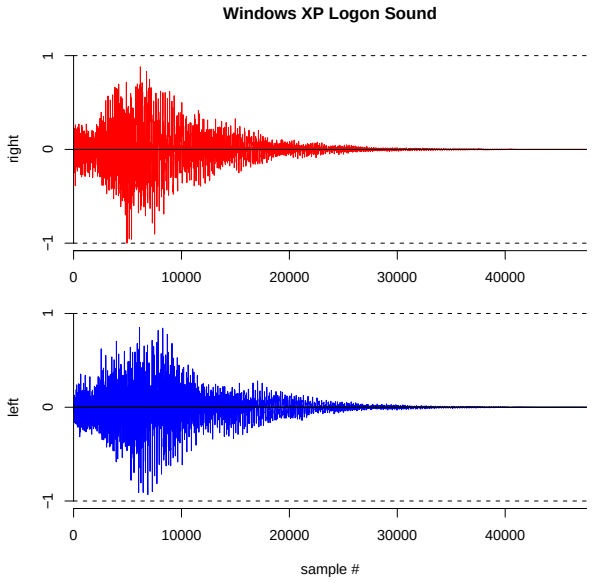

Figure 2: A plot of the Windows XP logon sound after applying the `normalize` command.

2.4 The Interplay with Other R Commands

When we were looking for methods to transform the output of our algorithms into good-sounding samples, we found that we could use the powerful routines that already exist in the standard R package. For example, we constructed a high-pass filter by simply applying one of the existing smoothing routines with a low degree of freedom to our data sequence and choosing the residuals of the fitting process as the waveform of the filtered sample.

To illustrate the strength of R for sonification, let us see how easily the above procedure could be realized. If `s` is the initial sample, we use the `sound()` command to get access to its waveform matrix and `sampleLength()` to read the number of columns of this matrix. In signal processing terms, `df` is related to the cutoff frequency of the filter.

```
design <- ns(1:sampleLength(s),df)
for (i in 1:channels(s)){
  fit          <- lm(sound(s)[i,]~design)
  sound(s)[i,] <- residuals(fit)
}
play(s)
```

In other contexts, especially with longer initial samples, other smoothing routines might be more appropriate, for example the usage of wavelets, and one can profit from the commands provided by the wavelet package for R.

3. Sonification of the Hybrid Monte Carlo Algorithm

3.1 The HMC Algorithm

We will now describe how we used these commands for sonification.

Statisticians often need values of a random variable with a given probability distribution. One of the algorithms that is used in these situations is the Hybrid Monte Carlo (HMC). This is a special version of the Metropolis sampler in which the rule for picking a new sample that might or might not be accepted as the next value of the algorithm is motivated by the Hamiltonian equations for Newton's law of motion.

To be more precise, suppose that we want to sample from a distribution $f(x) = \exp(-U(x))$, $x \in \mathbb{R}^n$. Instead of sampling from f we will sample from $g(x,p) \propto \exp(-H(x,p))$ with $H(x,p) = U(x) + \frac{m}{2}p^2$, $p \in \mathbb{R}^n$, and then simply forget about p. Now starting from an arbitrary pair (x_0, p_0), the nth step of the HMC is as follows:

(i) Generate a new vector p from the Gaussian distribution $\propto \exp(-\frac{m}{2}p^2)$,

(ii) regard (x_{n-1}, p) as the start point in the phase space of a mass m in the potential U and follow Newton's law of motion for a fixed amount of time Δt, ending in (x_n, p_n).

So far this is known as the Molecular Dynamics (MD) algorithm that has been proven to generate samples from f. But to correct the calculation error that inevitably occurs in the integration of the Hamiltonian equations in (ii), one uses the acceptance-rejection-rule from the Metropolis sampler as an additional step (iii). For more details we refer the reader to [8, chapter 9].

Now according to the theory, the distributions of the samples x_n tend to the desired distribution f. But it is in general hard to say how fast they converge and how long we have to wait before we pick the first sample. Another problem is to choose the right value of Δt: If we choose it too large then we waste calculation time, if we choose it too small then subsequent samples will be strongly dependent, and it will take more time for the algorithm to converge.

3.2 Sonification

Usually graphical methods are used to determine whether the samples are sufficiently "random". Since in higher-dimensional problems these methods begin to become difficult to handle, we thought about various ways to sonify these values. Similar work was done by Herman, Hansen and Ritter in [3] who sonified McMC algorithms.

Since the sequence of random variables can be regarded as a variable that changes over time, in this case it turned out that one promising way to sonify the data is to sonify each value in a certain way (using the underlying distribution f) and then to append the sequence of sound samples. So one would expect the development of the sound to be related to the changes of the random variable over time and therefore to the "randomness" of the sequence.

As in [3], our sonification model is to regard the given value as the starting position of a ball in a landscape with mountains and valleys, given by the negative distribution function $-f(x)$. The ball will start to move, and its kinetic energy will go up and down. The development of the kinetic energy is then used as the waveform of the sound sample for the given value. The character (especially the frequency) of the tone is strongly dependent on the starting position of the ball.

In contrast to [3] we could now do both the statistical and the sonification calculations in only one piece of code, in the same programming language.

3.3 Example Sounds

To get an impression of our sonification results you can listen to two example `wav`-files at [9]. To be able to compare sonification with graphical methods we chose a simple two-dimensional distribution function and drew 100 samples for each test run of the algorithm.

In the first one we chose Δt very small. Therefore the distance between subsequent samples is forced to be small (see the first picture in Figure 3), and they are strongly dependent. Consequently, you can hear that the sound changes only a little between subsequent samples.

For the second `wav`-file we chose Δt about 80 times larger, which seems to be enough: The second picture in Figure 3 shows that now it is easier for the algorithm to explore the whole x-space within only

a couple of steps, and as a result the sound sequence obviously changes more randomly. You can also hear how the algorithm needs some time to get started.

We conclude that although, of course, this sonification process cannot guarantee that the parameter Δt is well-chosen, it can help to see if Δt is chosen incorrectly.

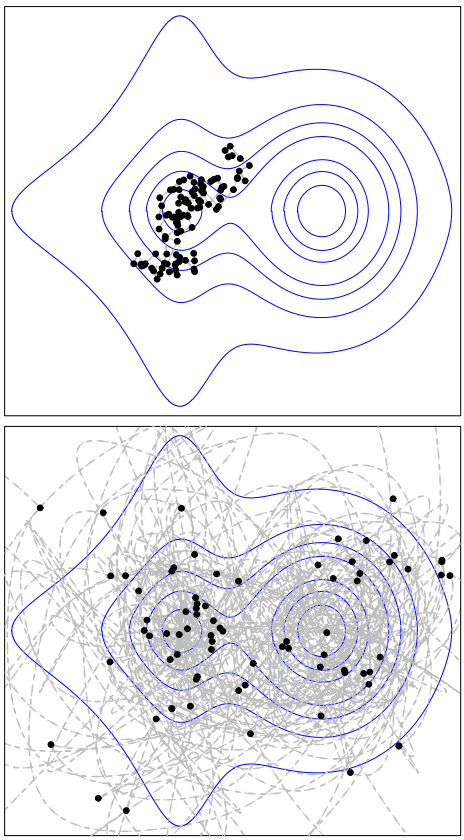

Figure 3: Two random walks of the HMC algorithm on a 2-dimensional distribution function (blue contour lines). The samples x_n are plotted in black, the ways of the mass m in step (ii) of the algorithm are indicated in grey.

4. New Possibilities for Musicians

During our statistical experiments with sound we quickly discovered the potential of the sound package for the purpose of electronic music. Effects like chorus

```
s <- Sine(440,1)
play(normalize(s+pitch(s,.2)))
```

and simple reverb as well as typical synthesizer techniques like loops and ADSR (Attack-Decay-Sustain-Release) amplitude envelopes can be coded in a couple of lines, and you can even imagine a complete (non real-time) synthesizer programmed in R.

Since the structure of our implementation allows you to work on the very basic level, everyone can invent his own waveforms and sound effects without the restrictions of many effect generators that only allow you to set the parameters for existing routines. To get a taste of it, enjoy this nice sound

```
wav <- 2*((seq(0,80,length=88200)^2)%%1-.5)
play(as.Sample(wav,44100,16))
```

that can also be found at [9].

5. Conclusion

We described our new set of sound commands for the programming language R and how we used it to sonify the samples of the Hybrid Monte Carlo algorithm. The result of our sonification process cannot guarantee that the parameter Δt in the HMC algorithm is well-tuned, but it can help to see that it is chosen incorrectly.

We hope that this work will motivate statisticians to experiment with the new commands and to try their own sonification ideas. Up to now not very much mathematical research has been done in this field, and there are still a variety of possible applications to discover. The sound package will make this kind of research easy for everybody, even for those without any background in electronic music.

References

[1] S.D. Speeth, *Seismometer sounds*, in J. Acoust. Soc. Amer. **33** (1961), pp. 909-916

[2] J.M. Chambers, M.V. Mathews and F.R. Moore. *Auditory Data Inspection*, Technical Memorandum no. 74-1214-20, AT&T Bell Laboratories, 1974

[3] T. Hermann, M.H. Hansen and H. Ritter, *Sonification of Markov chain Monte Carlo simulations*, in Proc. of the 2001 Int. Conf. on Auditory Display (ICAD), Espoo, Finland, pp. 208-216

[4] `www.icad.org`

[5] `www.r-project.org`

[6] `www.insightful.com`

[7] `cran.r-project.org`, click on "Package Sources" and then on "sound" in the list of packages

[8] J.S. Liu, *Monte Carlo Strategies in Scientific Computing*, Springer 2001

[9] `www.MatthiasHeymann.de/mathematics.html`