Digital Audio Coding
Lab Session 2 – SHRINK

Sébastien Boisgérault,
Mines-ParisTech.

Feb. 19, 2015

About SHRINK

SHRINK is an simple audio file format that relies on lossless audio compression algorithms and hence is similar to FLAC, ALAC or SHORTEN.

The command-line program `shrink` implements a SHRINK *codec*: the command

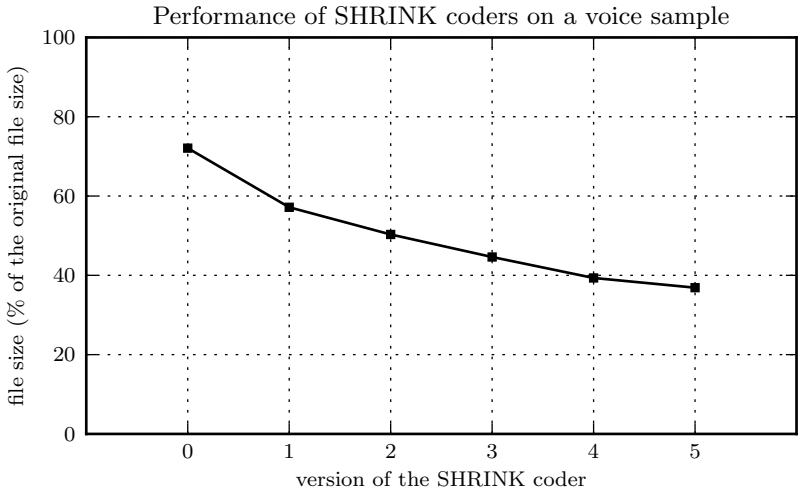
```
$ shrink sound.wav
```

turns the uncompressed WAVE file `"sound.wav"` into a `"sound.shk"` file (only 44.1 kHz, 16-bit linear PCM content is supported) and conversely the command

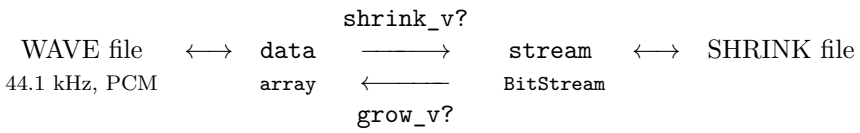
```
$ shrink sound.shk
```

uncompresses the SHRINK file `"sound.shk"` into `"sound.wav"`. The command `shrink --help` provides more options.

The `shrink` program is implemented on top of a Python module named `audio.shrink`. The SHRINK format actually gathers different codecs versions – numbered 0 to 5 – of increasing complexity and performance.



Each version of the codec is defined by two functions: a coder `shrink_v?` and a decoder `grow_v?` where `?` stands for the version number; these two functions transform `data` – a numpy array of type `int16` – into `stream` – a bitstream – and reciprocally, according to the following diagram:



The aim of the lab session is to study the algorithms behind the design of SHRINK and to implement the main features of this coder.

SHRINK Format Structure

The high-level structure of the SHRINK file format is given by:

Size	Unit	Content
89	bit	header
?	bit	channel 1
?	bit	channel 2
< 8	bit	zero-padding

where the content of the header is:

Size	Unit	Name	Type	Range/Value
6	byte	magic	ASCII, big endian	`"SHRINK"`
1	byte	version	unsigned integer	`[0, ..., 5]`
4	byte	length	unsigned integer, big endian	number of samples (per channel)
1	bit	stereo	boolean	`True` or `False`

The format of the channel fields is version-dependent.

1. What is the purpose of the zero-padding in this context ?
2. Generate a SHRINK bitstream with no channel data (for example with version 0 and a single channel). Make sure that the function `grow_v0` from the Python module `audio.shrink` can successfully decode this stream.

Amplitude Rice Coder

For each channel, the version 0 of the SHRINK compression algorithm:

- determines an appropriate value for the Rice parameter b (fixed bit width),
- encodes b as an unsigned 8-bit integer,
- encodes the channel array with this configuration of the Rice coder.

Table 3: SHRINK version 0 channel format.

Size	Unit	Content	Type
8	bit	Rice parameter	unsigned integer
?	bit	sample values	Rice encoded data

1. Explain why all Rice parameters fit into the range of unsigned 8-bit integers.
2. Consider a pure tone with frequency 440 Hz and maximal amplitude, represented as an array `A4` of 16-bits signed integers. Use the heuristic provided in the `rice` coder to compute the "best" value of the Rice parameter b for this data. Check the size of the coded data for Rice parameters between $b - 3$ and $b + 3$.
3. Implement the coder function `shrink_v0`. Test it on the data `A4`. Did we achieve any compression ? Why ? Generate a sound similar to `A4`

but more likely to be effectively compressed by `shrink_v0`, then test your hypothesis.

Simple Prediction Coders

Convention: finite sequences of numeric values $(x_0, x_1, ..., x_{N-1})$ – represented as one-dimensional arrays of length N – are assimilated to infinite sequences indexed on $\mathbb{Z}$, denoted $(x_n)_n$, with a value of 0 outside of the initial index range.

The difference operator Δ on sequences is defined by

$$\Delta(x_n)_n = (x_n)_n - (x_{n-1})_n$$

1. Implement the restriction of Δ to arrays as a function `D` (hint: use the function `numpy.diff`). Make sure that the length of the output array is the same as the input array. Is the operator Δ invertible ? What about the function `D` ? Implement the inverse function (hint: use the function `numpy.cumsum`).

   Implement version 1 of the SHRINK coder `shrink_v1` that applies the Rice coder to the difference of the audio PCM data:

   Table 4: SHRINK version 1 channel format.

Size	Unit	Content	Type
8	bit	Rice parameter	unsigned integer
?	bit	sample difference	Rice encoded data

   Check that the best Rice parameter for the difference values is smaller than the best Rice parameter for the original sample values. Compute the compression rate achieved on the data `A4`.

2. A *predictive coder* encodes the *prediction residual* $e_n = x_n - \hat{x}_n$ where $\hat{x}_n$ is a prediction of x_n based on the past values of this sequence. Show that version 0 and 1 of SHRINK are simple predictive coders. What is their prediction of x_n ?

3. Implement version 2 of the SHRINK coder `shrink_v2` that replaces the previous predictor schemes with a linear extrapolation of x_n based on x_{n-1} and x_{n-2}.

   Table 5: SHRINK version 2 channel format.

Size	Unit	Content	Type
8	bit	Rice parameter	unsigned integer

Size	Unit	Content	Type
?	bit	linear extrap. residual	Rice encoded data

Study the reduction of the Rice parameter for the new residual. Compute the new compression rate achieved on the data `A4`.

Adaptative Polynomial Prediction Coder

For any $n \in \mathbb{Z}$ and any numbers x_{n-m-1}, x_{n-m}, ..., x_{n-1}, there is a single polynomial P_n of order (at most) m such that:

$$P_n(n-m-1) = x_{n-m-1},\ P_n(n-m) = x_{n-m}, ...,\ P_n(n-1) = x_{n-1}.$$

The *polynomial prediction of order* m maps a sequence $(x_n)_n$ to the sequence $(\hat{x}_n)_n$ defined by $\hat{x}_n = P_n(n)$.

1. Describe versions 1 and 2 of the SHRINK coders in terms of polynomial prediction. Give in both cases an expression of the prediction residual $(e_n)_n = (x_n)_n - (\hat{x}_n)_n$ using the difference operator Δ. Use these results to guess the expression of the residual for the polynomial prediction of order m. Prove the result by an induction on the prediction order.

2. Show for a given prediction order m, there is a unique sequence of integers $(a_1, a_1, ..., a_{m+1})$ such that for any $n \in \mathbb{Z}$:

   $$\hat{x}_n = a_1 x_{n-1} + \cdots + a_{m+1} x_{n-m-1}$$

   Hence, the prediction residuals of a sequence of integers are integers.

3. Implement a function that, given a signal, computes the best Rice parameter for the coding of its amplitude, then compute the best Rice parameter for the coding of the residual by the polynomial prediction of order 0, 1, etc. and stops as soon as this value stops to decrease, in order to determine practically the "optimal" prediction order and the corresponding residual. We use the convention that no prediction (coding of the amplitude values) corresponds to a prediction order of -1.

4. Implement version 3 of the coder `shrink_v3`, with the following layout:

Table 6: SHRINK version 3 channel format.

Size	Unit	Content	Type
?	bit	prediction order + 1	Rice encoded data. Rice param. 3, unsigned.

Size	Unit	Content	Type
8	bit	Rice parameter p	unsigned integer
?	bit	prediction residual	Rice encoded data. Rice param. p, signed.

Compute the new compression rate achieved on the data `A4`.

Framed Prediction

The properties of an audio signal may change completely in different sections of its time frame but he still probably has some local consistency. We may therefore search for the best predictor locally instead of globally and expect higher compression ratios.

1. Implement the `shrink_v4` coder that splits a signal into frames of 20 ms, and applies the channel data coder from the version 3 to each frame separately.
2. This method has the drawback to include a "cold restart" of the algorithm for each new frame: every frame analysis is done as if the previous values of the signal were all zeros and the first predicted values are therefore probably very inaccurate. Modify the previous coder into `shrink_v5` to solve that problem.
3. Study the performance of version 4 and 5 of the coders with respect to the version 3 on smooth synthetic signals, such as `A4`, and on more realistic signals, such as music or voice signals.