INTERACTIVE SONIFICATION WITH CSOUND

INTRODUCING ICSOUND

Andrés Cabrera
andres AT mat.ucsb.edu

Auditory displays present information through sound. As part of an auditory display, the process of rendering information and interaction as sound is called *sonification. Sonification* can take many forms and be applied to many different problems: from understanding radiation through the clicks from a Geiger counter to developing the complex sound language presenting information in some computer games today. The study of sonification is very developed and a scientific community with expertise in sound synthesis, big data, user interaction, computer science and cognition (among others!) has gathered together around it [1].

There are essentially three ways to sonify data:

- **Audification:** The direct mapping of one-dimensional data to an audio signal. A typical example of audification is the earthquake data, where the measurements from seismographs are directly transformed into audio. Audification of earthquakes allows researchers to understand and compare properties of earthquakes from their sonic signature [2].

- **Parameter mapping sonification:** Data is used to control parameters of synthesis. This can be as simple as mapping a value to the frequency of an oscillator or as sophisticated as controlling multiple parameters of a granular synthesizer. This technique allows sonification of multidimensional datasets, but the great challenge is mapping parameters and ranges in a way that makes sense for the data and that allows characteristics in the data to be perceived.

- **Model based sonification:** The data itself is the model. The user interacts by "exciting" the model. There have been various techniques of this sort proposed, for example by mapping the data in space and then using physical interfaces (tactile, 2D, 3D, etc.) that trigger and affect sound according to the data currently being "scrubbed" [3]. Another example called "particle trajectory sonification model" turns data into a multidimensional gravitational field, where the user interacts by shooting "masses" into the field, and their speed affects the phase from oscillators. This technique can help identify clusters, which will produce a very different sound to very regularly spaced "masses" [4].

Although much effort has been put into developing models and techniques for sonification, it seems that there is less thought put into the process of sonification design: exploring and defining sonification models to select one that works with the data. What “works” means here depends entirely on the data and the purpose of the sonification. Some techniques will display the data better, some techniques might be too obnoxious when used for a long time for the purpose of data exploration and discovery, some techniques might have better aesthetic values and possibly provide poetic resonances which might be important when the sonification is part of an artwork. For example like in Chris Chafe's “Oxygen Flute”, where CO_2 and oxygen levels from sensors within a chamber filled with bamboo plants generate music through a physically modeled flute.

1 Sonification and Csound

Csound by itself provides the necessary tools for interactive sonification. It is a powerful synthesis engine with various mechanisms for interactive control. However, one thing it lacks is simple and performant data management and processing tools. When working with sonification it is common to have large multidimensional data sets. Although recent additions to Csound like multidimensional arrays can make this simpler, it can still be cumbersome to import, slice, transform and segment data. Additionally data for sonification is often stored in formats like matlab, XML, or JSON that can be challenging to import and work with using only Csound. For this reason, combining Csound with a language like python is ideal for sonification, as Csound can produce a robust and rich audio synthesis engine, while python can provide a means to interface with the data, to visualize it and to assist in the exploration of sonification models and synthesis techniques applied to the data.

2 Sonification Workflow

There are three stages involved in the process of sonification of data:

1. Data loading: This involves bringing the data from some container format or remote server into the sonification engine. The data may need to be trimmed to select only parts of it, or in some cases multiple diverse sets of data may need to be bundled together.

2. Sonification design: This is the process of trying out different synthesis techniques using the data as synthesis parameters, as well as developing different mapping techniques and

3. Interaction with the sonification: Once the sonification design is complete, the user can interact with the sonification through the controls/parameters available. Ocasionally, there might be no interactivity as the data itself might just be presented without any user control. This third step might also be part of the iterative design process where the experience of interaction serves to inform further sonificaiton design and data pre-processing.

3 icsound

icsound is a python module designed to be used in the ipython notebook [5]. The ipython notebook provides an interactive shell to python within a browser that can display inline plots. It provides an interactive data exploration environment that is widely used today in very diverse areas of science. But the nature of the notebook by itself precludes easy interaction with sound as it is designed to evaluate a group of lines (called a “cell”) at a time to then produce output, in diverse forms: graphs, files, transformations of data in memory, etc. Although python provides facilities to read and write audio files, what is needed for interactive sound work is a system like Csound which can run and generate audio in a separate thread and which can act as a server receiving commands to change and adjust the sound producing algorithms. So by combining both python and Csound through the ipython module, we can bridge the gap between data and sonification in an interactive environment.

In the ipython notebook, both Csound and python function as servers listening to commands, which open the possibility of collaborative sonification design and exploration. One option is to use Csound as the interactive server where various users can send commands to the *icsound* server to affect parameters and define instruments and note events. This method is somewhat limited, because it means that each user is running its own instance of python, and is only sending commands through the network to a single running instance of *icsound* (see section on *Collaborative Server* below). This can be sufficient in many instances, but since users are only controlling Csound, the state and data in the python interpreter is different for each participant. Another interesting possibility is running the ipython server and icsound in a single machine, and having multiple participants connecting to it across a network through the ipython notebook facilities.

Basic Usage

The *icsound* module is distributed with recent versions of Csound, starting with version 6.04. To use the *icsound* module, it should be imported like any other python module:

```
import icsound
```

This will bring in the icsound class as well as activate the csound “magics” for ipython (see below for details). Then a csound engine can be instantiated:

```
cs = icsound.icsound()
cs.start_engine()
```

The engine can be started with the default parameters, or arguments to the *start_engine()* function can define the audio engine parameters. The full signature for the function is:

```
start_engine(self,   sr=44100,   ksmps=64,   nchnls=2,   zerodbfs=1.0,
dacout='dac', adcin=None, port=12894, buffersize=None)
```

The meaning of the arguments should be self explanatory, except for the *port* argument, which sets the port on which the csound engine will listen to on the network for commands and the *buffersize* argument which sets. This can allow the csound engine to act as a server which can receive instruments and score events from other machines.
A useful debugging command is:

```
cs.print_log()
```

This will print the Csound message buffer. When the message queue is long and becoming cumbersome, you can call this function to clear the message buffer:

```
cs.clear_log()
```

To stop a running engine, or disconnect from a remote engine instance, call:

```
cs.stop_engine()
```

More verbose debugging information can be printed by *icsound* if the verbosity is set to True:

```
cs.setVerbose(True)
```

Collaborative server

As discussed above, an *icsound* engine is by default a server, so it can be controlled directly from te same python process as well as through the network.
To connect to a running Csound server, you can create a new *icsound* object and then call the *start_client()* function instead of *start_engine()*.

```
cs_client = icsound.icsound()
cs_client.start_client()
```

This client is only useful when it connects to a running server. By default, a client will send commands on the local host on port 12894, but the address and port can be set using the full signature of the function:

```
start_client(self, address='127.0.0.1', port=12894)
```

Sending instruments to Csound engines

Csound language code can be entered directly in the ipython notebook shell using what ipython calls "magics". For example, a cell containing:

```
%%csound
gkinstr init 1
```

Will set the global variable *gkinstr* to 1. Notice that you can evaluate any i-rate code or send instrument definitions using the "csound magic": `%%csound`.

```
%%csound
instr 1
asig oscil 0.5, 440
outs asig, asig
endin
```

This will send instrument 1 to the current *icsound* engine. The "current" *icsound* engine is the last one to be created, so if you are handling multiple instances, be aware that you can only send Csound through the "csound magic" to that specific instance. To communicate with other instances, you will need to use the *send_code()* and *send_event()* functions:

```
send_score(self, score)
score_event(self, eventType, pvals, absp2mode = 0)
```

If the orchestra code fails to compile, the error output from Csound will be printed directly in the ipython notebook.

Exchanging function tables

An important practical aspect of sonification is being able to transfer data to and from the audio engine. Data will likely be stored in python either as a python list or as a numpy array. For Csound to make use of it, one simple way is to copy that data from python into a Csound f-table. To do this, icsound provides the *fill_table()* function that takes a table number and a python array (either a python list or a numpy array) and transfers it into a Csound f-table:

```
fill_table(self, tabnum, arr)
```

The following two lines are valid for *icsound*:

```
cs.fill_table(1, [4,5,7,0,8,7,9])
cs.fill_table(2, array([8,7,9, 1, 1, 1]))
```

If the table does not exist, it is created, and if it does, it will be resized to the length of the python array. You need to be careful here if the table is being used by a running instrument, as this could cause a crash! Also, be aware that you need to pass one dimensional arrays, so you might need to slice the numpy array if it has more than one dimension.

It's also possible to generate tables using Csound's GEN functions using the *make_table()* function:

```
make_table(self, tabnum, size, gen, *args)
```

For example, to create an f-table that holds a sine wave, you would use Csound's GEN function 10 with a single argument of 1:

```
cs.make_table(3, 1024, 10, 1)
```

You can also bring data from Csound back to python using the *get_table_data()* function:

```
get_table_data(self, t)
```

For example:

```
In [10]: cs.get_table_data(1)
Out[10]: array([ 4.,  5.,  7.,  0.,  8.,  7.], dtype=float32)
```

And you can use *icsounds* plotting facilities to generate graphs that are styled in a convenient way for the common usage in Csound:

```
cs.plot_table(1)
cs.plot_table(2, reuse=True)
```

If *reuse* is set to *True* then the graph will be plotted in the existing figure, overlaid onto the previous graphs, otherwise a new figure is created.

If the ctypes python module is available then copying data back and forth will be relatively fast, as pointers are exchanged between the python kernel and the Csound C

API, so copying is an operation in C rather than python which can make a significant difference for large data.

Control parameters

Values can be sent to the Csound engine using Csound's bus mechanism. This is a convenient way to set parameters for an audio alorithm. It might be less efficient and there is no way to control accurate timing for this data since it is transmitted asynchronously, but it provides a simple and clear way to handle parameters. Buses in Csound are referred by name, and you can use the *invalue/*outvalue *chnset/chnget* Csound opcodes to read and write to/from channels. You can set/get values from python with the *icsound* functions:

```
set_channel(self, channel, value)

get_channel(self, channel)
```

Recording the output from Csound

The *icsound* module provides simple functions to record the output from Csound to an audio file without needing to write Csound code. This is done through new facilities in the Csound API, using an efficient and thread-safe lock-free queue, which prevents dropouts while maintaning responsiveness and low latency.

```
start_record(self, filename, samplebits = 16, numbufs=4)
cs.stop_record()
```

4 icsound in practice

This section will present two examples that can illustrate how icsound simplifies and assists in the design of sonification. The first example will deal with audification of seismographic data and the second with parameter based sonification of market data. The acquisition of data will be presented which is an important part of the process.

Audification with icsound

Audification of earthqueakes was one of the early applications of sonification. This simple technique turns the vibrations picked up by seismographs directly into sound vibrations. The ear is an instrument that is very good at picking up variations and changes in timbre over time, and these are characterized by the auditory system to give each earthquake audification a unique and distinguishable sound. Also, the ear can gauge how similar or different two events are. Audification can also provide other interesting “physical” cues as they can sound like rumbling, scraping, hammering, etc.
The first challenge in sonification is often the acquisition and injection of data, particularly if the data was generated by a third party. The Incorporated Research Institutions for Seismology (IRIS) provides an online API to retrieve earthquake events. Python can be used to access this web API, and get the data as a string:

```
prefix = 'http://service.iris.edu/irisws/timeseries/1/query?'
SCNL_parameters = 'net=IU&sta=ANMO&loc=00&cha=BHZ&'
```

```
times = 'starttime=2005-01-01T00:00:00&endtime=2005-01-02T00:00:00&'
output = 'output=ascii'
import urllib2
f = urllib2.urlopen(prefix + SCNL_parameters + times + output)
timeseries = f.read()
```

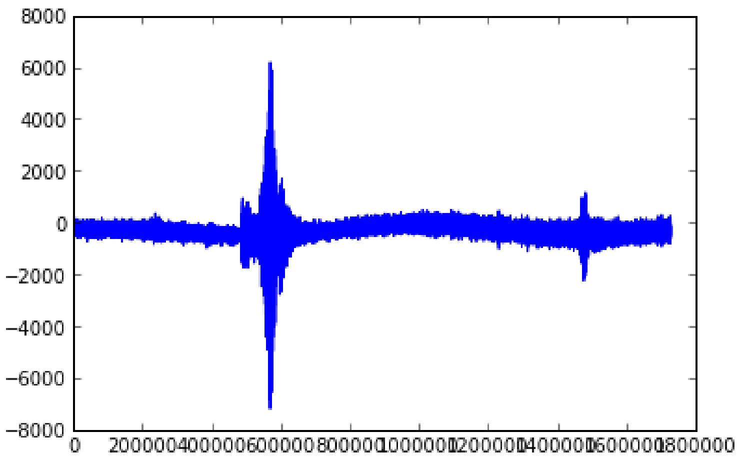
The data in this case is separated by new lines with the first line being metadata and the last line being empty. To turn the data within this string into a python data structure we can use this code:

```
data = timeseries.split('\n')
dates = []
values = []

for line in data[1:-1]:
    date, val = line.split()
    dates.append(date)
    values.append(float(val))
```

We can then plot the data:

```
plot(values)
```



To sonify the data, we can push it to a csound table:

```
import icsound
cs = icsound.icsound()
cs.start_engine()
cs.fill_table(1, values)
```

and then use an instrument to play it back. The instrument can be defined using csound code using the ipython magic:

```
%%csound
instr 1
idur = p3
itable = p4
asig poscil 1/8000, 1/p3, p4
outs asig, asig
endin
```

Then we can hear the earthquake with:

```
cs.send_score('i 1 0 3 1')
```

By changing the duration of the note event, the pitch of the audification is also altered. To expand the sonification, more data can be loaded into more csound f-tables, and they can be played interactively to allow for easy comparison. This process of loading can be automated to select time periods or geographical zones to quickly compare data. The playback can also be automated -either in python or csound- to listen to sets of earthquake data, for instance:

```
curtime = 0
padding = 1
score = ''
for ftable in range(10):
    for dur in range(5):
        score += 'i 1 %i %i %i'%(curtime, dur, ftable)
        curtime += dur + padding
```

Parameter based sonification

To realize parameter based sonification, one of the key decisions is the type of synthesis whose parameters wil be modified by the data. In this example, Phase Modulation synthesis will be chosen as it can produce rich timbral differences from few parameters. The source of the data will be stock data pulled from the yahoo finance API in XML format. The following code brings the data into python:

```
import xml.etree.ElementTree as ET
import urllib2
from datetime import datetime

symbols = 'MSFT', 'GOOG'

data = {}

for symbol in symbols:
    url = 'http://chartapi.finance.yahoo.com/instrument/1.0/' + \
        symbol + '/chartdata;type=quote;range=1y/xml'
    f = urllib2.urlopen(url)
    root = ET.fromstring(f.read())

    dates = []
    values = []
    for v in root[2].findall('p'):
        dateval = v.attrib['ref']
        date             =             datetime(year=int(dateval[0:4]),
month=int(dateval[4:6]), day=int(dateval[6:8]))
        dates.append(date)
        entryvalues = []
        for num in v:
            entryvalues.append(float(num.text))
        values.append(entryvalues)
    data[symbol] = [dates, array(values)]
```

We then need to instantiate the icsound engine and copy the data into csound f-tables:

```
import icsound
cs = icsound.icsound()
cs.start_engine()
cs.fill_table(1,data['GOOG'][1][:,1] - min(data['GOOG'][1][:,1]) )
cs.fill_table(2,data['MSFT'][1][:,1] - min(data['MSFT'][1][:,1]) )
```

Finally we need to define a phase modulation index that uses these f-tables as parameters to the synthesis:

```
%%csound
gisine ftgen 0, 0, 4096, 10, 1

instr fm

    amoddepth poscil 1/10, 1/p3, 2
    acarfreq poscil 3, 1/p3, 1
    icmratio = 5

    aphs phasor 220 + acarfreq

    aphsmod oscili amoddepth/2, acarfreq*( icmratio)

    asig table3 aphs + aphsmod, gisine, 1, 0, 1
    aenv linen 0.2, 0.01, p3, 0.01
    outs asig*aenv, asig*aenv

endin
```

To listen to the sonification, we just need to instantiate the PM instrument:

```
cs.send_score('i "fm" 0 20')
```

Conclusions

The *icsound* module for ipython provides a convenient mechanism for the exploration and design of sonification. Since the design of interactive sonification experiences is a mix between finding the right synthesis techniques, the right sonification model and the right mapping of the data, it's only natural that the environment for this should be a tool that can integrate all these. The ipython notebook, together with the icsound module simplifies the process of sonification design, allowing it as a collaborative exercise.

Acknowledgements

The *icsound* module is partially based on earlier work by François Pinot and Jacob Joaquin.

References

[1] ICAD: International Commumity for Auditory Display. http://www.icad.org/
[2] Meier, M., & Saranti, A. (2008). Sonic Explorations with Earthquake Data. Paper read at ICAD 2008, 24.–27.6.2008, at Paris, France.
[3] T. Bovermann, T. Hermann, and H. Ritter. Tangible data scanning sonification model. Proceedings of the International Conference on Auditory Display (ICAD 2006), pp. 77–82, London, UK, 2006.

[4] T. Hermann, J. Krause, and H. Ritter. Real-time control of sonification models with an audio-haptic interface. In R. Nakatsu and H. Kawahara, editors, Proceedings of the International Conference on Auditory Display (ICAD 2002), pp. 82–86, Kyoto, Japan, 2002.

[5] Shen, Helen. Interactive notebooks: Sharing the code. *Nature* 515.7525 (2014): 151-152.