Reading PCM Audio with Python

June 19, 2009 by Scott | Python | Leave a comment

When I figured this out I figured it was simply way too easy and way to helpful to keep to myself. Here I post (for the benefit of friends, family, and random Googlers alike) two examples of super-simplistic ways to read PCM data from Python using Numpy to handle the data and Matplotlib to display it. First, get some junk audio in PCM format (test.pcm).

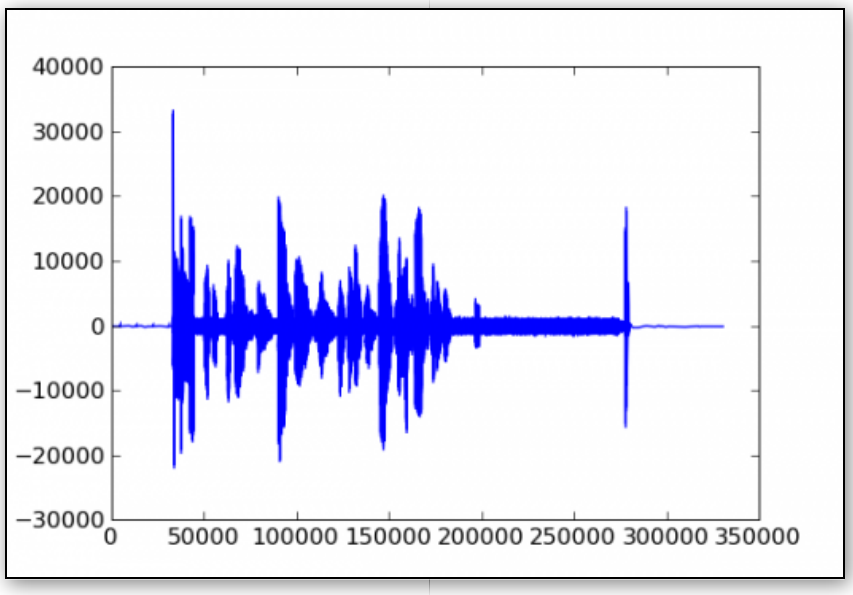
```
import numpy
data = numpy.memmap("test.pcm", dtype='h', mode='r')
print "VALUES:",data
```

This code prints the values of the PCM file. Output is similar to:
`VALUES: [-115 -129 -130 ..., -72 -72 -72]`

To graph this data, use matplotlib like so:

```
import numpy, pylab
data = numpy.memmap("test.pcm", dtype='h', mode='r')
print data
pylab.plot(data)
pylab.show()
```

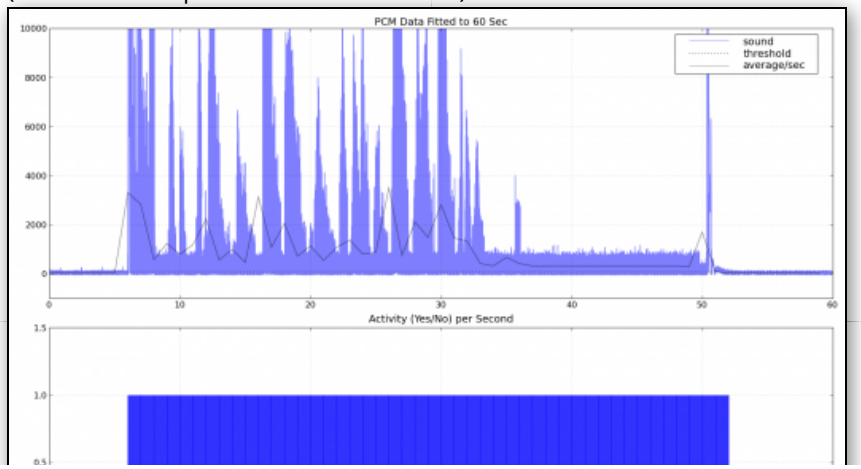
This will produce a graph that looks like this:



Could it have been ANY easier? I'm so in love with python I could cry right now. With the powerful tools Numpy provides to rapidly and efficiently analyze large arrays (PCM potential values) combined with the easy-to-use graphing tools Matplotlib provides, I'd say you can get well on your way to analyzing PCM audio for your project in no time. Good luck!

FOR MORE INFORMATION AND CODE check out the highly-relevant posts:
Linear Data Smoothing In Python
Signal Filtering With Python
Circuits Vs. Software
and pretty much anything in the DIY ECG category of entries.

Let's get fancy and use this concept to determine the number of seconds in a 1-minute PCM file in which a radio transmission occurs. I was given a 1-minute PCM file with a ~45 second transmission in the middle. Here's the graph of the result of the code posted below it. (Detailed descriptions are at the bottom)



Categories

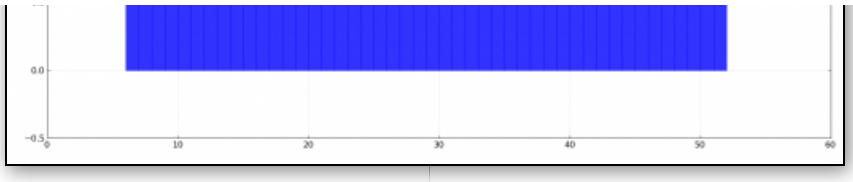


Figure description: The top trace (light blue) is the absolute value of the raw sound trace from the PCM file. The solid black line is the average (per second) of the raw audio trace. The horizontal dotted line represents the *threshold*, a value I selected. If the average volume for a second is above the threshold, that second is considered as "transmission" (1), if it's below the threshold it's "silent" (0). By graphing these 60 values in bar graph form (bottom window) we get a good idea of when the transmission starts and ends. Note that the ENTIRE graphing steps are for demonstration purposes only, and all the math can be done in the 1st half of the code. Graphing may be useful when determining the optimal threshold though. Even when the radio is silent, the microphone is a little noisy. The optimal threshold is one which would consider microphone noise as silent, but consider a silent radio transmission as a transmission.

```
### THIS CODE DETERMINES THE NUMBER OF SECONDS OF TRANSMISSION
### FROM A 60 SECOND PCM FILE (MAKE SURE PCM IS 60 SEC LONG!)
import numpy
threshold=80 # set this to suit your audio levels
dataY=numpy.memmap("test.pcm", dtype='h', mode='r') #read PCM
dataY=dataY-numpy.average(dataY) #adjust the sound vertically the avg is at 0
dataY=numpy.absolute(dataY) #no negative values
valsPerSec=float(len(dataY)/60) #assume audio is 60 seconds long
dataX=numpy.arange(len(dataY))/(valsPerSec) #time axis from 0 to 60
secY,secX,secA=[],[],[]
for sec in xrange(60):
   secData=dataY[valsPerSec*sec:valsPerSec*(sec+1)]
   val=numpy.average(secData)
   secY.append(val)
   secX.append(sec)
   if val>threshold: secA.append(1)
   else: secA.append(0)
print "%d sec of 60 used = %0.02f"%(sum(secA),sum(secA)/60.0)
raw_input("press ENTER to graph this junk...")

### CODE FROM HERE IS ONLY USED TO GRAPH THE DATA
### IT MAY BE USEFUL FOR DETERMINING OPTIMAL THRESHOLD
import pylab
ax=pylab.subplot(211)
pylab.title("PCM Data Fitted to 60 Sec")
pylab.plot(dataX,dataY,'b',alpha=.5,label="sound")
pylab.axhline(threshold,color='k',ls=":",label="threshold")
pylab.plot(secX,secY,'k',label="average/sec",alpha=.5)
pylab.legend()
pylab.grid(alpha=.2)
pylab.axis([None,None,-1000,10000])
pylab.subplot(212,sharex=ax)
pylab.title("Activity (Yes/No) per Second")
```

(manned experimental propagation transmistter)

RF (Radio Frequency)

Thermoregulation UCF Lab

Uncategorized

Categories

C/C++ (46)

Circuitry (103)

DIY ECG (16)

Electronics (15)

General (196)

GitHub (16)

HAB (high altitude balloon) (4)

Linux (15)

Microcontrollers (77)

Molecular Biology (3)

My Website (10)

PHP (4)

Prime Numbers (16)

Programming (4)

Python (69)

QRSS / MEPT (manned experimental propagation transmistter) (4)

RF (Radio Frequency) (84)

Thermoregulation (3)

UCF Lab (4)

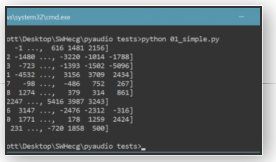
Uncategorized (2)

Archives

```
pylab.grid(alpha=.2)
pylab.bar(secX,secA,width=1,linewidth=0,alpha=.8)
pylab.axis([None,None,-0.5,1.5])
pylab.show()
```

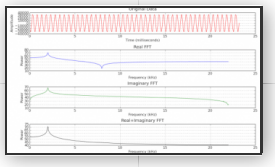
The output of this code:`46 sec of 60 used = 0.77`

Share this:



[Realtime Audio Visualization in Python](#)
July 19, 2016
In "DIY ECG"

[Fixing Slow Matplotlib in Python(x,y)](#)
I recently migrated to Python(x,y) and noticed my matplotlib graphs are resizing unacceptably slowly
April 15, 2013
In "General"



[Insights Into FFTs, Imaginary Numbers, and Accurate Spectrographs](#)
June 23, 2010
In "General"

Python | Leave a comment

‹ pySquelch – Frequency Activity Reports via Python

› Jedi Soldering Skills

January 2010 (2)
December 2009 (2)
September 2009 (1)
August 2009 (7)
July 2009 (4)
June 2009 (18)
May 2009 (11)
April 2009 (10)
March 2009 (2)
February 2009 (4)
January 2009 (18)
November 2008 (5)

You must log in to post a comment.

About Scott

Meta

Log in
Entries RSS
Comments RSS
WordPress.org

Scott Harden lives in Gainesville, Florida and works at the University of Florida as a biological research scientist studying cellular neurophysiology. Scott has lifelong passion for computer programming and electrical engineering, and in his spare time enjoys building small electrical devices and writing cross-platform open-source software. more →

Contact

SWHarden@gmail.com

Subscribe via Email

Enter your email address to subscribe to this website and receive notifications of new posts by email.

Email Address

Subscribe