Using Torch in Exploratory Signal Processing

Dr. Krista M Hill, University of Hartford

Dr. Krista M. Hill is an associate professor in Electrical and Computer Engineering at the University of Hartford in Connecticut. PhD and MSEE from Worcester Polytechnic Inst. in Worcester, Mass., and previously a project engineer at Digital Equipment Corp. She instructs graduate and undergraduate computer engineering computer courses, directs undergraduate and graduate research. Her current projects involve small system design, signal processing, and intelligent instrumentation.

Using Torch in Exploratory Signal Processing

Abstract

This research considers Torch along with two other packages that student researchers could use to assess the content in audio recordings. Our prior efforts have involved biomedical engineering undergraduate senior projects in our college. This research has application in our bowel-sounds project, which assesses abdominal sounds in newborn babies as signs of digestion. The research can potentially be used to estimate statistics, such as when a baby is crying or snoring.

Our focus has shifted, from making recordings, which are large in file size and may be several hours to two days in length, to considering the contents of the recordings. Our recorder automatically splits a session across multiple files, as a single file would likely exceed the FAT32 file system limit. Initially we used the well-known Audacity sound editor to manually select segments of a given recording for analysis.

The goal of this specific research is to consider and compare three packages, namely Torch, MATLAB, and Octave, then pick one for our next phase in research. This paper presents two examples, comparing execution times for various data set sizes. In many institutions MATLAB as well as Octave which is similar, are standard tools. I considered Torch as an alternative as it uses Lua, which is a powerful, fast, lightweight, and dynamically typed scripting language. Also, in reviewing the listing of Torch packages, there are resources for handling audio files, natural language processing, visualization, and machine learning.

The examples in this paper are straightforward and can be understood by students having an introduction to signal processing. Students in our electrical engineering program as well as biomedical engineering with electrical engineering concentration will have some experience with MATLAB scripting. They should also be able to use prewritten Torch scripts.

My role is as an instructor, planning for and directing our student work. In our next step, we will use one package to develop tools for student researchers, to first assist and eventually automate the analysis of such sound files. Given the investigative nature of our overall research, I avoided compiled languages such as C, C++, or Java, as developer convenience is more important than execution performance. Due to time limitations I was not able to consider Scilab, Scientific Python, SageMath, or a number of others.

In reviewing this research I am convinced that Torch has potential for use as a tool at the undergraduate and graduate levels. I also found that MATLAB is more robust than I suspected. We see that in some cases the performance of Torch is comparable to and in other cases exceeds that of MATLAB and Octave. Both Torch and MATLAB each have resources for handling audio and machine learning.

Introduction

The purpose of this research is to assess the performance of the Torch package against MATLAB as well as Octave, which are well known packages, for our purpose of processing audio files.

Torch is described[2] as a scientific computing framework with wide support for machine learning algorithms and other scientific areas. Torch uses Lua, which is a powerful, fast, lightweight, and dynamically typed scripting language. I briefly considered GSL-Shell[3], which provides an interactive environment to the GNU Scientific Library (GSL). Both Torch and GSL-Shell make use of a just-in-time compiler, for enhanced performance. After considering both packages, I settled on Torch for this research.

In the following we consider two examples, the source code is included in an appendix. First, a Finite Impulse Response Filter (FIR) using a moderately long impulse response. The second example reduces a file by calculating the sample-RMS value over periodic intervals. Such processing is useful for identifying in hours-long bowel-sound recordings, regions of interest. The examples are straightforward and can be understood by students having an introduction to signal processing.

Selected Platform and Software

The platform I selected is a DELL Optiplex 980 model 64-bit desktop PC with 8 Gbytes of RAM. The system is not a performance workstation. I chose Ubuntu version 16.04, which is a Linux operating system as it supports both Torch and MATLAB. With regard to Torch, its development has focused on Linux. With regard to MATLAB, recent versions for Linux only support 64-bit systems. Octave is available for a number of systems including 64-bit Linux.

Torch is an active open-source project. The project website[2] lists several supporting groups including research labs and companies. The files associated with Torch are publicly available from GitHub[4]. I found that Torch builds easily on the given Ubuntu system.

My first two impressions of Torch were, "*Oh my, this is big!*" and "*What's a tensor?*" After some digging I learned that matrices and vectors are actually special cases of tensors. To get started, I found Ata's[5] cross-listing, from MATLAB to Torch, as well as Collobert, Kavukcuolgu, and Farabet's[6] outline of their use of Torch, to be helpful.

The scripting language for Octave is very close to that of MATLAB. The scripts here for Octave are simple and except for one case here, execute by MATLAB with slight modification. I found that MATLAB uses the audioread command rather than wavread to read a wav-format file. For more sophisticated cases, with modest effort, writing such compatible scripts is possible.

Graphics

There are numerous supporting packages available for Torch for visualizing data. Strub[7] provides an overview of five such packages and refers to two additional resources. I used the Torch interface to gnuplot as it is similar to that of MATLAB.

In plotting lines with the gnuplot package, the format string allows for options other than the color and type of marker. The tilde (~) invokes smoothing using cubic interpolation while the hyphen(-) causes line plotting without markers. The lower case v is for drawing vector fields. A custom string can be passed to enable the full capability of gnuplot.

Performing Tests
The following is an outline of how timing measurements were performed. To avoid any affect of the desktop state, each application was executed remotely using the secure shell (ssh) program.

- Given a number of samples to process, execute a given script one or more times. If the computer was dormant, this appears to allow the computer to settle in its performance.
- For reasonably small measured time values (a few minutes or less), repeat the test eight times, remove the largest and smallest values, calculate the average of the remaining six remaining values.
- For measured times that exceed twenty minutes, calculate the average of four times.

FIR Filter Example
This example considers a Finite Impulse Response (FIR) filter. The notion of the well-known FIR filter is summarized by the block diagram in Figure 1, and maps neatly to linear algebra. Here, impulse response values are in an array *h*. With element numbering starting at one, the first impulse response value is in *h*(1). Each trapezoid weighs a sample value in *v* by a corresponding value in *h*.

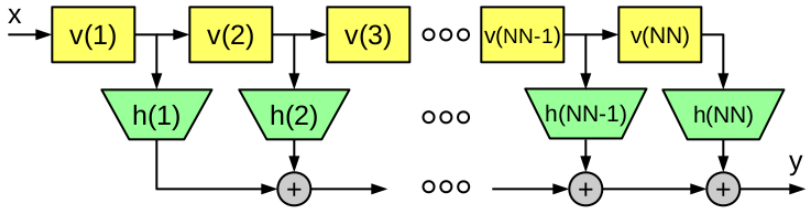

Figure 1: Block diagram of FIR filter

At each sample period the values in *v* are shifted to the right and the most recently received value *x* is stored into *v*(1). The output *y* is the dot-product of *v* and *h*. In teaching discrete time signal processing, this a simple way to introduce convolution. Consider that with *x* being a unit impulse, the output *y* is the impulse response. In applying linearity, the output is the sum of scaled and delayed impulse responses.

In this example we consider an impulse response that is an exponentially decaying sinusoid. Such an impulse response corresponds to a resonance and behaves as a band pass filter. The purpose of the example is to be fairly computationally intensive. The following are the details:

- Sinusoid frequency 1kHz
- Time constant for decay (tau): 10ms
- Sampling frequency: 96000 Hz
- Length of impulse response (NN): Four time constants or 3840 samples

A Gaussian random number generator provides input samples. For each run, a given number (NS) of samples is collected. Here we are interested in the execution time for several given values of NS as well as the time to produce the Power Spectral Density (PSD) in the output *y* by producing the magnitude in decibels by using the Fast-Fourier Transform (FFT).

A script was written for Octave and MATLAB to implement the filter system. Figure 2 shows example output from Octave for 24000 samples, or 0.25 seconds of simulated time. The output looks similar to an AM modulated carrier where the envelope is somewhat random. Figure 3 shows the corresponding PSD of *y,* produced from the FFT of the time values. The display is zoomed to show 0 Hz to 8 kHz. As expected, the peak in the PSD appears at 1kHz.

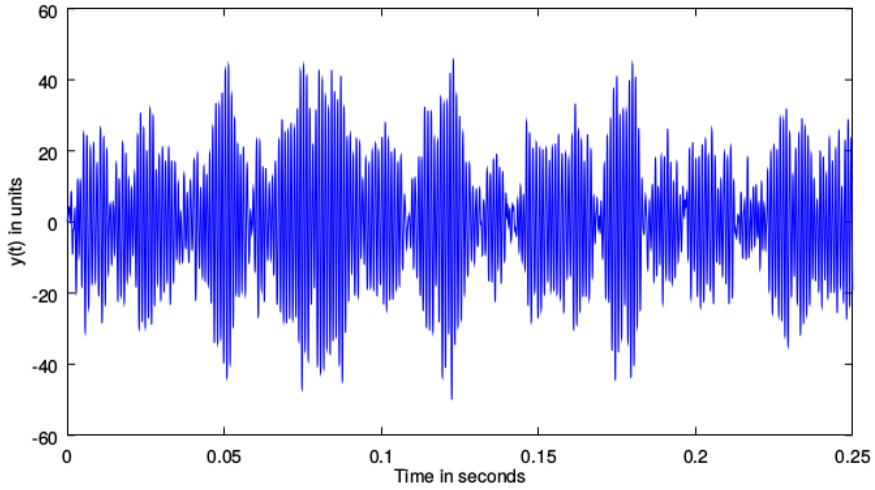

Figure 2: Output y(t) from filter (Octave)

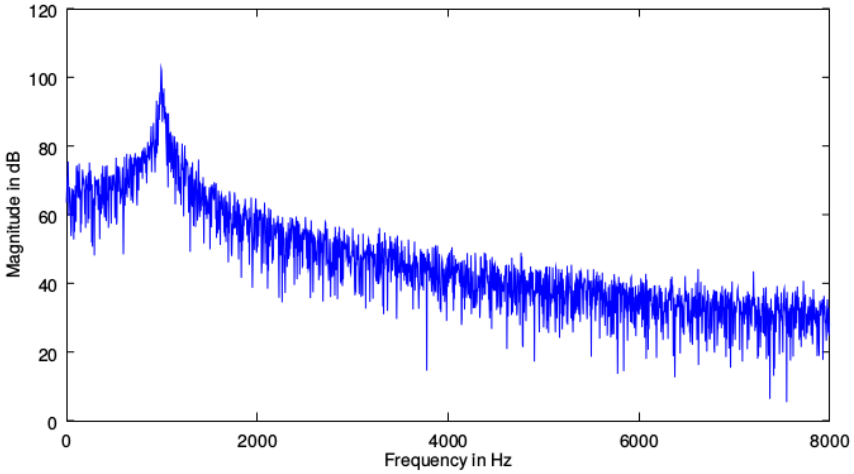

Figure 3: Power spectral density of y, view from 0 Hz to 20 Hx (Octave)

To compare the performance, a similar script was written for Torch. Each package was tested in turn. The scripts are in an appendix. The measured time values for the filter as well as that for the PSD are summarized in Table 1. The given values for NS correspond to 0.25sec, 1.042sec, and 10.42 sec of simulated time. For each NS value, T1 is the time to produce the output *y* and T2 is the time to calculate the PSD by producing the magnitude of the FFT, scaled to decibels. The time to produce the display was not considered.

Table 1: Summary of example measured time values for filter and PSD

Samples	Measure	MATLAB	Torch	Octave
NS.a = $24 \cdot 10^3$	T1.a	0.4330s	0.3493s	1.0580s
(0.25 sec)	T2.a	1.128ms	2.834ms	3.331ms
NS.b = $100 \cdot 10^3$	T1.b	1.8064s	1.8057s	4.3431s
(1.042 sec)	T2.b	3.593ms	12.095ms	9.880ms
NS.c = $1 \cdot 10^6$	T1.c	17.2594s	17.8232s	43.6896s
(10.42 sec)	T2.c	36.825ms	131.939ms	73.524ms

Table 2 compares the execution times of MATLAB as well as Octave to Torch.

- The ratio of T1 execution times for MATLAB to Torch are close to unity, suggesting similar performance.
- The ratio T1 values for Octave to Torch is approximately 2.4. This suggests better performance for Torch than Octave for T1.
- The ratio of T2 For MATLAB to Torch ranges from almost 0.4 to 0.3, so that for T2 MATLAB is from 2.5 to 3 times faster than Torch in producing a PSD.
- The ratio of T2 For Octave to Torch ranges from almost 1.2 to almost 0.56 so that in producing a PSD, Torch is slightly faster than Octave for smaller NS but slower for larger NS values.

Table 2: Comparison of times for FIR example

		MATLAB/Torch	Octave/Torch
NS.a = $24 \cdot 10^3$	Ratio T1.a	0.9637	2.3546
	Ratio T2.a	0.3978	1.1753
NS.b = $100 \cdot 10^3$	Ratio T1.b	1.0003	2.4052
	Ratio T2.b	0.2971	0.8168
NS.c = $1 \cdot 10^6$	Ratio T1.c	0.9684	2.4513
	Ratio T2.c	0.2971	0.5573

Performing a dot-product is a fundamental linear algebra operation. For this we note that MATLAB and Torch have similar performance. In producing a PSD, we see that MATLAB had the best overall performance.

Where to Look for Bowel-Sounds

We note that a hospital is a noisy place. Such a recording will have extraneous noises such as the sound of the baby wiggling, medical staff talking, push-carts squealing, adjacent patients crying, and the like. Given such interference, we suspect that it is unlikely that continuous assessment of bowel sounds is possible.

For now it is useful to identify segments in a recording that have less extraneous sounds. In his seminal research, Cannon[8] describes bowel sounds as varying and being impulsive [p.352], "Most prominent among these irregular sounds are the sudden quick discharges or pops, which can be heard, either singly or in a short series… Occasionally a continuous little gurgling can be heard for some moments… "

To help identify regions of interest in a recording in a way that reduces the file size, it can help to form the sample-RMS value over regular time intervals, as in eq.1. Each RMS value is produced essentially by first forming the sum of squares in a sample window with NSW samples, or *Wsec* seconds long. The sum-of-squares is formed by applying the vector dot-product to the sample window values. Next, divide by the number of window samples and form the square root. The resulting file is reduced by a factor of NSW.

$$r(n) = \left(\frac{1}{\mathrm{NSW}} \sum_{k = n \cdot \mathrm{NSW}}^{(n+1)\cdot \mathrm{NSW}-1} v(k)^2 \right)^{\frac{1}{2}} \qquad \text{eq.1}$$

To illustrate, Figure 4 shows a short recording displayed by the Audacity editor. The sample rate here is 44.1kHz. The following are the numbered features:

(1) Speaker's lips open, preparing to say something
(2) Speaker says, "*hello*"
(3) Series of three hand claps
(4) Speaker says, "*good-day*". The dip in amplitude differentiates the words.

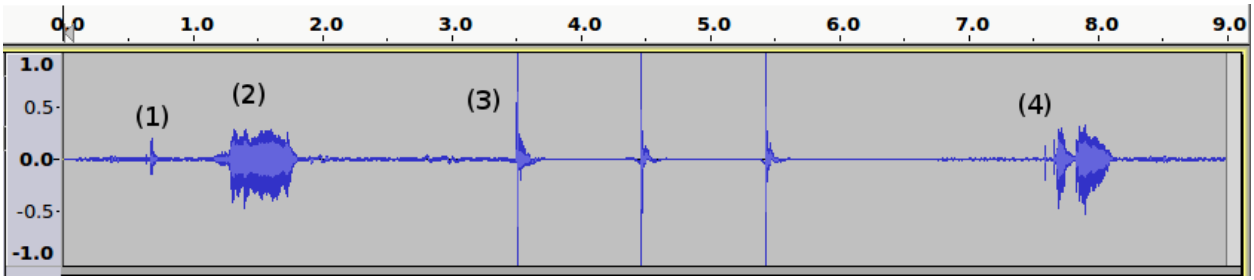

Figure 4: Recording to demonstrate RMS reduction

To reduce this file, a sample window of 100ms was chosen, which corresponds to NSW being 4410 samples, this is comparable to the time for a hand clap. In selecting a sample window, finite energy signals having a similar or smaller time scale are de-emphasized. In Figure 5 event (1) is hardly noticeable, the amplitude of the hand-claps in (3) are reduced to be close to that of the

speech in (2). Likewise in event (4) the differentiation between words is seen by a slight dip in the waveform.

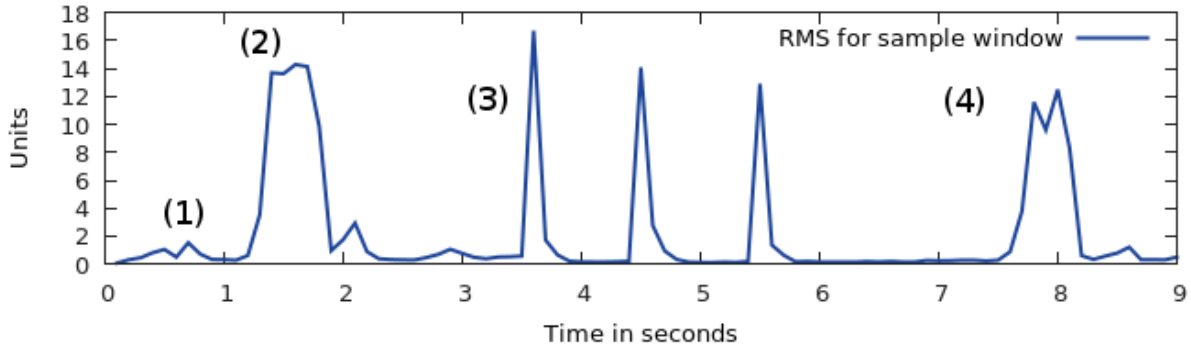

Figure 5: Recording reduced by RMS with 100ms window (Torch and gnuplot)

The point is to select a sample window size NSW to significantly de-emphasize impulsive signals such as bowel-sounds. With the remaining representing interference signals, the best locations to assess bowel-sounds is where the reduced waveform level is low.

Table 3 summarizes some times for each package to perform the above algorithm, when the packages were run remotely with ssh. The sample rate is 48 kHz and *Wsec* was arbitrarily ten seconds. Table 4 Provides a summary comparison of the times compared to that of Torch. The execution time for MATLAB seen here is approximately 5.3 times that of Torch. The execution time for Octave varies with respect to the number of samples and here is at least two orders of magnitude larger than Torch.

Table 3: Summary of measured time for reduction example

Recording time (hours)	MATLAB (sec)	Torch (sec)	Octave (sec)
1 Hr	4.884	0.918	199.341
2 Hr	9.474	1.777	798.520
3 Hr	14.272	2.663	2372.35

Table 4: Comparison of times for packages to Torch for reduction example

Recording time (hours)	MATLAB/Torch	Octave/Torch
1 Hr	5.320	217.134
2 Hr	5.332	449.448
3 Hr	5.359	890.975

In reviewing the second example, it appears that Torch has the most optimization for processing large audio recordings. Perhaps the Torch packages for natural language processing and machine

learning are driving factors. The performance of MATLAB also appears to be optimized. The performance of Octave suggests that it is not optimized for processing such data.

Summary

In reviewing this research we find that Torch has potential for use as a tool in performing such student research at the undergraduate and graduate level. I also found that MATLAB is more robust than I suspected. We see that in some cases the performance of Torch is comparable and in other cases exceeds that of MATLAB and Octave.

References

1. MATLAB, a product of MathWorks, http://www.mathworks.com/
2. Torch Project, “Torch | Scientific computing for LuaJIT,” retrieved Feb. 4, 2017 ; http://torch.ch/
3. Francesco Abbate, “The GSL Shell Project,” copyright 2012, retrieved Feb. 9, 2017; http://www.nongnu.org/gsl-shell/
4. GitHub repository, reference retrieved Feb. 4, 2017; https://github.com/
5. Ata Mahjoubfar, “Torch for Matlab(R) Users,” ata.m@ucla.edu, April 30, 2015, retrieved Feb. 3, 2017 ; http://atamahjoubfar.github.io/Torch_for_Matlab_users.pdf
6. Ronan Collobert, et al., “Torch7:A Matlab-like Environment for Machine Learning,” retrieved Feb. 4, 2017 ; cs.nyu.edu/~koray/files/2011_torch7_nipsw.pdf
7. Florian Strub, “Plotting with Torch7,” retrieved Feb. 12, 2017 ; http://www.lighting-torch.com/2015/08/24/plotting-with-torch7/
8. W. B. Cannon, “*Auscultation of the Rhythmic Sounds Produced by the Stomach and Intestines*,” The American Journal of Physiology, Volume XIV, No. IV, Oct. 2, 1905, pp. 339 – 353.

Appendix – Source Code for Examples

Code is for Octave and Torch. For MATLAB use Octave with audioread rather than wavread.

First example code for Octave:

```
% ring_wave_01.m - Krista Hill -01/31/2017
% For Octave - Apply resonance filter
Fs = 96000
Ts = 1/Fs
Fx = 1e3
Wx = 2*pi*Fx*Ts

% Make impulse response
Tau = 10e-3
NN = round(Tau/Ts)*4
time = Ts * (0:NN-1)';
hx = zeros(NN,1);
theta = 0;

for ix = 1:NN
  hx(ix) = sin(theta) * exp(-ix*Ts/Tau);
  theta = theta + Wx;
  if theta > 2 * pi
    theta = theta - 2*pi;
  end
end

% Prepare for filter
NS = 24000;
xvec = zeros(1,NN);
yvec = zeros(NS,1);
tvec = (0:NS-1)' * Ts;

% Produce the y output
tic
for ix = 1:NS
  yvec(ix) = xvec * hx;
  xi = randn();
  xvec = [xi xvec(1:NN-1)];
  %xvec = [0 xvec(1:NN-1)];
end
'Time 1:'
toc
```

```
% Plot output
figure(1)
plot(tvec,yvec)
xlabel('Time in seconds')
ylabel('y(t) in units')

% Produce the PSD
tic
fy  = fft(yvec);
fys = 20*log10( abs(fy) );
'Time 2:'
toc

% Plot the PSD
figure(2)
fvec = (Fs/NS) * (0:NS-1)';
plot(fvec(1:5000),fys(1:5000));
xlabel('Frequency in Hz')
ylabel('Magnitude in dB')
```

First example code for Torch:

```
-- ring_wave_08.lua - Krista Hill
-- 01/31/2017 Example written for Torch
require('gnuplot');
sig = require('signal');
Fs = 96000
Ts = 1/Fs
Fx = 1e3
Wx = 2*math.pi*Fx*Ts
print("Wx: "..Wx)
Tau = 10e-3
NN = math.floor(Tau/Ts +0.5)*4

-- Structures for FIR filter
vv = torch.zeros(NN)
hv = torch.zeros(NN)
for ix = 1,NN do
  hv[ix] = math.sin(Wx * (ix-1)) *
             math.exp(-(ix-1)*Ts/Tau)
end
bv = vv[{{1,NN-1}}]

-- Prepare for filter
NS = 24000
yv = torch.zeros(NS)
tv = torch.linspace(0,NS-1,NS)*Ts

-- Produce the y output
timer = torch.Timer()
timer:reset()
for iy = 1,NS do
  yv[iy] = vv * hv
  vv[{{2,NN}}] = bv:clone()
  vv[1] = torch.randn(1,1)
end
print('Time 1: ' .. timer:time().real ..
      ' sec')
print('NS: ',NS)

-- Plot the output
gnuplot.figure(1)
gnuplot.plot({'hwave',tv,yv,'~'})
fv = torch.linspace(0,NS-1,NS)*Fs/NS

-- Produce the PSD
timer:reset()
psd = torch.log( sig.complex.abs(
         sig.fft(yv)  ) )/math.log(10)
print('Time 2: ' .. timer:time().real ..
      ' sec')

-- Plot the PSD
gnuplot.figure(2)
gnuplot.plot('PSD',fv,psd,'-')
```

Second example code for Octave:

```
% fm_wrms_02.m - Krista Hill - Feb.2, 2017
% For Octave, periodically make RMS
FileName = 'rec_hello_02.wav';

[NumSamples,NumChannels] =
               wavread(FileName,'size')
[yx,fs,bd] = wavread(FileName,[1])
Wsec = 0.1
NSW  = fs * Wsec
range  = [1,NSW]

NSX = floor(NumSamples/NSW)
rvec = zeros(NSX,1);
tic
for ix = 1:NSX
  buf = wavread(FileName,range)(:,2);
  rvec(ix) = sqrt(buf' * buf);
  range = range + NSW;
end
disp('Time 1:')
toc

tvec = Wsec * (1:NSX)';
plot(tvec,rvec)
```

Second example code for Torch:

```
-- th_wrms_02.lua - Krista Hill
-- Feb.2, 2017 Example for Torch
require('sndfile')
require('gnuplot')
FileName = 'DR0000_0024.wav'
sf = sndfile.SndFile(FileName)
sx = sf:info()

-- Parameters
Wsec = 10
NSW  = sx.samplerate * Wsec
NSX  = math.floor(sx.frames/NSW)
rvec = torch.zeros(NSX);
xvec =
torch.linspace(Wsec,Wsec*NSX,NSX);

-- Start a timer
timer = torch.Timer()
timer:reset()

-- Loop through reading the file
for ix = 1,NSX do
  buf  = sf:readDouble(NSW)
  bufc = buf[{{},{2}}]
  bufr = bufc:t()
  rvec[ix] = torch.sqrt( bufr * bufc )
end

-- Produce timing value
print('Time 1: ' ..
timer:time().real .. ' sec')
print('Wsec: ',Wsec)
print('NSX: ',NSX)

-- Plot the resulting waveform
gnuplot.figure(1)
gnuplot.xlabel('Time in seconds')
gnuplot.ylabel('Units')
gnuplot.plot('RMS for sample
window',xvec,rvec,'-')
sf:close()
```