"Radio Rex": Frequency analysis practical
CSE 5525 / Eric Fosler-Lussier

The goal of this practical is to get you familiarized with the basic components of frequency analysis. For some of you (ECE students particularly) this will be old hat; please help your fellow students understand this better.

There are two practical objectives today: 1) write your own spectrogram routines, and 2) build your own virtual "Radio Rex".

We will be using octave, a free and open source MATLAB clone, as our language. To start octave, open up a terminal and type "octave". You may also choose to run MATLAB natively on your laptop, but this won't work in the VM.

You can find the data for the class in the directory rexdata. The wavefiles all contain the word "Rex" (although some have extra stuff in them as well). The files are sampled at 16000 Hz.

You may find it easier (as I did) to open up a wavefile in Audacity with the spectrogram analysis to see if you can replicate the results. Also, you can open a file "foo.m" in a text editor in the directory you're working in and put your commands in there. That way you can just type "foo" in matlab/octave and it will read your command list – easier for doing modifications.

Part 1: Building your own spectrogram

First, read in one of the files and plot the waveform and listen to it (if you use rex1.wav, then the stuff below makes more sense):

```
>> [x,fs,nbits]=wavread('rex1.wav');
>> plot(x)
>> sound(x,fs)
```

You'll notice that much of the file is silence. Now, we can get a feeling for the frequencies present in the entire file by taking its FFT (NOTE: we're about to violate the stationarity assumption we talked about in class – the energy patterns are moving as a function of time)

```
>> length(x)
ans =
       51456
>> X=fft(x);
>> length(X)
ans =
       51456

>> plot(real(X))
```

When you take an FFT, remember that you get out a complex number (real corresponding to cosines, imaginary corresponding to sines). So one way to get a feel for the data is to just plot the real portion. You'll notice that there are the same number of

points in the resulting FFT as in the original signal. The way to read the FFT is to take the sampling frequency and spread it evenly across the points in the FFT. In our case, the sampling frequency was 16000, so each point is 16000/51456=0.3109 Hz more than the previous point in the sequence.

Do you notice the aliasing? (Hint: what's being reflected?)

Well, if we wanted to build a Radio Rex, we wanted to find energy around 500 Hz. So: let's focus the plot in that area and see what we find. 500 Hz will be around 500*51456/16000 = 1608, so let's focus the plot on the area between 1000 and 2000:

```
axis([1000 2000 -200 150])
```

It looks like there are peaks around 1400 = 435 Hz and 1850 = 575 Hz. So it looks like there's some energy in the right place – off to a good start.

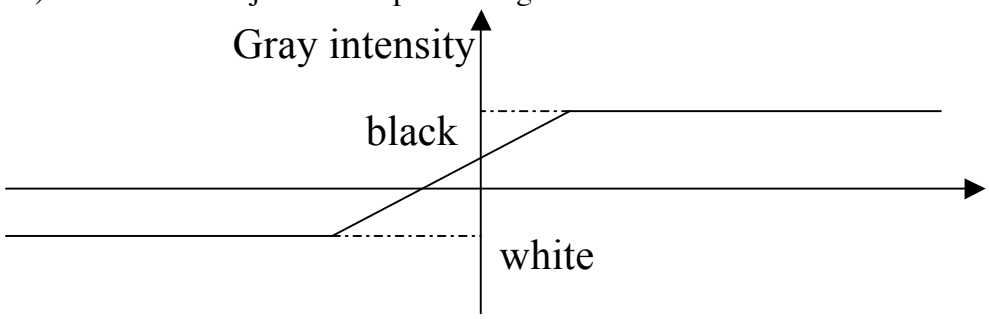
Now, to get a spectrogram, remember that we are going to do a few other things:

1) We want to estimate frequencies on a shorter timescale, so that we get better resolution and can see how frequencies evolve over time
2) To do (1) we need to window (and we might need different types of windowing)
3) We need to take the log spectrum accounting for both the real and imaginary parts

$$\log |X_m[k]|^2 = \log(\mathrm{Re}\{X_m[k]\}^2 + \mathrm{Im}\{X_m[k]\}^2)$$

$$X_m[k] = X_m(e^{j\omega}) \Big|_{\omega=\frac{2\pi k}{M}}$$

4) We need to adjust the output scaling so that it saturates



Perversely, let's go in the order (3), (1), (2), (4). If you get stuck, I've provided an answer for you in spect.m, but of course you won't peek!

For (3), you'll need to know the following functions:

- real(x): gives the real part of a complex number (works on vectors/matrices)
- imag(x): gives the imaginary part (as a real number) of a complex number
- x.^2: pointwise square of a vector/matrix x
- log(x): gives the natural log of a number (works on vectors/matrices)

Can you figure out the expression that will relate X to its log spectrum L?

(answer on the next page – don't peek until you've tried to figure it out)

Here's the code:

```
>> L=log(real(X).^2+imag(X).^2);
>> plot(L)
```

You can also use the axis command to focus in on the 500 Hz region as before.

Now let's see if we can get small windowed estimates of the spectrum (1). The idea is to take only "winsize" samples at a time and compute an FFT on that. Wavesurfer uses a maximum of 256 samples (I think the default is 64) NOTE: when they say 256 points, they may actually mean 512 points because of the aliasing – only the first 256 points are useful. Let's try taking a 256-point FFT around the center of the vowel.

```
>> start=31000;
>> winsize=256;
>> xwindowed=x(start:start+winsize-1);
>> plot(xwindowed)
>> Xwindowed=fft(xwindowed,winsize);
>> Lwindowed=log(real(Xwindowed).^2+imag(Xwindowed).^2);
>> plot(Lwindowed)
```

Notice that the FFT is also 256 points – meaning that the frequency resolution of this is much less (256 points covering 16000 Hz = 62.5 Hz/point).

Great, we now have a local estimate of the spectrum, but we will want to accumulate that over multiple windows. So we will want to shift this window (meaning for different values of start, maybe skipping by 20 samples or so) and build up a matrix out of the column vectors that we get. Just so that you can work it out for yourself, I'll show you the answer on the next page. But here's a quick example of how to fill in a matrix as I've described. This example erases the matrix M, takes the column vector 1:4, multiplies it by i, and then inserts it as column i in the matrix.

```
>> clear M
>> for i=1:5
M(:,i)=(1:4)'*i;
end
>> M

M =

     1     2     3     4     5
     2     4     6     8    10
     3     6     9    12    15
     4     8    12    16    20
```

Here's my answer:

```
winsize=256;
shift=20;
c=1;
clear L;
for i=1:shift:length(x)-winsize
     X=fft(x(i:i+winsize-1),winsize);
     L(:,c)=log(real(X).^2+imag(X).^2);
     c=c+1;
end
```

Now we can look at this:

```
>> pcolor(L); shading('flat')
```

Get rid of the annoying aliasing

```
>> axis([1,size(L,2),1,128])
```

Focus in on the interesting part

```
>> axis([1500 2000 1 128])
```

Well, getting closer. You can also try out

```
>> surf(L); shading('flat')
```

to get a 3-d plot. Also, try different colormaps:

```
>> colormap(gray)
>> colormap(1-gray)
>> colormap(jet)
```

Halfway there. Now, let's try putting in non-rectangular windowing (2). Luckily, the Hamming window is built-in to the matlab on stdsun as "hamming" (your matlab may or may not have this – if it doesn't you can copy the numbers from someone).

Where would you put windowing in? (Answer below.)

```
winsize=256;
shift=20;
c=1;
clear L;
h=hamming(256);

for i=1:shift:length(x)-winsize
     X=fft(x(i:i+winsize-1).*h,winsize);
     L(:,c)=log(real(X).^2+imag(X).^2);
     c=c+1;
end
```

This clears up the picture a little bit (do a pcolor plot and then focus in again on the interesting stuff – you may want to do "figure(2)" to open a new figure window to compare. In particular, it gets rid of much of the vertical line artifacts above the vowel.

Finally, we want to put in some thresholding. Here's a fact to know: MATLAB only plots 64 different colors as part of a colormap. So there's really only 64 quantities to

worry about. If we map stuff on the ends to either 1 or 64, and linearly scale the rest, you can adjust the contrast the same way that you do in audacity.

This one's a bit tricky and takes some knowing of matlab to do efficiently, so I'll give you this one. Comments start with %

```
% do rescaling (4)
% find the floor of L, reset to be 0
mn=min(min(L));
L=L-mn;

% find the max of L.  Map L to a number between 0 and 128 (at first)
% and then subtract 50 (-50 to 78)
mx=max(max(L));
L=floor(L/mx*128)-50;

% map all numbers below 1 to 1, all numbers above 64 to 64
L(find(L<1))=1;
L(find(L>64))=64;
```

Try out different scalings to see what happens. When you get done, try loading some of the other wave files and displaying their spectrogram.

Part 2: Building a "Radio Rex"

Well, if you've gotten this far you're probably doing pretty well. There are fewer instructions on this part – I want you to think about the problem some more.

Radio Rex operated by doing a band-pass filter around 500 Hz and then threshholding the energy.

The idea that I want you to pursue is to try to identify when there's significant enough energy around 500 Hz that Radio Rex should fire. We don't know what the bandwidth should be (490-510? 400-600? 300-700?). We don't know what the threshold should be. So I want you to examine the data, and come up with some rules by hand.

Take a weighted sum of the log spectral energy (i.e. what you calculated in the spectrogram) at every time step; the weights will depend on your bandwidth and the shape of your filter (i.e. triangular sloping to the edges?). For that filter shape, figure out a threshold that will separate out the /eh/ from the rest of the data.

rex1.wav through rex5.wav are from the same speaker (me). rex6.wav through rex10.wav are from other speakers. See how robust your rule is.