This is the assignment sheet I made for a class describing the use of lpc. Everything in this directory should work as described here. You'll ust have to adjust some path names.

# Music/Computer Science 325
**Assignment #7**

## Linear Predictive Coding

This week the assignment is to create an lpc data set of your own voice, and do various synthesis experiments with it.

1) Record a short sentence or fragment using the digital microphone. I recommend that you keep it under 5 seconds. You can do this using the built-in microphone on the NeXTStations or the Digital Microphone in room 212 in Woolworth. The Digital Microphone will give you better quality speech, but the NeXT microphone will also produce acceptable results for this exercise. For the best results record something quietly on a GOOD quality cassette and cassette machine and transfer it via the Digital Ears in the NeXT Cluster or room 212.

If you use the Digital Ears or Microphone:

**sndrecord -d *yourfile.snd***
and then convert the file to a 22k MONO file using sndconvert, in two steps
**sndconvert -s 22050 -o *newfile.snd yourfile.snd***
**sndconvert -c 1 -o *final.snd newfile.snd***

If you use the built-in microphone

**sndrecord *yourfile.snd***
**muto16 *yourfile.snd newfile.snd***
**sndconvert -c 1 -o *final.snd newfile.snd***

You should now have a 22k, mono, 16-bit integer file of voice, ready for processing. (You might want to create a cute phone message.) Well modulated female voices are generally easier to deal with than grungy male voices.

2) Edit the soundfile with **edsnd** or **soundworks** and delete silence at the beginning or end of the file.

3) Create an lpc data set.
To do this you say
**lpc** -o yourfile.lpc -p 24 -f 250 yourfile.snd

This will create a 24 pole analysis of your voice with frames of 250 samples. The framerate is thus 22050/250 = 88.2 frames per second. I recommend these parameters for most speech at 22khz.

4) Now you have to stabilize this data set. Just say

**stabilize** yourfile.lpc 24

(Make sure you have two arguments here, the name of the data set and the number of poles it contains.)

This will grind for a few minutes. Don't worry about the "bad read" statement at the end..

5) Now you have to do a pitch analysis of your data set. Listen to it and see if you can get a rough estimate of the pitch range. I have installed a program called **hz** which will beep a specific frequency back at your for 1 second. Just say *hz 200* and you will hear a 200 hz tone, or you can also specify the frequency in 8ve.pc form, as in *hz 8.01,* which will beep the c# just above middle C back at you. The program will also print the 8ve.pc and hz equivalents of the pitch you request. (Note that you have to execute this program on the local machine, since you are calling the sounddriver.) Listen to your speech sample and see if you can estimate the top and bottom pitches by comparing with the beeps of the **hz** program. You will need to estimate the pitch range for the analysis program and you should give it specifications somewhat above and below the estimates you make. Then say **ptrack** -q. This will ask you for the file name, whether it has a header, the framsize (samples per analyzed segment)---use 350, the interframe offset (newsamples per segment (should be the same as the framesize in the lpc analysis) -- use 250, (this number must match the frame size of the lpc analysis) the low and high pitch estimates, the initial input skip and the duration of analysis. It will then grind for a few minutes. Here is a sample session: (my answers are given in italics)

```
ptrack -q
Enter soundfile name : demo2.snd
Is soundfile headerless? (y or n) n
Enter output file name : demo2.pch
Enter framesize (samples per analyzed segment) : 350
Enter interframe offset (new samples per segment) : 250
Enter low pitch estimate (cps) : 70
Enter high pitch estimate : 150
Enter initial skip (seconds) : 0
Enter duration of analysis (seconds) : 9

Writing to demo2.pch
Reading from demo2.snd
Sampling rate = 22050.000000
Pitch boundary estimates = 70.000000 (low) 150.000000 (high)
Framesize = 350
Interframe offset = 250
rflag: 0 Hflag: 0
```

4) Now you have to merge the pitch analysis and the lpc analysis. For this you use a program called **merge** Here is a sample session:

```
merge
 Enter name of lpc analysis file          demo2.lpc
 Enter name of pitch analysis file        demo2.pch
 Enter number of poles in lpc analysis   24
 Enter starting frame in lpc analysis    1
 Enter starting frame in pitch analysis 1
 Enter final frame in pitch analysis     999
Bad read on pitch analysis file
```

(I gave a final frame number that was higher than the size of the pitch analysis, and it ran off the end of the data set--not fatal.)

Now you have a dataset that is ready to use.   You can look at it with lpcplot
Here is part of a sample session

```
lpcplot demo2.lpc
 Enter npoles, width of plot, thresh       24 80 .01
Last frame is 290
----> ? n 60 80
    60 v0.0051 p   80.583 ***                                    a   310
    61 v0.0078 p   70.000                                        a   253
    62 v0.0048 p   70.000                                        a   335
    63 v0.0029 p   71.933                                        a   432
    64 v0.0053 p   93.906 *******                                a   338
    65 v0.0071 p   92.483 *******                                a   333
    66 v0.0104 p   70.000                                        a   335
    67 v0.0086 p   70.000                                        a   453
    68 v0.0024 p   74.358 *                                      a 1391**
    69 v0.0010 p   73.131                                        a 2939******
    70 v0.0010 p   76.954 **                                     a 2577*****
    71 v0.0022 p   70.000                                        a 2093****
    72 v0.0008 p 122.123 ****************                        a 4383**********
    73 v0.0007 p 121.746 ****************                        a 5734**************
    74 v0.0008 p 126.393 *****************                       a 6433***************
    75 v0.0013 p 128.179 ******************                      a 7619******************
    76 v0.0011 p 130.363 ******************                      a 7448******************
    77 v0.0013 p 133.729 ********************                    a 7729*******************
    78 v0.0012 p 137.701 *********************                   a 9186**********************
    79 v0.0013 p 142.222 **********************                  a 9578************************
    80 v0.0010 p 146.394 ************************                a 8990**********************

----> ?
```

n 60   80 means, draw frames 60 through 80.   With lpcplot it is possible to repair a bad pitch analysis.   Here is an explanation of the lpcplot commands
lpcplot commands:

1) n frame1 frame2
   will display frame1 to frame2.   If frame2 is 0 it will begin current chunk size on frame1.   If frame1 is negative it will backup that many frames and display current chunk size from there.   If neither frame1 or frame2 is specified it will simply display previous segment again.
2) p frame1 frame2 mult botcps topcps
   Multiply the pitches of frames frame1 to frame2 by mult if the current pitch is >= botcps and <=topcps.   default for botcps and top cps are 0 and 999999.   If mult is 0 it will interpolate pitches in these frames according to the frequencies of frame frame1-1 and frame2+1.   If frame2 is 0 it defaults to frame1. Thus p 100 will simply replace the frequency of frame100 by

interpolating between frames 99 and 101.

3) v frame1 frame2 value.
This will add value to the error numbers of frames frame1 toframe2. In this way you can force a voiced or unvoiced decision by subtracting 1 or adding 1, respectively.

4) null line (blank space-return)
this will simply display the next chunk of frames according to the currently defined chunk size.

You can use the lpcplot program to figure out where words begin and end. On the NeXT machines I recommend using the Shell application to do this so that you can review output easily. You can repair bad pitches with the p command, and force certain frames to be voiced or unvoiced with the v command. I recommend that you make a copy of your dataset before you begin to tinker with it. If you make a mistake you may not be able to reconstruct the original. (Sometimes you have to hit the return key twice--don't ask me why, this was the second C program I ever wrote.) Cntrl-c terminates the program.

There is also a program called **pchplot** which will act similarly on a pitch analysis data set.

1) n frame1 frame2
will display frame1 to frame2. If frame2 is 0 it will begin current chunk size on frame1. If frame1 is negative it will backup that many frames and display current chunk size from there. If neither frame1 or frame2 is specified it will simply display previous segment again.

2) p frame1 frame2 mult botcps topcps
Multiply the pitches of frames frame1 to frame2 by mult if the current pitch is >= botcps and <=topcps. default for botcps and top cps are 0 and 999999. If mult is 0 it will interpolate pitches in these frames according to the frequencies of frame frame1-1 and frame2+1. If frame2 is 0 it defaults to frame1. Thus p 100 will simply replace the frequency of frame100 by interpolating between frames 99 and 101.

3) null line (blank space-return)
this will simply display the next chunk of frames according to the currently defined chunk size.

4) Now you are ready to rock. The program is called lpcplay, and it writes a one channel output. Since amplitudes are unpredictable, it is best to write a floating point file and rescale it to integer format after you are done. Here is a sample Minc data file:

```
/* p0=start,p1=dur,p2=8ve.pch,p3=frame1,p4=frame2,p5=amp,p6=warp,p7=cf,p8=bw*/
/* p9,10,   p11,12,   p13,14 etc are optional pairs of frame,pitch statments.   Explained further below */

output("sf/demo2.synth")          /* open output file */
dataset("rlr/demo2.lpc",32)       /* open data set, second arg is number of
                                  poles */
float thresh,randamp,unvoicedrate,rise,decay
float start,amp,warp,bw,cf,fpw,frame1,frame2,transp
float fps,dur
```

```
lpcstuff(thresh=.01,   randamp=.10,      unvoicedrate= 0,rise=0,decay=0)
/* unvoiced rate feature currently unimplemented */

start=0   amp=1 warp=0 bw=0 cf=0

fps = 22050/250.     /* define frame rate */

frame1=40 frame2=410
lpcplay(start=0,dur=(frame2-frame1+1)/fps,transp=.0001,frame1,frame2,amp,warp,cf,bw)
setdev(30)
lpcplay(start=start+dur+2,dur=(frame2-frame1+1)/fps,transp=.0001,frame1,frame2,amp,warp,cf,bw)
setdev(8)
lpcplay(start=start+dur+2,dur=(frame2-frame1+1)/fps,transp=.0001,frame1,frame2,amp,warp,cf,bw)
setdev(0)
lpcplay(start=start+dur+2,dur=(frame2
 -frame1+1)/fps,transp=.0001,frame1,frame2,amp,warp,cf,bw,frame1+40,8.00,frame2,7.00)
lpcstuff(thresh=.01,   randamp=.10,      unvoicedrate=fps,rise=0,decay=0)
lpcplay(start=start+dur+2,dur=2*(frame2-frame1+1)/fps,transp=.0001,frame1,frame2,amp,warp,cf,bw)
lpcplay(start=start+dur+2,dur=2*(frame2
 -frame1+1)/fps,transp=9,frame1,frame2,amp,warp=10,cf,bw,frame2,6)
```

*dataset("lpcdataset",npoles)*

This opens an lpc data set.   You must specify the number of poles.   It should be stable,   and the pitch analysis must have been merged.

*lpcstuff(thresh, randamp)*

The thresh argument is a number against which the v argument in the lpc dataset will be compared.   If v is greater than thresh you will get unvoiced speech for that frame, if less, you will get voiced speech.   Generally a value of .01 is good.   You should look at your dataset, however, and verify if consonants are in this range. randamp is the relative amplitude of unvoiced signal.   Since the noise signal is likely to have a higher intensity than the voiced signal it is good to set this to about .1.   Different speakers may require different settings for the thresh and randamp.

*lpcplay(start,dur,transp,frame1,frame2,amp,warp,cf,bw [,frame1a,transp2,frame1b,transp3...*

start=start time
dur = duration of note
transp = transposition level.   If in 8ve.pc form the program will compute the overall pitch of the segment and transpose it to this level.   If tthe absolute value of transp is less than 1 the program will transpose the segment up or down   according to that interval as specified in 8ve.pc form (e.g. .02 = up two semitones, -.-4 =   down 4 semitones.   If   transp has an absolute value greater than one and is a negative value it will play the segment with a flat pitch at that level
frame1 = first frame
frame2 = last frame
amp = general amplitude multiplier
warp = formant shifter (in the range -1 to +1)   If you specify a number > 1 the program will compute the value for you according to the relation between the original pitch and the transposed pitch (this feature can cause instability).   Note that formant shifting slows   down the computation.

cf and bw: Center frequency (specified as a multiple of the pitch currently being used) and bandwidth (specified as a fraction of center frequency) for a reson on the output. If these are 0 this will be bypassed.

p9,10, p11,12 etc. These are pairs of frame and pitch values which will result in different transpositions for different segments of the note. The transposition level will be interpolated between these. frame1 and transp are considered the initial values, p9 and 10, the second pair etc. The level of transposition will then be interpolated between transp and p10, during the period of synthesis of frame1 to the frame specified in p9. If you want a single segment to have a single level of transposition you will have to specify two frames with the same transp, followed by a new frame, perhaps one frame later than the second with a new level. The transp arguments in p10-> must have the same mode as the transp argument specified for transp, either flat pitch or transposition of frame values.

*setdev(deviation)*

This will cause a weighted deviation figure to be computed for the given segment and the resultant synthesis stretched or shrunk depending on whether the deviation you specify is greater or less than the weighted deviation. Whenever a note is synthesized, the deviation figure will be printed. The default deviation is 0, which means that it will just abide by its normal range. calling setdev(0) will reset it.

*lpcin(start,dur,frame1,frame2,amp,inskip)*

This command will cause an input sound to be read into the filter rather than a buzz or rand. In order to do this you must open an input file with the input() command. Also, since the inputfile is not likely to have as many high frequencies as a buzz or rand, and since the lpc filters are generally lowpass filters, it is preferable to highpass the input signal first. You can do this easily enough with ein, or ellipse or fir. In ein the following simple filter should have a significant effect:

```
tap 1 1
y=.5 *(y-S1);
```

if you need anything steeper use the elliptical filter program. Unless you do this you will notice that the input signal going through the lpc filters will sound as if it is being played under water. If you use ein to create the file, there are several things you should do, particularly if the file is longer than 10 seconds. First set the "don't plot" button. Second, type your open file name in the outputfile form. Be sure to hit return when you finish typing. Finally, be sure to set the endtime in the input end form, otherwise you will just get the default 2 seconds, or whatever is in the dur form.