ANALYSIS OF PITCH CONTOURS IN REPETITION-DISFLUENCY USING STEM-ML

by

Rajiv M. Reddy

Faculty Advisor: Mark A. Hasegawa Johnson

ECE 497/499

May 2, 2006

ABSTRACT

F0 analysis-by-synthesis methods are used in order to test the hypothesis that the pitch contour in the alteration segment of disfluency tends to mimic the pitch contour in the reparandum segment of that disfluency. Reparandum-Alteration pairs selected by transcribers as having perceptually similar F0 contours were compared to arbitrarily selected fluent word-pair sequences using Stem-ML. All word-pair sequences had similar pitch; disfluent pairs were not more similar than others.

TABLE OF CONTENTS

1. INTRODUCTION

Spontaneous speech contains high rates of disfluencies like repetitions, repairs, filled pauses, etc. It is estimated that about 10% of all spontaneous utterances contain disfluencies [1]. Disfluency can be characterized by a three-region surface structure illustrated in figure 1 [2].

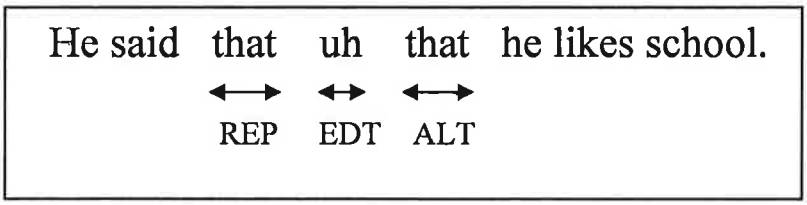

Figure 1. Illustration of the three-region structure of a "repetition-same-disfluency"

The three regions are the Reparandum (REP), the Edit (EDT) and the Alteration (ALT). The first region of disfluency, the REP, is the segment that is being replaced. At the end of the REP there is an interruption point where the speaker realizes that there was an error in the REP and decides to repair it. The next phase is the EDT phase which contains the region between the interruption point and the onset of repair. This region may be a silent pause or a filled pause (um, uh, I mean). The last region is the ALT which represents the repair of the REP and the resumption of fluent speech [2].

Disfluencies present a challenge for automatic speech recognition since they are often unmodeled in speech recognition systems [1]. A model that could detect disfluencies would decrease the errors in speech recognition systems. In this experiment, "repetition-same-disfluencies" are modeled using Stem-ML (Soft Template Mark up Language). In "repetition-same-disfluencies," the transcriber perceived the REP and the ALT to be the same.

Stem-ML is a tagging system that is used to describe intonation and prosody in human speech. These tags are used in automated training of accents shapes and parameters from acoustic databases [3]. Stem-ML is used to synthesize pitch contours of disfluent speech in this experiment.

Cole et al. proposed that "the frequent occurrence of parallel prosodic features in the reparandum (REP) and alteration (ALT) intervals of complex disfluencies may serve as strong perceptual cues that signal the disfluency to the listener [4]." The goal of this research is to test whether prosodic features in the REP and ALT (specifically, F0) resemble one another. The preliminary impression from looking at the data is that the REP and ALT seem similar; if so, this similarity might be used to detect disfluencies.

2. STEM–ML PARAMETERS

Certain features of Stem-ML described below are the most relevant to understanding how the hypothesis was tested.

2.1. Stress Tags

The *stress* tag specifies the local F0 contour of a period of time normally corresponding to a syllable or word [3]. In our system, the REP and ALT of each disfluency are assumed to be instantiations of the same stress tag. Different utterances use different stress tags. The *stress* tag is defined by a number of parameters. The most important to this experiment are the *strength* and the number of points that are trained. The parameters that are irrelevant to the experiment were set to 0 and have hence been taken out of the original Stem-ML equations [5].

The final synthesized f0 is given by

$$f_0(t) = p(t) * range + base \tag{1}$$

where *p(t)* is the normalized F0 relative to the range of the speaker [6]. *p(t)* is a compromise between the articulatory effort (G) and the weighted error (R).

$$p(t) = \arg\min_{p(t)} (G + R) \tag{2}$$

The articulatory effort (G) is represented by

$$G = \sum_t \left(\dot{e}_t^2 + \tau^2 \ddot{e}_t^2 \right) \tag{3}$$

where τ is a constant and the dots represent derivatives. The weighted error (R) is the sum of the errors in each tag weighted by the tag *strength* s_k [8].

$$R = \sum_{k \in \text{tags}} s_k^2 r_k \tag{4}$$

and r is the error for each template.

$$r_k = \sum_{t \in \text{tag k}} \cos(type.\pi/2)\left((e_t - \bar{e}_t) - (y_{k,t} - \bar{y}_k)\right)^2 + \sin(type.\pi/2)(\bar{e}_k - \bar{y}_k)^2 \tag{5}$$

where type is set to 0 in all the experiments.

$$\Rightarrow r_k = \sum_{t \in \text{tag k}} \left((e_t - \bar{e}_t) - (y_{k,t} - \bar{y}_k)\right)^2 \tag{6}$$

where e_k is the normalized F0 contour, $\bar{e}_k$ is the average normalized F0 contour of that tag, y_k is the normalized target F0 contour and $\bar{y}_k$ is the average normalized target F0 contour for that *stress* tag [7].

2.2. Strength

Strength controls the interaction of accent tags with their neighbors. If the strength tag is low the smoothness of the synthesized F0 contour is more important than the accuracy [8].

2.3. Base and Range

The base and range are speaker dependant constants. To reduce the number of parameters Stem-ML needs to learn, the base and range are calculated outside of Stem-ML. The base is estimated as the 25th percentile of the F0 in each file and the range is estimated as the difference between the 25th and 75th percentile of the F0 contour in that file.

2.4. Word Scale

This parameter is represented by the variable *wscale*. It is the ratio of the length of the segment on which the tag is being placed to the length of the original F0 contour. This allows us to place tags on two segments of different lengths since it stretches or compresses the stress tag according to the length of the segment i.e. REP/ALT.

The *wscale* of the alteration stress tag is set to 1 and the wscale ratio for the REP is set to duration (REP) / duration (ALT).

2.5. Step to tags

The *Step to* tag forces the phrase curve to have a certain frequency at the tag's position. It is specified as a fraction of the speaker's range [7].

2.6. Center Shift

This parameter allows the stress tag to be shifted within the ALT or REP to give a better fit. If it is set to 0, Stem-ML is forced to follow the start and end given by the transcriber. If it is allowed to be used as a parameter that is learnt by Stem-Ml, it moves the stress tag around to minimize the error in the fit.

3. DATA

The database used for these experiments is a subset of Swtichboard. It is the same data set that was transcribed for [3]. The data contain 71 two minute blocks of speech with added transcription tiers including disfluency type (repetition, repair, …), disfluency segment (REP, EDIT, ALT), and perceived relationship between REP and ALT pitch contours (same, stress, phrase boundary, …). Tokens marked "repetition-same-disfluency" were extracted; these are repetition disfluencies in which the REP and ALT F0 contours were perceived by the transcriber as sounding the same. The REP and ALT segment markings bound the domain of stress tags in Stem-ML. For comparison, fluent word pairs were extracted: a fluent word pair contains any two words uttered sequentially during normal fluent speech.

4. EXPERIMENTAL DESIGN

To test the hypothesis that REP mimicked ALT, Stem-ML models with tags are created to represent the disfluent speech. Stem-ML is forced to learn the same *stress* tag (pitch contour) for the reparandum and alteration. If REP mimics ALT, we should get a lower RMS pitch error value per sample in disfluent word pairs as compared to fluent word pairs.

4.1. Parameters

The *strength* of the *stress* tags are varied to see the effect of changing the strength on the RMS of the pitch error per sample. We set the *stress* tag to learn 3 points i.e. the 25^{th}, 50^{th} and 75^{th} percentile for each placement on the pitch curve. *Ctrshift* was set to 0 or learnt by Stem-ML.

The model is used to learn the pitch contour of each REP/ALT pair, and the RMS error per sample for each disfluent pair is calculated. To compare these values with fluent speech we run the same model on randomly selected consecutive words of fluent speech.

4.2. Implementation

The experiment was implemented in Python. The data came from 2 sources. The first source was the .wav files which gave the f0's and the other source was .TextGrid files which gave the word boundaries, disfluency markings and disfluency types. The data from the .TextGrid files were extracted by using a Praat script that was generated by the Python script for each file. This was stored in variables in the Python script. The f0 was obtained from the .wav files by running the *get_f0* script on them. With this information

model files were created for each individual case from the Python script and saved as .pf files. The Python script then ran *resid_for_opt.py* which is the Stem-Ml python script to learn the parameters for the disfluency case. After the script terminates the fitted f0 is plotted. If the machine learning script *resid_for_opt.py* terminated improperly due to errors, nothing will be plotted. If this happens than the file name and the disfluency case number is saved onto the log file and it is stated that the file crashed. If the learning script goes to completion and terminates properly we will get a proper plot. Then the file name and disfluency case number is saved onto the log file along with the calculated E (Error energy as calculated by Stem-ML) value and the MSE (Mean Square Error) value. E is defined by

$$E = \sum_{t \in disfluency} \left(\frac{(e_t - y_t)^2}{\sigma^2} \right) - \log(\sigma^2) * dof \tag{7}$$

where e_t is the synthesized F0 in erbs, y_t is the target F0 in erbs, σ is the standard deviation and *dof* is the degrees of freedom. The E values are normalized by the degrees of freedom to compare results from different models [9].

Numerous experiments were conducted and the E values were gathered. The experiments included varying the strength values, setting the ctrshift to a particular value, learning 4 points instead of 3, etc. It was found in all the experiments that fluency produced better results than disfluency which was contrary to the hypothesis. Towards the end of the year, the MSE values were calculated as well for some of those experiments to find that the MSE values showed an even larger deviation from the hypothesis. Since it is still unclear if the E values are accurate measures to differentiate between fluency and disfluency only the MSE values have been reported here.

5. RESULTS

The mean, median and standard deviation of the RMS pitch errors for fluency and disfluency cases are shown in Table 1 and Table 2 for two experiments with different parameter values.

Table 1: RMS pitch error for fluent speech cases and disfluent speech cases after training with strength of the stress tag= 8. The *step_to* tags are forced to jump to the same frequency at the beginning of each tag.

	Fluency Errors		**Disfluency Errors**	
	RMS	**Normalized E**	**RMS**	**Normalized E**
Mean	11.47 Hz/Sample	0.37 Erbs/sample	18.29 Hz/Sample	0.45 Erbs/sample
Median	7.91 Hz/Sample	0.13 Erbs/sample	15.62 Hz/Sample	0.47 Erbs/sample
StdDev	9.13 Hz/Sample	0.60 Erbs/sample	14.24 Hz/Sample	0.37 Erbs/sample

Table 2: RMS pitch error for fluent speech cases and disfluent speech cases after training with strength of the stress tag= 8. The *step_to* tags of REP and ALT are allowed to jump to different frequencies that are learnt during machine learning and minimization of the error.

	Fluency Errors		**Disfluency Errors**	
	RMS	**Normalized E**	**RMS**	**Normalized E**
Mean	11.53 Hz/Sample	0.38 Erbs/sample	18.02 Hz/Sample	0.77 Erbs/sample
Median	8.21 Hz/Sample	0.13 Erbs/sample	13.71 Hz/Sample	0.39 Erbs/sample
StdDev	8.70 Hz/Sample	0.60 Erbs/sample	13.41 Hz/Sample	0.96 Erbs/sample

Table 3: Normalized E values for fluent speech cases and disfluent speech cases for a few different experiments:
(a) The EDT region is forced to be learned with the same strength(=8) as the REP and ALT stress tags. Also the wscale parameter is set by the user according to the length of the ALT and REP.
(b) The EDT region is forced to 0 irrespective of whether it is a voiced or silent EDT. The wscale parameter is learnt by stem-ML
(c) The EDT region is learnt with strength 1 while the stress tags are at strength 8, the wscale parameter is learnt by Stem-ML
(d) Same as experiment (b) except that 4 points are learnt instead of 3

Experiment	**Fluency Errors** (Erbs/sample)			**Disfluency Errors** (Erbs/sample)		
	Mean	**Median**	**StdDev**	**Mean**	**Median**	**StdDev**
(a)	0.1625	0.1177	0.1141	0.3555	0.2283	0.3419
(b)	0.5343	0.1983	0.7562	0. 4006	0.2581	0.4499
(c)	0.3679	0.2201	0.3528	0.3692	0.2396	0.4058
(d)	0.2429	0.0920	0.2729	0.3973	0.2870	0.4318

Contrary to the hypothesis, a lower average RMS pitch error per sample is found in the fluent word pairs than in the disfluent pairs. There is a high standard deviation, which means that the error rates are not concentrated at a particular range, and are instead distributed more uniformly. In the few cases that higher means are found fluency, it is due to a few extreme values in the results. Figures 1 and 2 contain some examples of the fitting for fluency and disfluency cases.

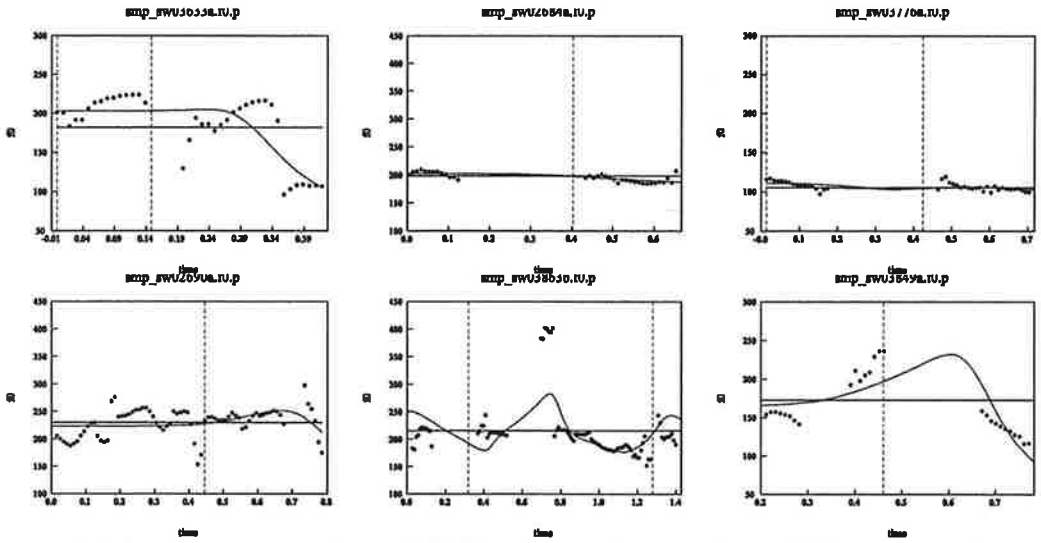

Figure 2. These are some plots of disfluent speech cases from different experiments. The big black dots represent the original F0, the flat line represents the baseline F0 and the curved line represents the learnt F0.

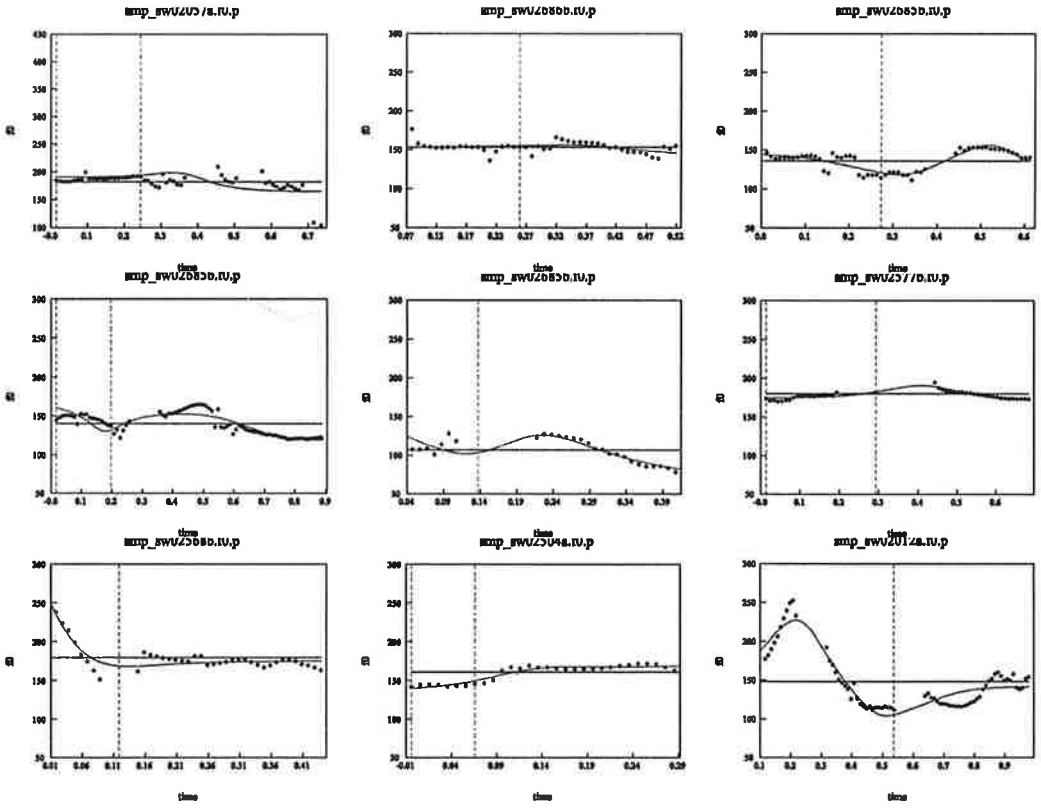

Figure 3. These are some plots of fluent speech cases from different experiments. The big black dots represent the original F0, the flat line represents the baseline F0 and the curved line represents the learnt F0.

6. DISCUSSION AND CONCLUSIONS

We found fluent pairs to have a lower pitch error than disfluent cases even though the model was constructed to force the REP and ALT to mimic each other. We cannot differentiate between fluency and disfluency by RMS pitch error. It is not possible to demonstrate, experimentally, that the F0 contour of reparandum mimics that of alteration. Rather, it seems that any two consecutive words have similar pitch contours since Notice that, when using one fluent word to predict the next word's F0, we incur an RMS error of only 11.47Hz.

One possible conclusion is that the Switchboard database is primarily monotone. In order to explore the hypothesis that switchboard is monotone, I calculated F0 standard deviation as a percentage of F0 mean in each utterance file. 31 out of 71 files have an F0 standard deviation that is less than 16% of the mean value. The histogram is shown in figure 3.

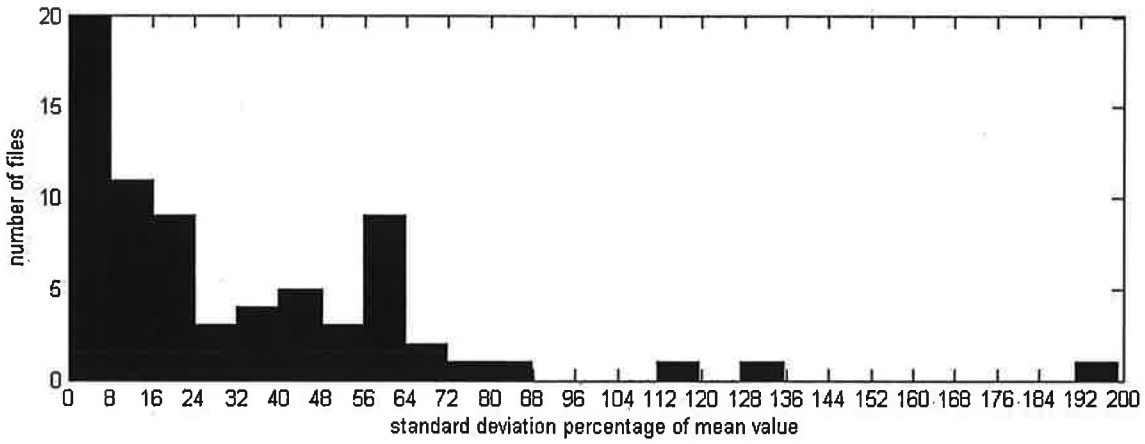

Figure 3. Histogram of the standard deviation percentage of the mean value in the switchboard file.

Thus, the lower average RMS pitch error per sample for fluency cases may be due to the fact that a large part of the database is spoken in monotone; disfluency does not reduce the difference between successive words because all word pairs have similarly flat F0 contours.

In conclusion, there were no cues detected by Stem-ML that could be used to differentiate between repetition-same-disfluency and fluent speech.

7. ACKNOWLEDGEMENTS

This research was supported by REU funding under NSF grant IIS 04-14117.

Conclusions are those of the authors, and are not endorsed by the NSF. Special thanks go to Chilin Shih and Greg Kochanski for their valuable comments.

8. APPENDIX 1. CODE IN PYTHON FOR DISFLUENCY

This is the code that was used to do the experiments on disfluency cases. A similar code

was used to detect fluency cases.

```
#!/usr/bin/python
# This file needs to be run 1 folder deep in the location of the .wav and .TextGrid files
import fileinput
import os
import string
import datetime

# This sets the current directory to what is specified in the parenthesis.
# In this case the parent directory is made as the current directory is the code is stored in
the /code folder.
os.chdir('..')
filelist=[]

# this lists all the files in the directory in the parenthesis
names =os.listdir('.')
#I am stripping the ".wav" from the file name so that I can use the name to create files
with different extensions
for i in range(len( names)):
      if names[i][-3:]=="wav":
           filelist.append( names[i][:-4])
filelist.sort()

#This line may be used if the session suddenly closes so that files that have already been
processed can be skipped over.
#filelist=filelist[28:]

#This cleans out all the old files
os.system("rm *.pf *.f0 *.f0.p *start_end *.ps *.praat *_dis *_fit foo* *.tmp pfile.map
>& output_tmp")

#This creates the log directory to save all the logs
os.system("mkdir log >& output_tmp")

#This opens up the log file and writes the date and some description information to
reference each job
log = open("log/disfluency_log", 'a')
date = str(datetime.datetime.today())
log.write("\nstart  "+date[:-10])
log.write("\nstep=1,stress=1,3 pts, edt=1,to=XXX,ctr=XXX,wscale word2=1")
```

```
log.close()

#To test only 1 file at a time uncomment the next 2 lines and comment out the third line.
# also 'i' would need to be set to index of the file in filelist[i] to test a particular file

#i=0
#if i==0:
for i in range(len (filelist)):

# This tracks the F0
    os.system("get_f0 -P Pget_f0 %(a)s %(b)s >& output_tmp"
%{'a':filelist[i]+".wav",'b':filelist[i]+".f0"})
    os.system("pplain %(a)s >& output_tmp > %(b)s " %{'a':filelist[i]+".f0",
'b':filelist[i]+".f0.p"})
    os.system("awk '{print $1, $2}' < %(a)s > tmp" %{'a':filelist[i]+".f0.p"})
    os.system("mv tmp %(a)s" %{'a':filelist[i]+".f0.p"})

# This sorts the F0 so that the 25th and 75th percentile can be used for the base and range
# as an approximation. The F0 is stored in 2 formats. In the variable "f0" contains only
# the voiced frequency data which is sorted to get the mean and range. The variable
# "voicing_info" stores the F0 in the original order
    file = open(filelist[i]+".f0.p", 'r')
    data = file.readlines()
    file.close()
    voicing_info=[]
    f0=[]
    for j in range(len(data)):
        k=data[j].strip().split()
        voicing_info.append(k)
        if k[1]=='1':
            f0.append(float(k[0]))
    f0.sort()

# This creates a praat file to extract the relevant information from the TextGrid files.
# This saves the min max and label into a new file that can easily be read into python
# The new file is saved as "FileName" appended with "_dis"
    praatfile = open("disfluency_TextGrid.praat" , 'w')
    praatfile.write("directory$=\".\" \n")
    praatfile.write("clearinfo \n")
    praatfile.write("fileName$ = %(a)s\n" %{'a':"\""+filelist[i]+".TextGrid\""})
    praatfile.write("Read from file... 'directory$'/'fileName$'\n")
    praatfile.write("fileName$ = fileName$ - \".TextGrid\"\n")
    praatfile.write("select TextGrid 'fileName$' \n\n")
    praatfile.write("nItem1 = Get number of intervals... 1 \n")
    praatfile.write("nItem2 = Get number of intervals... 2 \n")
    praatfile.write("nItem3 = Get number of intervals... 3 \n\n")
```

```
    praatfile.write("for type to nItem2 \n")
    praatfile.write("   min2 = Get starting point... 2 type \n")
    praatfile.write("   max2 = Get end point... 2 type \n")
    praatfile.write("   label2$ = Get label of interval... 2 type \n")
    praatfile.write("   if label2$ == \"hesi-r-same\" \n")
    praatfile.write("       for form to nItem3 \n")
    praatfile.write("           min3 = Get starting point... 3 form \n")
    praatfile.write("           max3 = Get end point... 3 form \n")
    praatfile.write("           label3$ = Get label of interval... 3 form \n")
    praatfile.write("           if (min3 >= min2) and (max3 <= max2 ) \n")
    praatfile.write("               print 'min3' 'tab$' 'max3' 'tab$' 'label3$' 'newline$' \n")
    praatfile.write("           endif \n")
    praatfile.write("       endfor \n")
    praatfile.write("   endif \n")
    praatfile.write("endfor \n")
    praatfile.write("select TextGrid 'fileName$' \n")
    praatfile.write("Remove")
    praatfile.close()
    os.system("praat disfluency_TextGrid.praat > %(a)s" %{'a':filelist[i]+"_dis"})

# This reads the disfluency data saved by the praat file and makes a 3D array dis with
# rows= each disfluency case, columns = forms in the disfluency eg alt rep edt ,
# depth = start end and label of each form
    file = open(filelist[i]+"_dis", 'r')
    data = file.readlines()
    file.close()
    m=[]  #temporary variable to hold each disfluency case
    dis=[]  #holds a 3D array with disfluency case, disfluency forms, start and end
timings

    for j in range(len(data)-1):
        k=data[j].strip().split()
        f0_counter1=0
        flag=0
        if k!="":
            for f in range(int((float(k[1])-float(k[0]))*100)):
                if voicing_info[f+ int(100*float(k[0]))][0]!='0':
                    f0_counter1 =f0_counter1+1
            if k[2] !="edt" and f0_counter1 < 7:
                flag=1
            m.append(k)
            if k[2] =="alt":
                if flag==0:
                    dis.append(m)
                m=[]
```

```
    if len(dis)==0:
        log = open("log/disfluency_log", 'a')
        log.write("\n\t"+filelist[i]+"  has no disfluency case")
        log.close()

# This creates the model file. Seperate model files are created for each disfluency
# case even if the disfluencies are from the same ".wav" file
    for j in range(len(dis)):
        log = open("log/disfluency_log", 'a')
        os.system("rm foo* a.tmp pfile.map best_energy >& output_tmp")
        modelfile = open(filelist[i]+str(j)+".pf", 'w')

# These are the settings in each model file.
        modelfile.write("ltype=settings; base= %(a)s; \n"  %{'a':f0[len(f0)*25/100]})
        modelfile.write("ltype=settings; range= %(a)s; \n" {'a':float(f0[len(f0)*75/100])-
float(f0[len(f0)*25/100])})
        modelfile.write("ltype=settings; add=0;\n")
        modelfile.write("ltype=settings; max=800;\n")
        modelfile.write("ltype=settings; min=20;\n")
        modelfile.write("ltype=settings; pdroop=0;\n")
        modelfile.write("ltype=settings; adroop=0;\n")
        modelfile.write("ltype=settings; jitter=0.0;\n")
        modelfile.write("ltype=settings; jittercut=1.0;\n")
        modelfile.write("ltype=settings; smooth=0.05;\n")

# This is the declaration of the stress tag with 3 points.
        modelfile.write("ltype=aclass; name=REPAIR; atype=1; stype=0; type=0;
wscale=1;")

    modelfile.write("shape=[(0.25,XXX),(0.50,XXX),(0.75,XXX)];ctrshift=math.exp(X
XX);\n\n")
        edt_flag=0
        for k in range(len(dis[j])):

# This creates the same step_to tag and stress tag for each reperandum and the alteration.
# In case of an edt it creates only a step to tag.
# It also creates the phrase end at the position of the end of the alt
            if dis[j][k][2]!="edt" and dis[j][k][2]!="alt":
                modelfile.write("ltype=tag; Cname=step_to; pos=%(a)s; strength=8;"
%{'a': int(100*float(dis[j][k][0]))*0.01})
                modelfile.write(" to=math.exp(XXX); name=%(a)s \n"
%{'a':filelist[i]+".f0.p"})
                modelfile.write("ltype=tag; Cname=stress; ac_name=REPAIR;
pos=%(a)s;" %{'a': float(int(100*float(dis[j][k][0])))*0.01})
```

```
                modelfile.write("strength=8; wscale=%(a)s ; " %{'a':
int(100*(float(dis[j][k][1])-float(dis[j][k][0]))/(float(dis[j][-1][1])-float(dis[j][-
1][0])))*0.01})
                modelfile.write ("name= %(a)s \n\n"%{'a':filelist[i]+".f0.p"})

            if dis[j][k][2]=="edt":
                modelfile.write("ltype=tag; Cname=step_to; pos=%(a)s;strength=8;"
%{'a': float(int(100*float(dis[j][k][0])))*0.01})
                modelfile.write(" to=math.exp(XXX); name=%(a)s \n" %
{'a':filelist[i]+".f0.p"})
                edt_flag=1

            if dis[j][k][2]=="alt":
                modelfile.write("ltype=tag; Cname=step_to; pos=%(a)s; strength=8;"
%{'a': int(100*float(dis[j][k][0]))*0.01})
                modelfile.write(" to=math.exp(XXX); name=%(a)s \n"
%{'a':filelist[i]+".f0.p"})
                modelfile.write("ltype=tag; Cname=stress; ac_name=REPAIR;
pos=%(a)s;" %{'a': float(int(100*float(dis[j][k][0])))*0.01})
                modelfile.write("strength=8; wscale=1 ; ")
                modelfile.write("name=%(a)s \n\n"%{'a':filelist[i]+".f0.p"})
                modelfile.write("ltype=tag; Cname=phrase; pos=%(a)s; name=%(b)s
\n" %{'a':float(int(100*float(dis[j][k][1])))*0.01,'b':filelist[i]+".f0.p"})
                modelfile.close()

#This creates a start file for each disfluency case
        start_end_file = open(filelist[i]+str(j)+"_start_end", 'w')
        start_end_file.write(" file=%(a)s; start=%(b)s; end=%(c)s\n"
%{'a':filelist[i]+".f0.p",'b':int(100*float(dis[j][0][0])),'c': int(100*float(dis[j][len(dis[j])-
1][1]))})
        start_end_file.close()
#This creates the parameter file model.
        os.system("XXX_to_pfile.py %(a)s > %(b)s" %{'a':filelist[i]+str(j) +
".pf",'b':filelist[i] + str(j) + "_model.pf"})

# This learns the parameters for each disfluency case.
        os.system("resid_for_opt.py %(a)s %(b)s `list_prm.sh foo.cf` > foo.2"
%{'a':filelist[i]+str(j) + "_start_end",'b':filelist[i] + str(j)+"_model.pf"})

# This updates the parameters learnt and saves the error output in best_enerygy.
        os.system("update_opt_in.sh foo.cf a.tmp >& best_energy")

# This creates a graphical plot of the fit.
        os.system("plot_stem_fit.py -o %(a)s -gr 50 300 -g %(b)s %(c)s %(d)s %(e)s
`list_prm.sh foo.cf` "%{'a':filelist[i]+str(j) + "_fit",'b':filelist[i]+str(j) + "_foo",'c':filelist[i]
+ str(j) + "_start_end",'d':filelist[i]+str(j) + "_model.pf", 'e':filelist[i]+".f0.p"})
```

```
# The rest of the code is to calculate a value for "E" that can be used
# to compare between wav files and to find the MSE of the model

# This calculates and NP for the degrees of freedom
        if edt_flag==1:
            NP=6
        else:
            NP=5

# This creates the log file
        log.write("\n"+filelist[i]+str(j)+"\t")
        try:
            file = open(filelist[i]+str(j)+"_fit", 'r')
            fit_data = file.readlines()
            file.close()
            rms=0
# the variable "counter" counts the number of voiced samples
            counter=0
            for j in range(len(fit_data)):
                k=fit_data[j].strip().split()
                if len(k)==6:
                    if k[4]=='1':
                        rms=rms+(float(k[3])-float(k[1]))*(float(k[3])-float(k[1]))
                        counter= counter+1

            file = open("best_energy", 'r')
            data = file.readlines()
            file.close()
            k=data[0].strip().split()
            if k[1][7:-1]!="":
                log.write(str(float(k[1][7:-1])/(counter/2-NP))+"\t"+str(rms/counter))
            else:
                log.write("crashed"+"\t"+str(rms/counter))
        except:
            log.write("crashed")

        log.close()
log = open("log/disfluency_log", 'a')
date = str(datetime.datetime.today())
log.write("\nend   "+date[:-10]+"\n\n")
log.close()
```

9. REFERENCES

[1] C. Nakatani and J. Hirschberg, "A corpus-based study of repair cues in spontaneous speech." *Journal of the Acoustical Society of America*, vol. 95, no. 3, pp. 1603-1616. 1994.

[2] Shriberg, E. "To 'errrr' is human: ecology and acoustics of speech disfluencies," *Journal of the I.P.A.*, vol. 31, no. 1, pp. 153-169. 2001

[3] J. Cole, M. Hasegawa-Johnson, C. Shih, H. Kim, E. Lee, H. Lu, Y. Mo, T. Yoon, "Prosodic parallelism as a cue to repetition and error correction disfluency", *DiSS*, pp. 53-58. 2005

[4] G. Kochanski, C. Shih, "Stem-ML: Language independent prosody description", *ICSLP*, vol. 3, pp. 239-242. Beijing, China. 2000

[5] G. Kochanski, C. Shih, "Prosody modeling with soft templates", *Speech Communication,* vol. 39, no. 3-4, pp.311-352, February 2003.

[6] G. Kochanski, C. Shih and H. Jing, "Quantitative measurement of prosodic strength in Mandarin", *Speech Communication*, vol. 41, no. 4, pp. 625-645, November 2003.

[7] G. Kochanski, C. Shih and H. Jing, Erratum to "Quantitative measurement of prosodic strength in Mandarin", *Speech Communication*, vol. 47, no. 3, pp. 394, November 2005.

[8] C. Shih, G. Kochanski, "Modeling of Vocal Styles Using Portable Features and Placement Rules," *International Journal of Speech Technology*, vol. 6, no.4, pp. 393-408. October 2003.

[9] G. Kochanski, personal communication, March 2006