Facilities to assist people to research into stammered speech

Peter Howell[1] and Mark Huckvale[2]

[1] *Department of Psychology, University College London, Gower St., London WC1E 6BT*
p.howell@ucl.ac.uk

[2] *Department of Phonetics and Linguistics, University College London, Gower St., London WC1E 6BT*
m.huckvale@phon.ucl.ac.uk

Abstract. The purpose of this article is to indicate how access can be obtained, through *Stammering Research*, to audio recordings and transcriptions of spontaneous speech data from speakers who stammer. Selections of the first author's data are available in several formats. We describe where to obtain free software for manipulation and analysis of the data in their respective formats. Papers reporting analyses of these data are invited as submissions to this section of *Stammering Research*. It is intended that subsequent analyses that employ these data will be published in *Stammering Research* on an on-going basis. Plans are outlined to provide similar data from young speakers (ones developing fluently and ones who stammer), follow-up data from speakers who stammer, data from speakers who stammer who do not speak English and from speakers who have other speech disorders, for comparison, all through the pages of *Stammering Research*. The invitation is extended to those promulgating evidence-based practice approaches (see the *Journal of Fluency Disorders, volume 28, number 4* which is a special issue devoted to this topic) and anyone with other interesting data related to stammering to prepare them in a form that can be made accessible to others via *Stammering Research*.

Keywords: UCL Psychology speech group database http://www.speech.psychol.ucl.ac.uk/, SFS http://www.phon.ucl.ac.uk/resource/sfs/, CHILDES http://childes.psy.cmu.edu, PRAAT *http://www.praat.org*, the Wellcome Trust *http:wellcome.ac.uk.*

1. Introduction

Over the last 20 years, spontaneous speech data from speakers who stammer have been collected by the Speech Group in the Psychology Department of University College London (UCL). These data have mainly been collected through research funding from the Wellcome trust who, as a matter of policy, encourage public access to science. A proportion of these data has been transcribed. All these materials will be shared with the research community with the intention of encouraging research into stammering. In the future, we plan to make other types of data available provided by our own, and hopefully other, groups. Some background considerations about the choice of approach to make these data available are given in section 2 of this article.

Section 3 includes a description of the data we hold, and indicates which subset we are currently making available. The transcribed data are available in CHAT, PRAAT TextGrid and (in some cases) as SFS annotation items aligned against the audio records. The audio data have been prepared in WAV, SFS and MP3 formats. Undoubtedly there are other packages available too and authors or users of those systems are welcome to prepare the data so that they can be processed with them. Users can submit articles that make the case for including other formats. If such articles are accepted after peer review, they will be invited to prepare the data in that format and it will be included in the archive.

Providing samples of data from speakers who stammer is an important step in encouraging research into stammering. However, people wishing to do research also need to have facilities to manipulate these data (for analyses of speech characteristics, to use these data in perceptual assessments and so on). The formats that are provided allow readers who are familiar with CLAN, PRAAT and SFS programs to investigate these data. Section 4 gives tutorial material and some illustrative analyses using the SFS software suite. SFS is used by the UCL speech group for the reaons also given in section 4. It is necessary to emphasize that exclusive use of SFS is *not* being advocated; CHILDES and PRAAT each have facilities that are not available in SFS (and vice versa). Thus, different software packages should be chosen according to what analyses end-users wish to perform. As indicated in the previous paragraph, some preparation has been done on the data so that each of these software packages can process the data. Future issues of *Stammering Research* will include details and demonstration of some of the capabilities of these other software suites. A general feature that commends SFS, CHILDES and PRAAT, is that they are all available for free. Details of how to access these software are included.

In addition to data and software for analyzing these data, researchers need an outlet for their findings. It is suggested that researchers consider *Stammering Research* as such an outlet. This will allow an archive of

analyses to be built up on data available through the journal, provide a forum to make corrections to the data, and offer a repository that can be used to access new software that are useful for analysis of data like these.

This is an extensive document. It has been written as far as is possible so that the different sections can be read in isolation. The body of the text gives the description of what data and software are available and also reports studies using these data that we hope have some intrinsic interest to readers. The material in the Appendices will need to be consulted when a reader has some specific need. Appendix A will be used when the user wishes to select some of the data in whatever format they need. Appendix B describes the machine-readable transcription convention that UCL Psychology's Speech Group has adopted which users may choose to use or convert to one they are more familiar with. Appendices C and D give worked exercises using SFS utilities in two important areas for manipulation of these data (transcription and formant analysis). Appendix E gives some details of applications of a Hidden Markov model software suite to recognition of dysfluencies. The background knowledge that is assumed is given at the start of each Appendix. These appendices were, in part, written as tutorial material for this article though they also serve as important sources for general users of SFS. Also, applications in the body of this paper draw on the information provided in these appendices. For all these reasons, it is appropriate that they be included in this document. It should be apparent that these facilities are only part of what SFS offers. There is extensive other software in and various options for people wishing to write their own scripts (in MATLAB, C, and in speech measurement language, SML which is a high level scripting language for manipulating information in SFS files, for details of SML see section 1.5 of the SFS manual at http://www.phon.ucl.ac.uk/resource/sfs/). The CHILDES and PRAAT systems have various similar options that, it is hoped, will be described in future articles.

2. Background to release of data on speakers who stammer and software for its analysis

Speech data are expensive to collect, speech is time-consuming to analyze, the range of analyses that one group can do is limited (e.g. because of time constraints). Individuals who want to start a research program into stammering often have the daunting tasks of obtaining expertise, equipment, software and an administrative structure (to locate participants, obtain ethical permission etc.) before they can conduct their research. If they are interested (like us) in developmental issues associated with stammering, they may have to collect longitudinal data for several years.

There are pitfalls when attempting to make recordings. Clinics and schools are not ideal recording environments. (Many recordings that speech therapists have offered us in the past have been too poor in quality for analysis.) Special skills need to be acquired for eliciting speech from young speakers and also some children who stammer.

UCL Psychology Department's speech group has extensive audio data on speakers who stammer of appropriate quality for linguistic and acoustic analysis, which it is prepared to supply to the wider community thereby circumventing these problems in 'getting started' in research into stammering. Generally speaking, three issues need to be addressed so the data can be used. These are 1) conventions for data preparation need to be specified, 2) information needs to be supplied about how to use available software to manipulate data in these formats, and 3) information has to be provided about where and how other similarly formatted data can be deposited or accessed.

The way in which three different systems address these issues is discussed. The three systems selected for consideration are MacWhinney's CHILDES system, Boersma's PRAAT system and Huckvale's SFS system. Each of these systems was developed for different purposes and each has advantages that make it more useful for some purposes than are the others. This is implicitly acknowledged as, for instance, there is provision in the CHILDES system to access PRAAT software to take advantage of the facilities for speech analysis provided by PRAAT. The components of each system and the purpose for which they were developed are briefly indicated.

CHILDES. The CHILDES project was developed specifically for research into child language. It is masterminded by Brian MacWhinney and has three separate components that address the above three issues: CHAT (Codes for the Human Analysis of Transcripts) provides the data conventions, CLAN (Computerized Language ANalysis) is the software package and CHILDES (CHIld Language Data Exchange System) is the repository for the data. The three components together are referred to as the CHILDES project (MacWhinney, 1995). CHILDES was developed for higher level linguistic analysis. This makes sense in terms of the target populations for which the system was developed. Lengthy recording sessions are often necessary to obtain even limited samples of speech from very young children.

Consequently, it would not make sense to archive the entire audio recordings in these cases (a lot of the data so stored would be taken up by interlocutors and sections where the child says nothing). The system has been developed so that audio data can be logged and there are links to audio analysis software (e.g. PRAAT). This is particularly useful when samples from older speakers are employed. Most of the data that are available through CHILDES are from fluent speakers. Further details can be found on the CHILDES home page at http://childes.psy.cmu.edu.

PRAAT. PRAAT was developed by Paul Boersma and is specifically an audio data analysis tool and PRAAT has the advantage that it is simple to use. It reads all kinds of standard audio format files including WAV. It runs on Unix, Mac and Windows platforms. Transcription data are in the form of a TextGrid which can easily be created within the PRAAT system or imported via a suitably formatted text file. The software incorporates sub-tools for neural net analysis, speech synthesis and some statistical functions, and a scripting facility is available which allows users to write specialized functions. There is a PRAAT user group who share data (as well as software). PRAAT is close to SFS in the facilities it provides, but each of them have specialized processing software. For example, PRAAT does sophisticated analyses of harmonicity while SFS incorporates software for dealing with ancillary signals provided from a laryngograph . The PRAAT Home Page is at http://www.praat.org and the manual is accessed within the software.

SFS. SFS stands for Speech Filing System. It provides an integrated method of dealing with different sources of information about speech sounds. The raw audio record is at the core of the system. There are options so that a number of other analyses (manual or computational) on the same speech data can be displayed for inspection. Transcriptions can be manually entered in any format (Appendix B gives the Joint Speech Research Unit format, JSRU, that the UCL Speech Group uses). The filing system provides the system that integrates analyses from these several sources for visual or statistical inspection. The integration of these sources of information is the attraction, though there are utilities that allow the audio recordings to be uploaded or dumped in WAV or other standard formats and, similarly, TXT files can be dumped or uploaded.

For the Speech Group's work, the SFS facility that allows, *inter alia*, audio data and aligned transcriptions to be concurrently displayed has been particularly useful in the development of Howell's EXPLAN theory of spontaneous speech control (Howell, 2002, 2004; Howell & Au-Yeung, 2002). This theory maintains that motor execution of one segment takes place concurrent with the planning of the following segment and that fluency may break down when execution time on one segment does not allow sufficient time to complete planning of the following segment. SFS displays of the audio waveform and the associated transcription provide the information necessary for evaluation of predictions of this theory. Thus, the audio item provides an indication of the time required for execution of the current segment, the annotation item indicates the structure of the following segment that can be used to ascertain how complex the word is to plan (Dworzynski & Howell, 2004; Howell & Au-Yeung, 1995a; Howell, Au-Yeung & Sackin, 2000). These two factors can be examined jointly to determine whether they lead to fluency breakdown.

The software provided by SFS includes many of the same facilities as PRAAT. SFS can display selected analyses of the same stretch of speech aligned in time. Appendix A of this article gives details of how to access an extensive SFS database (i.e. the data on speakers who stammer). SFS was developed on a Unix system, and certain advanced software features require use of the Unix command line interpreter (such as those in Appendix E). The SFS Home Page can be found at http://www.phon.ucl.ac.uk/resource/sfs/

Although our work employs SFS extensively, other users may prefer to work with one of these other systems if they have specific needs of the facilities provided. Basic versions of the files have been supplied that allow users of these other systems to get started. For CHILDES users, files have been converted according to TEXTIN (MacWhinney, 1995, p.158), and links to WAV files provided. Other information that could be added to the CHAT files (speaker, age etc.) are available in the Access file described in Appendix A. The files have also been prepared as PRAAT TextGrids (and WAV files) at Paul Boersma's request. He intends to inform his email list of PRAAT users about the possibility of analysing and reporting results on these data. It is also recognised that some users may just need to listen to the files or read the transcriptions. The transcriptions can be examined with current word processing packages. Users who do not intend to carry out acoustic analyses probably do not want hi-fidelity WAV files as these are cumbersome to access. For this reason, MP3 files have been made available which are much shorter files. Though there is some data loss, the audio files are still of good quality. Most of the remainder of this article refers to the SFS versions of the files and associated analysis software.

Access to UCL Psychology Department Speech Group's data files. This article is intended to set the ball rolling to stimulate research in this area. Part of UCL's data on speakers who stammer will be made available. We are not just going to supply the audio data, but also, where available, orthographic and phonetic transcriptions. Some of the latter are also aligned against the audio records.

Appendix A indicates how the data in the various formats can be accessed. The data will also be distributed in an alternative medium that some users may find more convenient. CDs have been made of different subsets of the data set (described in Appendix A). The listening center at UCL's Phonetics and Linguistics Department holds copies of these CDs (the data cannot be customized for individual clients so if you need selections of data appearing on two or more CDs, you will have to purchase the CDs concerned). CDs can be purchased for a cost of around £10 each (including p&p) for those who prefer their data in this format. To comply with the European Union's data protection act, we have ensured that speakers cannot be identified from the recordings.

Transcription of spontaneous speech is a difficult task and it is unlikely to lead to 100% agreement between transcribers, and the transcription procedure has been designed to be more detailed at some points in utterances (around dysfluencies) than others (the rest of the speech). The reliability estimates that have been made for a selection of the transcriptions indicate that there is satisfactory agreement between trained transcribers (Howell, Au-Yeung & Sackin, 1999, 2000). However, this is the first time these transcriptions have been open to public scrutiny and although we have been rigorous in preparing the transcriptions, there will inevitably be some errors. Users should notify by email to psychol-stammer@ucl.ac.uk any errors they locate so that these can be corrected in subsequent release versions of the data. This 'public' correction procedure is preferred to one where audio files are not available, so errors are not visible. In addition, the recordings where there are audio files alone can be used by anyone who wishes to start from scratch (using the current or any other transcription scheme). We consider it imperative that new and improved transcriptions should be made available for scrutiny in the same manner as those we have provided.

Access to UCL's Phonetics Department's SFS software. SFS can be obtained from http://www.phon.ucl.ac.uk/ under Research Resources.

We do not have the resources to offer software support. We believe that providing these facilities offers a forum for scientific collaboration and exchange of ideas, which is the ethos behind *Stammering Research*. Copyright to the data is held by Howell and copyright to the software is held by Huckvale. The data and software are freely available to anyone for research and teaching purposes. If the data and/or software are used in publications, theses etc., users have to a) notify Howell (p.howell@ucl.ac.uk), b) acknowledge the source in any publication by referencing this article, c) include an acknowledgement that data collection was supported by the Wellcome Trust.

Outlet. It is intended that publications reporting analyses of these data will appear in *Stammering Research*. The prospect of extending research and assisting beginners to get started in research was made possible with the advent of Internet publications. In the main, the Internet has failed to deliver these possibilities to date because, where e-journals have appeared, they have usually been electronic versions of the printed journals that were available previously and have not provided access to data sources. *Stammering Research* welcomes submissions of articles for consideration that report analyses of these data, comparison between these data and previously published findings and so on. Submissions are invited at any time. There are no restrictions about what these analyses can be nor who may submit their work: Acoustic, articulatory, phonetic, phonological, prosodic and syntactic analyses would all be appropriate. The reports could also cover type/token, qualitative, transactional analyses. As implied, they can be compared with fluent speech or with samples of speech from people with other disorders or just analyses reporting on characteristics of stammered speech. Authors should be prepared to return analyses and scripts back to the archive (email submissions to psychol-stammer@ucl.ac.uk).

The SFS system can also be used for preparing material for perceptual tests. For instance, the software and the data could be used to replicate the classic Kully and Boberg (1988) study that showed that interclinic agreement in the identification of fluent and stammered syllables was poor. They can also be used to check some of the claims made by Cordes-Bothe and Ingham in support of time interval analysis as a means of assessing stammered speech. In this connection, it would be particularly useful to have TI sections of these freely-available materials judged by some members of these authors' expert panel, as the judgments of this panel have previously been used as benchmarks for other data about which intervals are, and are not, stammered.

It is hoped that these data will be of some lasting value to the research community (in the areas of stammering research and speech in general). In order to gauge whether there is a call for a facility like this,

we have prepared a limited selection of our data set at present (more will follow if this proves popular). As stated above, a section of *Stammering Research* has been set up which is devoted to analyses that include (though is not necessarily restricted to) these data.

3. Description of the data

A complete description of the UCL archive of data from speakers who stammer is given and then, the subset in the initial release is described. The complete data set currently includes 249 speakers who are categorized into five classes depending on the range of ages over which they have been recorded. There is also a holding class for young children we are still seeing but cannot project how long they will be available for recording. The data within each of these classes have undergone different amounts of preparation levels. This document describes the version one release of data from the first class where recordings are only available over a limited age range and where there is no possibility of obtaining more recordings. Samples are a) available as audio alone (as SFS files), b) with orthographic transcriptions (separate TXT file), c) with phonetic transcriptions (again separate TXT files) and d) where phonetic transcriptions are aligned against audio waveforms (available as SFS files). (CHILDES and PRAAT versions of the files are also available.) The alignment step under d) has a final check at the point where the transcriptions are aligned against the audio waveforms so these represent our highest level of data preparation. A full description of all classes of data we hold and current level of preparation is given below. The procedure for revising data in later revisions (corrections and generally useful ancillary analyses) is to send these in (as indicated in section 2). Depending on demand, other data classes will be released in phases as work is completed.

We record all speakers who stammer who volunteer. For our current project work, we are particularly interested in speakers in the age range eight to teenage. Pre teen speakers who stammer have a good chance of recovery. Consequently, we wish to follow up children who stammer and controls over this period, examine their speech and see whether different paths of fluency development are followed by those children who persist and those who recover. Ideally we want a minimum of three samples in this period (one in the age range 8-10 years, one between 10 and 12 years and one at teenage). We have complete sets of such recordings for 24 of our speakers.

Class 1, includes speakers who were either a) older than the maximum age of our target group when they were first seen (i.e. only seen after they have reached teenage), b) speakers who are in the target age ranges but who were only available at one target age because they live too far away from the laboratory or c) speakers who were in the required age range but with whom we have lost contact (most often because they have moved home and have not notified us of their new address and telephone number). Permission for data release cannot be obtained for speakers in class c). c) represents attrition of the sample, though there is no reason to suppose that this affects those speakers who persist differentially relative to those who recover. Class 2 consists of speakers who have not reached teenage who have been recorded at all target ages they have passed through that we are continuing to see (this class includes children who are under 8 years). Class 3 and 4 have not provided data at one of the first two target ages but attended at the third target age (i.e. they have reached teenage). For class 3, recordings are available at 8 and teenage, but not age 10-12 (often because the recording sessions clashed with school or family obligations and could not be rescheduled). For class 4, we have recordings for 10-12 and teenage. The lack of recordings at age 8-10 reflects the fact that these children were not seen at clinic until they were aged 10+. Class 5 are participants for whom we have at least three recordings at the designated ages, and we have continued to see most of these beyond the stipulated upper age. Many have supplied other forms of data. Appendix A describes an ACCESS file that is included in the data directory, which gives demographic information about the speakers.

Table 1 gives an indication of what is available and in what form. Not all data can be released at present (we have ethics permission, but are still waiting written consent by individuals or by clinics). Also, there are some data where the audio quality is not good enough for release. The numerator in each cell indicates what is being released and the denominator the total available. Thus 41/158 in the participants column indicates that data from 41 participants out of a total of 158 are being released.

Table 1. Summary of recordings available on UCL Department of Psychology Speech Group's data archive. Recordings are indicated according to class (see text) and type of file available.

	No of participants	No of files	No where orth avail	No where phon avail	No where phon aligned against audio
Class 1 –release 1	41/158	95/426	19/42	17/34	11/11
Class 2 - speakers who have not reached teenage, who have been recorded at all target ages they have passed through that we are continuing to see plus children who are < 8 years we are following up	7/ 21	13/57	5/7	1/1	2/2
Class 3 – available at 8-10yrs and 12yrs+	1/5	3/21	0/14	0/13	0/0
Class 4 – available at 10-12yrs and 12yrs+	7/41	20/184	6/57	6/54	0/1
Class 5 – recordings at the 3 target ages	5/24	7/142	1/63	0/44	3/69
Totals	249 61	830 138	183 31	147 24	82 16

4. Some uses for the data including illustrations of applications of SFS tools and concepts for assessing stammered speech

Data similar to those made available in Appendix A have been used to investigate a range of questions about stammering from many different perspectives. A comprehensive list of all studies conducted is beyond the scope of this article. Studies conducted by the UCL group range from acoustic analysis of articulatory features associated with stammering, to pragmatic analyses of speakers who stammer in conversation with others. At the acoustic level, the UCL group has examined how the phonation source operates in people who stammer (Howell, 1995; Howell & Williams, 1988, 1992; Howell & Young, 1990), whether the vowel in a series of repetitions is neutralized using formant frequency analysis methods similar to those described in Appendix D (Howell & Vause, 1986) and speech rate has been measured from digitized oscillograms (Howell, Au-Yeung & Pilgrim, 1999; Howell & Sackin, 2000). PRAAT offers acoustic analysis software that could extend our understanding of what happens to the voice when fluency breaks down. Phonetic and phonological analyses have been performed to assess whether these factors are implicated in stammering (Dworzynski & Howell, 2004; Dworzynski, Howell & Natke, 2003; Howell & Au-Yeung, 1995a; Howell, Au-Yeung & Sackin, 2000). The change in pattern of stammering over development has been examined within prosodically-defined units in a variety of languages (Au-Yeung, Vallejo Gomez & Howell, 2003; Dworzynski, Howell, Au-Yeung & Rommel, 2004; Howell, Au-Yeung & Sackin, 1999) and different ways of defining these units (based on lexical or metrical properties) have been investigated for Spanish (Howell, in press). Various forms of syntactic analysis have been performed to establish whether syntactically complex utterances are more likely to be stammered than simpler ones (Howell & Au-Yeung, 1995b; Kadi-Hanifi & Howell, 1992). A pragmatic factor that has been examined is whether the speech of the interlocutor affects the speech of the person who stammers (Howell, Kapoor & Rustin, 1997). CHILDES offers a variety of techniques that extend the possibilities of examining other high order effects on stammering (including pragmatic ones). There is considerable scope for further phonetic, phonological, prosodic, syntactic and pragmatic analysis of these data and some suggestions follow.

Suggested studies

Perception of stutterings. The data that are supplied can be used for assessing the effect of different perceptual procedures on stammering assessment, for training therapists/pathologists on stammering assessments, for showing how heterogeneous stammering patterns can be within and across age groups. The materials could also be used to replicate the classic, but somewhat dated, Kully and Boberg (1988)

study that showed judgements about the same sample of speech is judged differently by different clinics.

Studies on speech control in speakers who stutter. Some basic familiarity with acoustic phonetics is assumed to understand this section (for those requiring a refresher, see Ladefoged, 1975). The stammered sequence "kuh, kuh, Katy" contains a different sounding vowel ("uh" or as it is known more precisely "schwa"). Van Riper argued that a speaker who produces such a sequence had selected the wrong vowel at the start of this sequence and detected this by listening the sound of his or her voice (called feedback monitoring). As the speaker cannot produce "Katy" when the incorrect ("schwa") vowel has been inserted, the speaker interrupts speech and tries again. Howell and Vause (1986) argued that the vowel in a sequence of repetitions might sound like schwa because the vowels are short and low in amplitude (by analogy with vowel reduction that occurs in rapidly spoken, or casual, speech where the vowels also sound like schwa even when some other vowel is intended). They tested this hypothesis by acoustic analyses that compared the vowels in a sequence of repetitions with the vowel after fluent release (Howell and Vause also conducted perceptual tests, see the preceding section, which are not discussed here). They found that the formants of the vowels in sequences of repetitions and after fluent release occurred at the appropriate frequencies (suggesting that the vowel in the sequence of repetitions had been correctly articulated). Thus, they concluded that van Riper's feedback monitoring account of part word repetitions was not correct. The recordings of speaker 210 (at age 11 years 3 months) can be used to check Howell and Vause's finding. At 20.4s, the speaker appears to say "guh-go". The values of F1, F2 and F3 are similar in the "uh" and "o" sections indicating the vowels are similar (as Howell & Vause, 1986 reported).

Van Riper's argument could be applied to consonants that are prolonged. Prolonged /s/s. sound canonically like /s/. However there are different forms of /s/ that sound different which depend on what vowel follows. So, for example, an /s/ before an /i/ vowel sounds clearly different to an /s/ before an /u/ vowel. A possible explanation of /s/-prolongation could be that the wrong form of /s/ was selected and produced and when the speaker detected this, the transition to the following vowel could not be made leading to the speaker prolonging the /s/. This can be tested by seeing whether the /s/ in words that have an /i/ following is acoustically identical when prolonged compared with when it is spoken fluently (in the same way this was done when comparing the vowels in a sequence of part word repetitions to the intended vowel at fluent release in the previous study). Speaker 61 at age 14 years 8 months produced two prolonged /s/s before an /i/ vowel – one at the beginning of the word "CD" at 47 s and one at the beginning of "CCF" at 115 s. Though there are no fluent /s/s before /i/ in this recording, a recording was made a month later (14 years 9 months) in which the speaker says 'CD' twice (both times fluently) (these appear at 112.2 and 116 s in this file). Oscillograms, spectrograms and cross sections were taken of the fluent and dysfluent /s/s. The main feature is that the spectra peak at around 5kHz and this applies to fluent and dysfluent forms of /s/ before /i/. Based on the acoustic similarity and informal listening to these examples, speakers appears to be articulating the form of /s/ that would permit coarticulation with the intended vowel that follows. Thus, an account of prolongation based on selection of the inappropriate allophone of /s/ does not seem correct.

So far only continuant phones (vowels and fricatives) have been discussed. These are produced with articulatory positions that do not change over time. It is possible that speakers who stammer have problems in controlling speech timing which would be reflected in phones that require changes in articulation over the time of their production. Plosive stop consonants are one class of sounds where such timing problems might be manifest. Plosives start with a short period of broad band energy that marks sound onset (the burst). The plosives can be divided into voiced (/b, d, g/) and voiceless (/p, t, k/) forms where corresponding pairs (e.g. /b/and /p/) have the same place of articulation. The differences in voicing arise because speakers control the timing of articulatory gestures in distinct ways for these two classes of plosives. After burst onset, vocal fold vibration starts almost immediately for voiced plosive (voicing gives rise to the pitch epoch markers mentioned in the pitch synchronous analysis part of section 3 of Appendix D which appear as striations in broad band spectrograms). In contrast, voiceless plosives have a period, after the burst, during which the phone is aspirated before voicing starts. The time between burst onset and onset of voicing can be used as a simple measure of voice onset time (VOT) that characterizes the difference between voiced (short VOT) and voiceless (long VOT) plosives. If a speaker who stammers has problems initiating voicing in time, this would be reflected in longer VOTs for /d/s than /t/s and make /d/s sound something like /t/s.

Some speakers who stammer appear to have problems initiating voicing so this should be reflected in the VOT measure. Speaker 1100 shows this characteristic. For instance, 367 s into his audio file he shows multi part word syllable repetitions on the word 'David' (prior to 'Hockney') which sound devoiced (i.e. the /d/s sounds like /t/s). A /d/ realized as /t/ should have a longer VOT (close to a voiceless plosives) than

that of a true /d/. Acoustic analysis supports this notion, as you will see if you measure the VOT of the /d/s in the part word repetitions of the attempts at 'David' (prior to 'Hockney') and compare them with the VOT of /d/ in a fluent word (e.g. the "don't" that occurs at 80.7 s). You should also listen to the voiced sounds to confirm that they appear to be devoiced (i.e. the /d/ sounds like /t/).

Performance of HMM automatic dysfluency recognizer. Like fluent speech corpora, the data may also provide material for training automatic speech recognizers (Howell, Hamilton & Kyriacopoulos, 1986; Howell, Sackin & Glenn, 1997a, b; Noth, Niemann, Haderlein, Decher, Eysholdt, Rosanowski, & Wittenberg, submitted). Some hidden Markov model (HMM) utilities that have been used to construct a dysfluency recognizer are described in Appendix E. Performance of this recognizer on 5s intervals that experts agreed about for six test samples (0030_17y9m.1, 0061_14y8m.1, 0078_16y5m.1, 0095_7y7m.1, 0098_10y6m.1, 0138_13y3m.1, 0210_11y3m.1, 0234_9y9m.1) correctly classified 60% of intervals as stuttered/fluent. Though above chance, this is not particularly impressive. It does set a benchmark against which better dysfluency recognizers can be developed and assessed. One obvious improvement would be to be more selective when training the phone models (in the benchmark version, these were obtained from fluent speakers, not speakers who stammer). In addition to providing training material specifically for automating dysfluency counts, these data could be used more generally to test the robustness of recognizers developed for fluent speech. As it is claimed that recognizers perform at high levels when material is fluent, to what extent do failures in these algorithms coincide with stuttered dysfluencies? These are just some of the topics that the group at UCL are addressing and doubtless there are numerous other topics that the data and software will be helpful in investigating.

The JSRU coding scheme is presented in Appendix B. Modifications for the application of this scheme for dealing with stammered speech and codes for properties that are important for studying stammering are given in section 6 of Appendix C. The material that has been prepared according to this scheme can be used for analyses similar to those described in the studies cited above. A tutorial on manipulating transcriptions in SFS (with a focus on aligning them against the audio waveforms) is given in Appendix C. As stated earlier, SFS may prove particularly useful when researchers want to visualize how disparate sources of information (e.g. duration and characterizations of phonetic difficulty) on different segmental units interact and lead to dysfluency. The aligned display convention should also prove useful when comparing the results of different analysis methods (such as those used for syntactic characterization of these materials) applied to the same stretch of data. A second SFS tutorial (Appendix D) covers basic aspects to do with acoustic analysis of speech that have been employed in some of the studies cited at the start of this section. Appendix E describes tools that could be used for developng HMM recognizers for dysfluent speech.

Acknowledgement

This research was supported by the Wellcome Trust.

References

Au-Yeung, J., Vallejo Gomez, I., & Howell, P. (2003). Exchange of disfluency from function words to content words with age in Spanish speakers who stutter. *Journal of Speech, Language and Hearing Research*, **46**, 754-765.

Dworzynski, K., & Howell, P. (2004). Predicting stuttering from phonetic complexity in German. *Journal of Fluency Disorders*, **29**, 149-173.

Dworzynski, K., Howell, P., Au-Yeung, J., & Rommel, D. (2004). Stuttering on function and content words across age groups of German speakers who stutter. *Journal of Multilingual Communication Disorders*, **2**, 81-101

Dworzynski, K., Howell, P., & Natke, U. (2003). Predicting stuttering from linguistic factors for German speakers in two age groups. *Journal of Fluency Disorders*, **28**, 95-113.

Howell, P. (1995). The acoustic properties of stuttered speech. In *Proceedings of the First World Congress on Fluency Disorders, Vol II*. Pp. 48-50. C. W. Starkweather & H. F. M. Peters (Eds). Nijmegen: Nijmegen University Press.

Howell, P. (2002). The EXPLAN theory of fluency control applied to the treatment of stuttering by altered feedback and operant procedures. In *Pathology and therapy of speech disorders*. Pp. 95-118. E. Fava (Ed.). Amsterdam: John Benjamins.

Howell, P. (2004). Assessment of some contemporary theories of stuttering that apply to spontaneous speech. *Contemporary Issues in Communicative Sciences and Disorders*, **39**, 122-139.

Howell, P. (in press). Comparison of two ways of defining phonological words for assessing stuttering

pattern changes with age in Spanish speakers who stutter. *Journal of Multilingual Communication Disorders.*

Howell, P., & Au-Yeung, J. (1995a). The association between stuttering, Brown's factors and phonological categories in child stutterers ranging in age between 2 and 12 years. *Journal of Fluency Disorders*, **20**, 331-344.

Howell, P., & Au-Yeung, J. (1995b). Syntactic determinants of stuttering in the spontaneous speech of normally fluent and stuttering children. *Journal of Fluency Disorders*, **20**, 317-330.

Howell, P. & Au-Yeung, J. (2002). The EXPLAN theory of fluency control and the diagnosis of stuttering. In *Pathology and therapy of speech disorders*. Pp. 75-94. E. Fava (Ed.). Amsterdam: John Benjamins.

Howell, P., Au-Yeung, J., & Pilgrim, L. (1999). Utterance rate and linguistic properties as determinants of speech dysfluency in children who stutter: *Journal of the Acoustical Society of America*, **105**, 481-490.

Howell, P., Au-Yeung, J., & Sackin, S. (1999). Exchange of stuttering from function words to content words with age. *Journal of Speech, Language and Hearing Research*, **42**, 345-354.

Howell, P., Au-Yeung, J., & Sackin. S. (2000). Internal structure of content words leading to lifespan differences in phonological difficulty in stuttering. *Journal of Fluency Disorders*, **25**, 1-20.

Howell, P., Hamilton, A., & Kyriacopoulos, A. (1986). Automatic detection of repetitions and prolongations in stuttered speech. In *Speech Input/Output: Techniques and Applications.* Pp. 252-256. London: IEE Publications.

Howell, P. Kapoor, A., & Rustin L. (1997). The effects of formal and casual interview styles on stuttering incidence. In *Speech Production: Motor Control, Brain Research and Fluency Disorders.* Pp. 515-520. W. Hulstijn, H. F. M. Peters & P. H. H. M. van Lieshout (Eds.). Amsterdam: Elsevier.

Howell, P., & Sackin, S. (2000). Speech rate manipulation and its effects on fluency reversal in children who stutter. *Journal of Developmental and Physical Disabilities*, **12**, 291-315.

Howell, P., Sackin, S., & Glenn, K. (1997a). Development of a two-stage procedure for the automatic recognition of dysfluencies in the speech of children who stutter: I. Psychometric procedures appropriate for selection of training material for lexical dysfluency classifiers. *Journal of Speech, Language and Hearing Research*, **40**, 1073-1084

Howell, P., Sackin, S., & Glenn, K. (1997b). Development of a two-stage procedure for the automatic recognition of dysfluencies in the speech of children who stutter: II. ANN recognition of repetitions and prolongations with supplied word segment markers *Journal of Speech, Language and Hearing Research*, **40**, 1085-1096.

Howell, P., & Vause, L. (1986). Acoustic analysis and perception of vowels in stuttered speech. *Journal of the Acoustical Society of America*, **79**, 1571-1579.

Howell, P. & Williams, M. (1988). The contribution of the excitatory source to the perception of neutral vowels in stuttered speech. *Journal of the Acoustical Society of America*, **84**, 80-89.

Howell, P., & Williams, M. (1992). Acoustic analysis and perception of vowels in children's and teenagers' stuttered speech. *Journal of the Acoustical Society of America*, **91**, 1697-1706.

Howell, P., & Young, K. (1990). Analysis of periodic and aperiodic components during fluent and dysfluent phases of child and adult stutterers' speech. *Phonetica*, **47**, 238-243.

Kadi-Hanifi, K., & Howell, P. (1992). Syntactic analysis of the spontaneous speech of normally fluent and stuttering children. *Journal of Fluency Disorders*, **17**, 151-170.

Kully, D., & Boberg, E. (1988). An investigation of interclinic agreement in the identification of fluent and stuttered syllables. *Journal of Fluency Disorders*, **13**, 309-318.

Ladefoged, P. (1975). *A course in Phonetics*. New York: Harcourt, Brace, Jovanovich.

MacWhinney, B. (1995). *The CHILDES project*. Hillsdale NJ: Lawrence Erlbaum.

Noth, E., Niemann, H., Haderlein, T., Decher, M., Eysholdt, U., Rosanowski, F., & Wittenberg, T. (submitted). Automatic stuttering recognition using Hidden Markov models.

Rosen, S., & Howell, P. (1991). *Signals and Systems for Speech and Hearing*. London and San Diego: Academic Press.

Appendix A – Data description

This Appendix contains an indication where UCL's archive of recordings of speech from speakers who stutter are located and how they can be accessed.

1. Ancillary information about speaker and audio files

Some information we have about the speakers is given in the ACCESS file 'information table'. People who are familiar with ACCESS can use this to search and select the file for the recordings they want (by gender, age etc.). Each row of the table corresponds with one of the 138 files in the directory. The filename is given in the column labeled 'file id'. This starts with a code for gender (M/F), then has underscore and a four figure code to identify speaker, UCL group, underscore class number (as given in Table 1), underscore age NNvNm, followed by undersore and a number indicating which recording in that month this represents. The second column gives gender (M for male and F for female). The age of the speaker at the recording session is given in months in column three. The fourth column indicates where the recording was made (clinic, UCL or home). Recording conditions are indicated in column five as either as quiet room (QR) or sound-treated room (STR). The sixth column gives the type of therapy received (either including the family, F, or holistic, H). An indication is given about the time (in months) between the recordings and therapy in the seventh column. The eighth column indicates whether the speaker had any history of hearing problems. The ninth and tenth columns indicate whether the speaker had a history of language problems or special educational needs respectively, and the eleventh, twelfth and thirteenth columns indicate whether any manual transcripts are available (orthographic, phonetic and, for the files that have phonetic transcriptions, those which are time aligned, respectivly).

2. Formats available

For CHILDES. Transcriptions are prepared according to TEXTIN (MacWhinney, 1995) and corresponding audio files are also available in WAV format. The WAV files have been linked to the CHAT files. CLAN has options that allow PRAAT programs to process the associated WAV files. Other information that CHAT files include in their headers are available in the ACCESS file described above.

CHAT files + WAV = CHILDES

For PRAAT. The transcriptions have been converted to PRAAT TextGrids. PRAAT provides acoustic analysis facilities for dealing with WAV files that are also available in the directory.) The way PRAAT works is described on the PRAAT website (*http://www.praat.org*).

TextGrids + WAV = PRAAT

For SFS. There is an SFS file corresponding to each of the recordings. Most of these just contain the audio file. The remaining ones have phonetic transcriptions that have been aligned manually against the audio file. Separate orthographic and phonetic transcriptions are available for some of the files as text files. The phonetic transcriptions are in JSRU format (see Appendix B), but it should be easy to translate them to other phonetic formats. The audio waveforms and aligned transcriptions can be displayed and manipulated using the whole rang of SFS utilities. The separate orthographic and phonetic files can be used in the transcription exercises indicated in Appendix C.

Other format. All data are also available as MP3 files.

The speech and ancillary data can be accessed at:

http://www.uclass.psychol.ucl.ac.uk

3. Description of discs

The dataare on eight CDs that follow the directory structure of the online Release 1 (August 2004) dataset.

Disc 1 contains all the available transcription data. It also contains an MS Access database and an HTML formatted page of the files in Release 1: flat orthographic and phonetic transcriptions and time aligned transcriptions in SFS Annotation, CHILDES CHAT and Praat TextGrid format.

Disc 2 contains all audio data in compressed mp3 format.

Discs 3-5 contain all audio data in SFS format and Disc 3 also contains the available SFS audio data with time aligned annotations and also those files used in the worked examples.

Discs 6-8 contain all the audio data in wav format.

Appendix B The JSRU alphabet

This Appendix gives the Joint Speech Research Unit (JSRU) alphabet.

1. The JSRU Alphabet

The JSRU (Joint Speech Research Unit) alphabet is a phonetic transcription system that was developed for text-to-speech synthesis. It has a machine readable format. The correspondence between 'qwerty' keyboard characters and phonetic symbols is given in Table B.1 for consonants and Table B.2 for vowels.

Table B.1. JSRU symbols and IPA equivalents for English consonants. IPA symbols are only given when they differ from the JSRU symbols.

JSRU	IPA	Sound info	Example (word initial)	Example (non-initial)
p	(same)	p sound	pen	Lip
b	(same)	b sound	bad	nib
t	(same)	t sound	tea	Light
d	(same)	d sound	do	Lad
k	(same)	k sound	cat	crack
g	(same)	g sound	got	Fog
ch		ch sound	chin	watch
j		j sound	June	village
f	(same)	f sound	food	off
v	(same)	v sound	voice	give
th		th sound	thin	Tenth
dh		dh sound	then	with
s	(same)	s sound	same	success
z	(same)	z sound	zoo	was
sh		sh sound	show	wash
zh		zh sound	genre	beige
h	(same)	h sound	happy	behave
m	(same)	m sound	man	swim

n	(same)	n sound	know	gone
ng		ng sound	N/A	sing
l	(same)	l sound	leg	girl
r		r sound	red	arrow
y		y sound	year	value
w	(same)	w sound	wet	quick
x		x sound	N/A	Loch (Scots)
gx		gx sound	got in Cockney	glottal stop

Table B.2. JSRU symbols and IPA equivalents for English vowels. IPA symbols are only given when they differ from the JSRU symbols.

JSRU	IPA	Sound info	Example
i		i sound	sit
o		o sound	got
oo		oo sound	put
aa		aa sound	hat
e		e sound	ten
U		u sound	cup
a		a sound	ago

ee		
	ee/ey sound	see
aw		
	aw sound	saw
uu		
	uu sound	too
ar		
	ar sound	arm
er		
	er sound	fur
ai	ai sound	page
ie		
	ie sound	eye
oi		
	oi sound	boy
oa		
	oa sound	home
ou		
	ou sound	now
ia		
	ia sound	beer
ei		
	ei sound	bare
ur	ur sound	tour

ɪ ɒ ʊ æ ɛ ʌ ə iː ɔː uː ɑː ɜː ɛɪ aɪ ɔɪ əʊ aʊ ɪə ɛə ʊə

Appendix C Orthographic and Phonetic Annotation with SFS

*The previous Appendix described the JSRU alphabet but did not indicate how the transcriptions can be linked with the audio signal. SFS provides an environment for representing audio and transcriptional information together. This Appendix provides a tutorial introduction to the use of SFS for the orthographic and phonetic transcription of a speech recording, including tools for automatic alignment of phonetic transcriptions to the signal. Some software is presented and described in this Appendix but programming experience is not assumed. You may not need to do all steps described in this Appendix. For instance, if you only intend to use the data supplied in Appendix A, the sections on acquiring the audio signal are not needed. This tutorial refers to versions 4.6 and later of SFS and appears in the documentation on the SFS website. Visit the SFS website to obtain your software (*http://www.phon.ucl.ac.uk/resource/sfs/).

1. Acquiring and Chunking the audio signal

Acquiring the signal

You can use the SFSWin program to record directly from the audio input signal on your computer. Only do this if you know that your audio input is of good quality, since many PCs have rather poor quality audio

inputs. In particular, microphone inputs on PCs are commonly very noisy.

To acquire a signal using SFSWin, choose File|New, then Item|Record. See Figure C.1.1. Choose a suitable sampling rate, at least 16000 samples/sec is recommended. It is usually not necessary to choose a rate faster than 22050 samples/sec for speech signals.

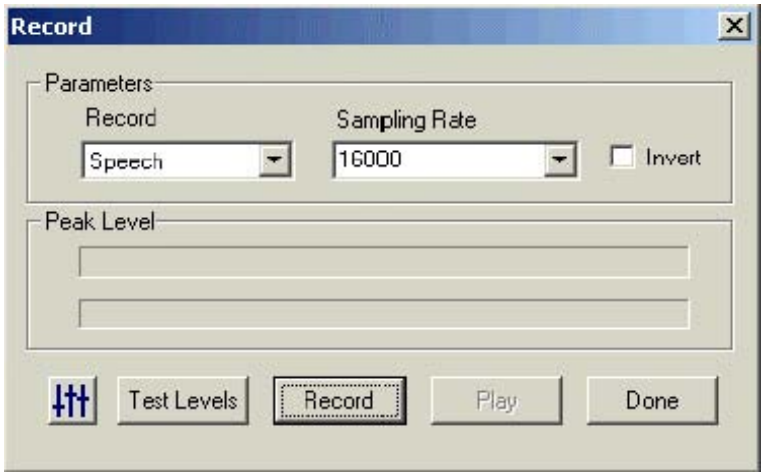


Figure C.1.1 - SFSWin record dialog

If you choose to acquire your recording into a file using some other program, or if it is already in an audio file, choose Item|Import|Speech rather than Item|Record to load the recording into SFS. If the file is recorded in plain PCM format in a WAV audio file, you can also just open the file with File|Open. In this latter case, you will be offered a choice to "Copy contents" or "Link to file" to the WAV file. See Figure C.1.2. If you choose copy, then the contents of the audio recording are copied into the SFS file. If you choose link, then the SFS file simply "points" to the WAV file so that it may be processed by SFS programs, but it is not copied (this means that if the WAV file is deleted or moved SFS will report an error).

Figure C.1.2 - SFSWin open WAV file dialog

Preparing the signal

If the audio recording has significant amounts of background noise, you may like to try to clean the recording using Tools|Speech|Process|Signal enhancement. See Figure C.1.3. The default setting is "100% spectral subtraction"; this subtracts 100% of the quietest spectral slice from every frame. This is a fairly conservative level of enhancement, and you can try values greater than 100% to get a more aggressive enhancement, but at the risk of introducing artefacts.

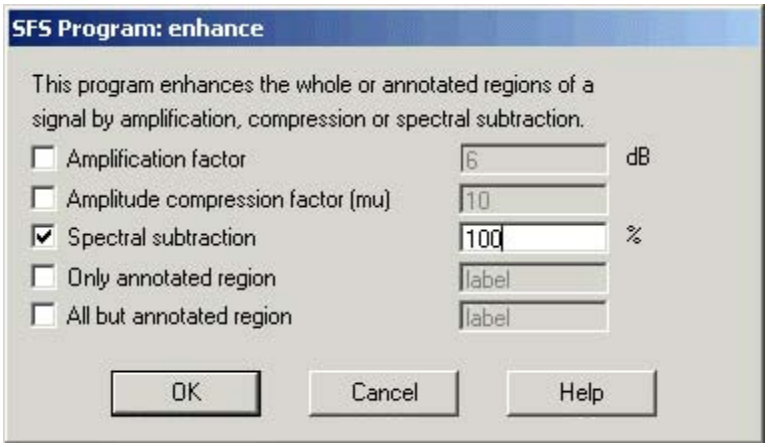


Figure C.1.3 - SFSWin enhancement dialog

It is also suggested at this stage that you standardise the level of the recording. You can do this with Tools|Speech|Process|Waveform preparation, choosing the option "Automatic gain control (16-bit)" (AGC). No noise reduction (spectral subtraction) nor AGC have been made on the recordings that can be accessed using the information in Appendix A.

Chunking the signal

If your audio recording is longer than a single sentence, you will almost certainly gain from first chunking the signal into regions of about one sentence in length. Chunking involves adding a set of annotations which delimit sections of the signal. The advantages of chunking include:

- it means that transcription is roughly aligned to the signal .
- it makes it easier to navigate around the signal.
- it improves the performance of automatic phonetic alignment.
- it allows the export of a "click-to-listen" web page using VoiScript (see below).
- it allows us to use SFS annotations to store the transcription, since SFS limits annotations to 250 characters.

An easy way to chunk the signal is to automatically detect pauses using the "npoint" program. This takes a speech signal as input and creates a set of annotations which mark the beginning and end of each region where someone is speaking. It is a simple and robust procedure based on energy in the signal. To use this, select the speech item and choose Tools|Speech|Annotate|Find multiple endpoints. See Figure C.1.4. If you know the number of spoken chunks in the file (it may be a recording of a list of words, for example), enter the number using the "Number of utterances to find" option, otherwise choose the "Auto count utterances" option. Put "chunk" (or similar) as the label stem for annotation.

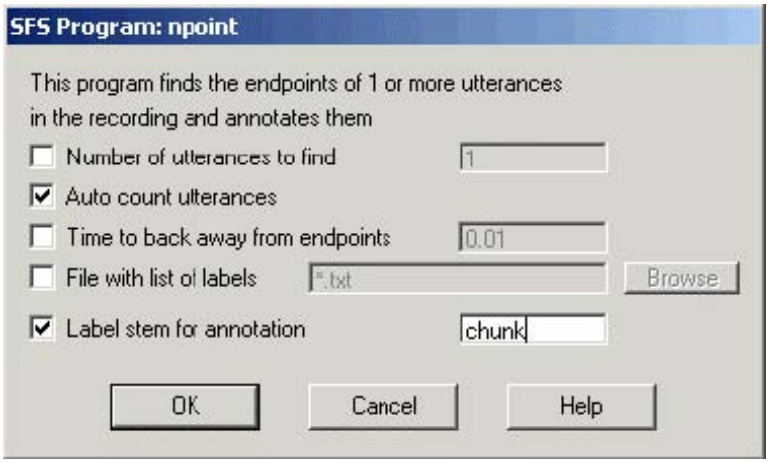


Figure C.1.4 - SFSWin find multiple endpoints dialog

If you view the results of the chunking you will see that each spoken region has been labeled with "chunk*dd*", while the pauses are labelled with "/". See Figure C.1.5 for the results of applying this procedure to one of the files provided.

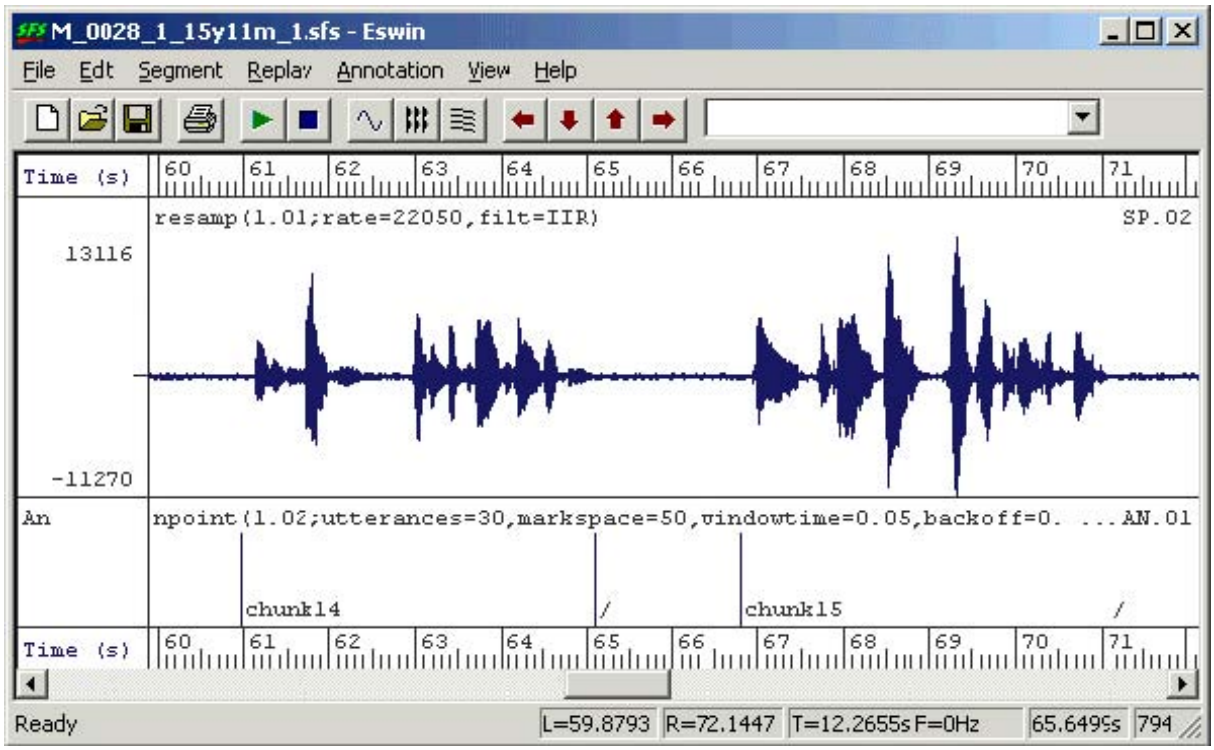


Figure C.1.5 - chunked signal

If the chunking has not worked properly, or if you want to chunk the signal by hand, you can use the manual annotation facility in Eswin. To do this, select the signal you want to annotate and choose Item|Display to start the eswin program. Then choose eswin menu option Annotation|Create/Edit Annotations, and enter either a set name of "chunks" to create a new set of annotations, or enter "endpoints" to edit the set of annotations produced by npoint.

When eswin is ready to edit annotations you will see a new region at the bottom of the screen where your annotations will appear. To add a new annotation, position the left cursor at the time where you want the annotation to appear. Then type in the annotation into the annotation box on the toolbar and press [RETURN]. The annotation should appear at the position of the cursor. See Figure C.1.6.

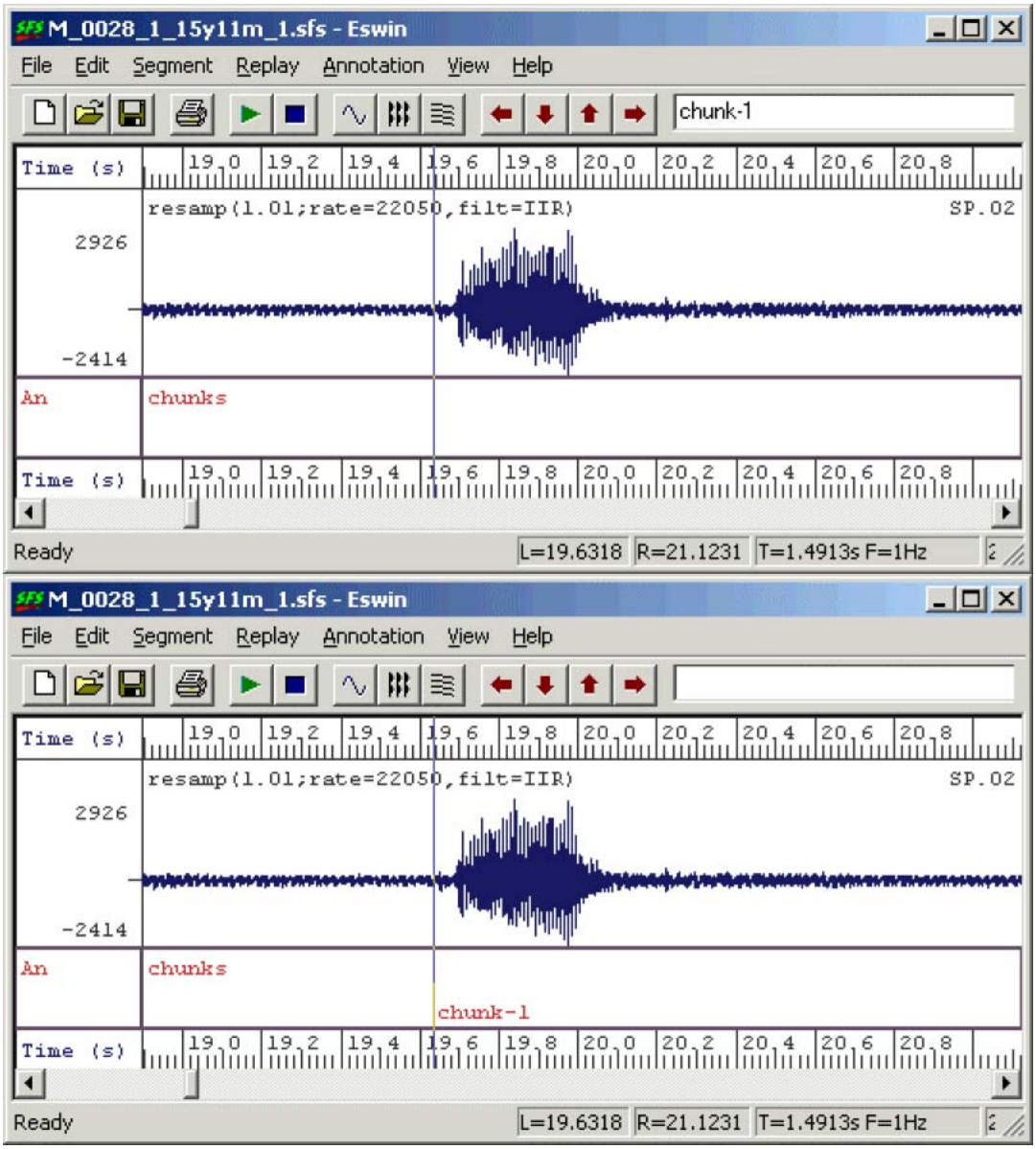


Figure C.1.6 - adding an annotation in eswin

To move an annotation with the mouse, position the mouse cursor on the annotation line within the bottom annotation box. You will see that the mouse cursor changes shape into a double-headed arrow. Press the left mouse button and drag the annotation left or right to its new location. This is also an easy way to correct chunk endpoints found automatically by npoint.

Finally, to hear if the chunking has worked properly, you can listen to the chunked recording using the SFS wordplay program. This program is not on the SFSWin menus, so to run it, choose Tools|Run program/script then enter "wordplay -SB" in the "Program/script name" box. See figure C.1.7. This will replay each chunk in turn, separating the chunks with a small beep.

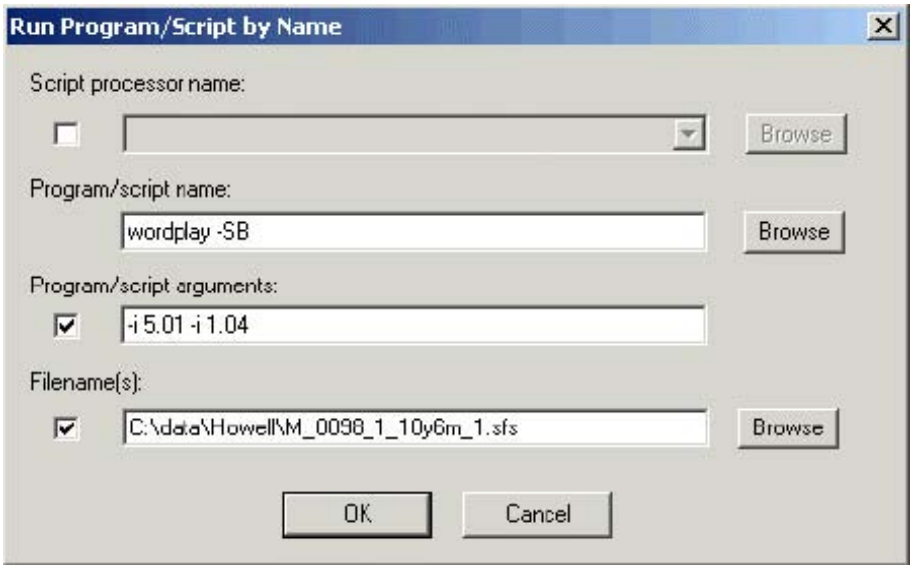


Figure C.1.7 - SFSWin run program dialog

2. Orthographic transcription

Entering orthographic transcription

Assuming that your recording has been chunked into sentence-sized regions, the process of orthographic transcription is now just the process of replacing the "chunk" labels with the real spoken text. The result will be a new annotation item in the file, but where each annotation contains the orthographic transcription of a chunk of signal. See Figure C.2.1.

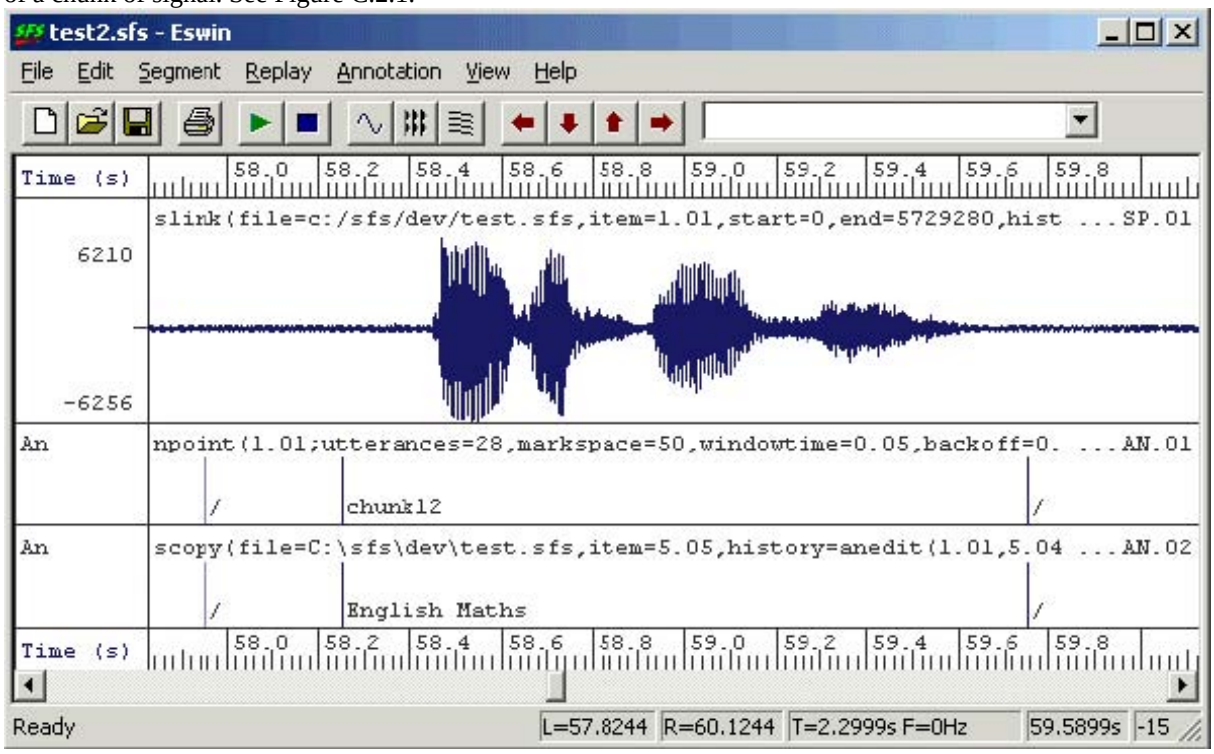


Figure C.2.1 - Speech chunks and orthographic transcription

You *can* edit annotation labels using the eswin display program, but it is not very easy - you have to overwrite each annotation label with the transcription. A much easier way is to use the anedit annotation

label editor program. This program allows you to listen to the individual annotated regions and to edit the labels of annotations without affecting their timing. To run anedit, select a speech item and the annotation item containing the chunks and choose Tools|Annotations|Edit Labels. Since you are mapping one set of annotations into another, change the "output" annotation type to "orthography". See Figure C.2.2.

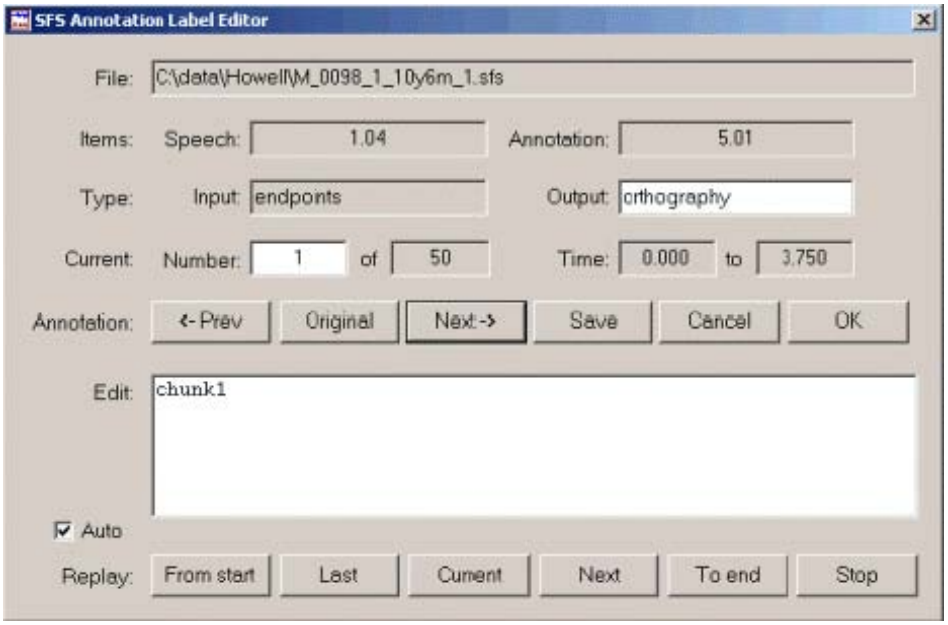


Figure 2.2 - anedit window

The row of buttons in the middle of the annotation editor window control the set of annotations:

<–Prev
Move to previous annotation in the list.
Original
Reload the original annotation label at this position.
Next–>
Move to next annotation in the list.
Save
Save the changes you've made so far to the SFS file.
Cancel
Forget annotation label changes and exit.
OK
Save annotation label changes and exit.

The row of buttons at the bottom of the annotation editor window control the replay of the speech signal:

From start
Replay from the start of the file to end of the current annotated region.
Last
Replay from the start of the last annotated region to the end of the current annotated region.
Current
Replay the current annotated region.
Next
Replay from the start of the current annotated region to the end of the next annotated region.
To end
Replay from the start of the current annotated region to the end of the file.
Stop
Stop any current replay.

The "Auto" replay feature causes the current annotated region to be replayed each time you change to a different annotation.

To use anedit for entering orthographic transcription, first check that the "Auto" replay feature is enabled and that you are positioned at the first chunk of speech. Replay this with the "Current" button, select and over-write the old label with the text that was spoken. Then press the [RETURN] key. Two things should happen: first you should move on to the next chunk in the file and second that chunk of signal should be replayed. You can now proceed through the file, entering a text transcription and pressing [RETURN] to move on to the next chunk. If you need to hear the signal again, use the buttons at the bottom of the screen. It is suggested that every so often you save your transcription back to the file with the "Save" button. This ensures you will not lose a lot of work should something go wrong.

One **word of warning**: at present SFS is limited to annotations that are less than 250 characters long. Anedit prevents you from entering longer labels. There is no limit to the *number* of labels however.

Conventions

It is worth thinking about some conventions for entering transcription. For example, should you start utterances with capital letters, or terminate them with full stops? Should you use punctuation? Should you use abbreviations and digits? Should you mark non-speech sounds like breath sounds, lip smacks or coughs?

Here is one convention that you might follow, which has the advantage that it is also maximally compatible with SFS tools.

- put all words except proper nouns in lower case.
- do not include any punctuation.
- spell out all abbreviations and numbers, i.e. "g. c. s. e." not "GCSE", "one hundred and two" not "102", "ten thirty" not "10:30".
- mark non-speech sounds in a special way, e.g. "[cough]".

For pause regions you can either choose to label these using a special symbol of your own (e.g. "[pause]"), or leave them annotated as "/", or label them with the SAMPA symbol for pause which is "...".

Making a clickable script

Once you have a chunked and transcribed recording you can distribute your transcription as a "clickable script" using the VoiScript program (available for free download from **http://www.phon.ucl.ac.uk/resource/voiscript/**). The VoiScript program will display your transcription and replay parts of it in response to mouse clicks on the transcription itself. This makes it a very convenient vehicle for others to listen to your recording and study your transcription.

VoiScript takes as input a WAV file of the audio recording and an HTML file containing the transcription coded as links to parts of the audio. Technical details can be found on the VoiScript web site. To save your recording as a WAV file, choose Tools|Speech|Export|Make WAV file, and enter a suitable folder and name for the file. The following SML script can be used to create a basic HTML file compatible with VoiScript:

```
/* anscript.sml - convert annotation item to VoiScript HTML file */

/* takes as input file.sfs and outputs HTML
        assuming audio is in file.wav */

main {
  string basename
  var    i,num

  i=index("\.",$filename);
  if (i) basename=$filename:1:I-1 else basename=$filename;

  print "<html><body><h1>",basename,"</h1>\n";

  num=numberof(".");
  for (I=1;i<=num;i=i+1) if (compare(matchn(".",i),"/")!=0) {
    print "<a name=chunk",i:1
    print " href='",basename,".wav#",timen(".",i):1:4
    print "#",(timen(".",i)+lengthn(".",i)):1:4,"'>"
    print matchn(".",i),"</a>\n"
  }

  print "</body></html>\n"
}
```

Copy and paste this script into a file "anscript.sml". Then select the annotation item you want to base the output on and choose Tools|Run SML script. Enter "anscript.sml" as the SML script filename and the name of the output HTML file as the output listing filename in the same directory as the WAV audio file.

If you now open the output HTML file within the VoiScript program, you will be able to read and replay parts of the transcription on demand, see Figure C.2.3.

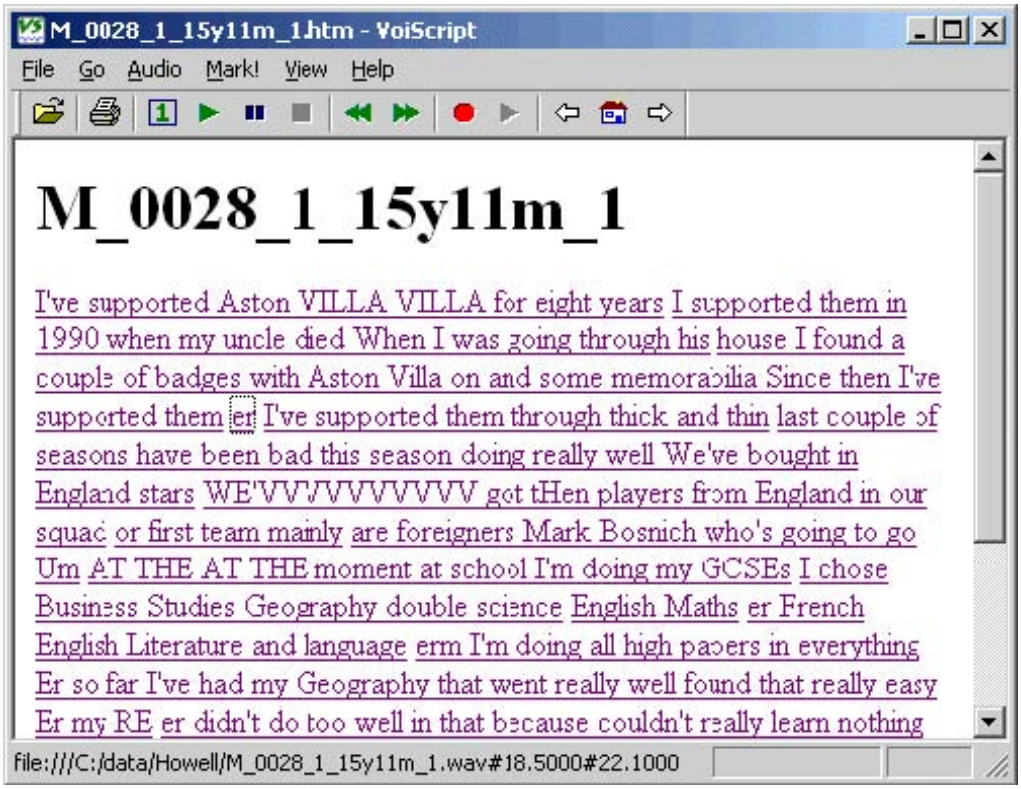


Figure C.2.3 - example VoiScript clickable script

3. Phonetic transcription

Spelling to sound

We now have a chunked orthographic transcription of our recording roughly aligned to the audio signal. The next stage is to translate the orthography for each chunk into a phonetic transcription. If we know the language, this is a largely mechanical procedure of looking up words in a dictionary. If the language is English, the mechanical part of the process can be performed by the antrans program.

The SFS program antrans performs the phonetic transcription of orthography using a built-in English pronunciation dictionary. The program takes orthographic annotations as input and produces transcribed annotations as output, in which only the content of the labels has been changed. See Figure C.3.1

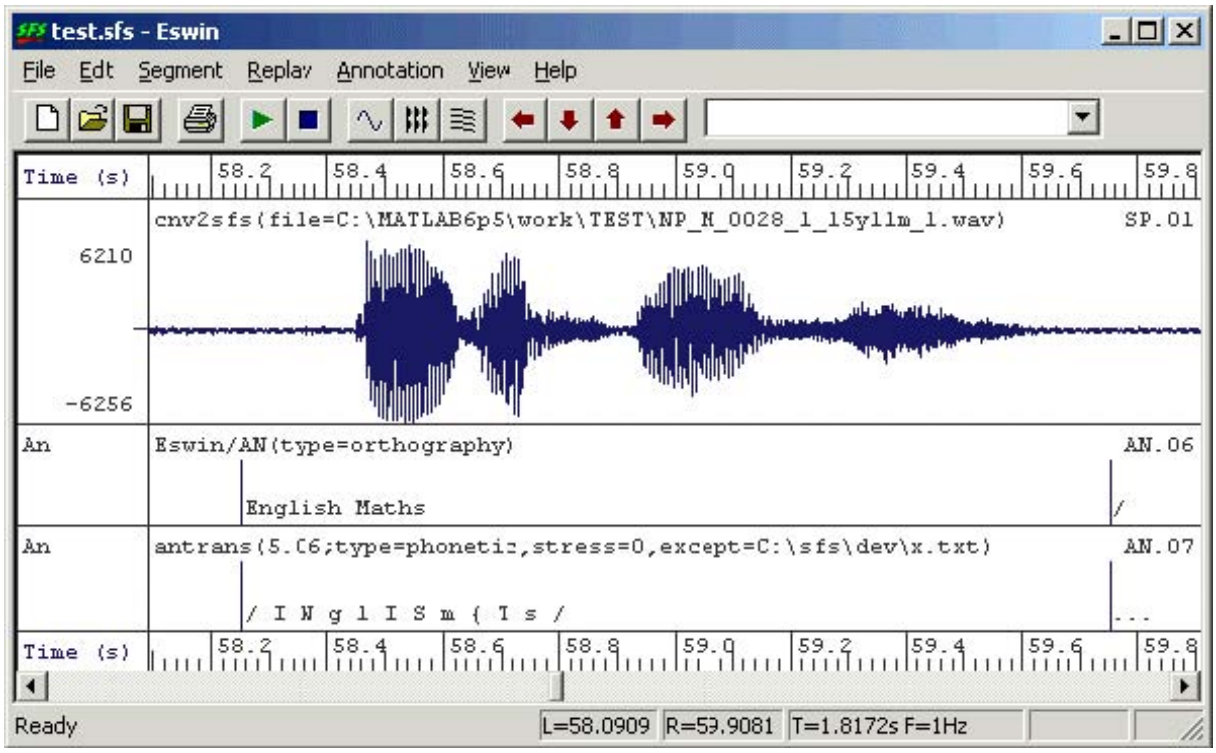


Figure C.3.1 - Transcribed annotations

It will almost certainly be the case that antrans will do an imperfect job in any real situation, since:

- it only produces a single pronunciation for each word, and that may not be the pronunciation used by the speaker;
- it may not know all the words used: although it has a large dictionary, it cannot know all names, places and abbreviations;
- it does not take into account any possible or actual contextual changes to pronunciation, such as assimilations and elisions;
- it can only guess that each chunk begins and ends in silence.

To run antrans, select the input annotation item and choose Tools|Annotation|Transcribe labels. The first time this is run, collect a list of words that antrans doesn't know by using the 'Missing word list' option, see Figure C.3.2. After the program has run, edit the word list (in Windows notepad for example) and add a transcription to each word, saving the resulting file as an exceptions list. This can then be incorporated in a second run of antrans (you can delete the output of the first run), see Figure C.3.3.

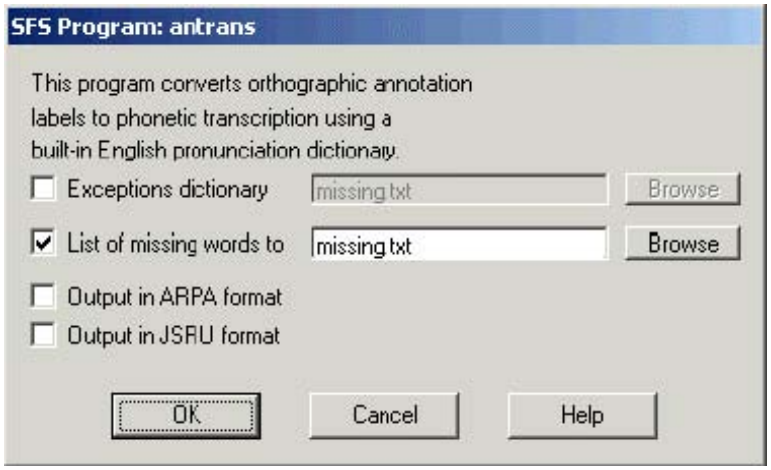


Figure C.3.2 - SFSWin Transcribe labels dialog (1)

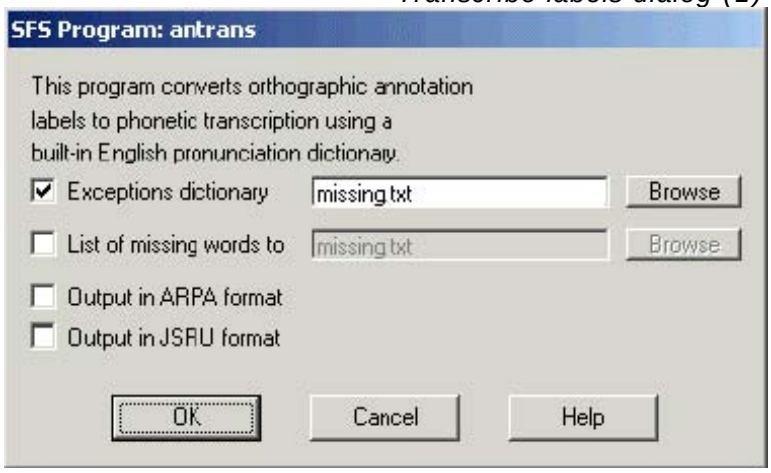


Figure C.3.3 - SFSWin Transcribe labels dialog (2)

The format of the exceptions file is as follows. It is a text file where each line is the pronunciation of a single word. A word is a sequence of printable characters that do not contain a space. The spelling of the word is followed by a TAB character, and then the transcription follows in SAMPA notation. It is usually not necessary to separate the SAMPA segment symbols with spaces, but it does not do any harm. Include stress symbols only if you intend to use them later. Here is an example:

```
1990    naInti:n naIntI
Bosnich bQznItS
MATHSSSSSSS     m{Ts
WE'VVVVVVVVVV   wi:v
```

A simple way to correct the transcription is to use the anedit program again, just as we did for entering orthographic transcription in section C.2.

Transcription systems

The SFS tools are designed to work with the SAMPA transcription system by default, but antrans can also use transcriptions in ARPA and JSRU systems. The table below gives a comparison of the symbol system as compared to the IPA.

IPA Keyword SAMPA ARPA JSRU	IPA Keyword SAMPA ARPA JSRU
P	ɪ
Pin	pit
P	I
P	ih
P	i
B	e
Bin	pet
B	e
B	eh
B	e
T	æ
Tin	pat
T	{
T	ae
T	aa
D	ɒ
Din	pot
D	Q
D	oh
D	o
K	ʌ
Kin	cut
K	V
K	ah
K	u
G	ʊ
Give	put
G	U
G	uh
G	oo
tʃ	ə
chin	another
tS	@
ch	ax
ch	a
dʒ	iː
gin	ease
dZ	i:
jh	iy
j	ee

In addition, the following symbols are used to mark stress and silence:

IPA	Description	SAMPA	ARPA	JSRU
ˈ	*primary stress*	"		"
ˌ	*secondary stress*	%		'
	Silence	/	sil	q
	Pause	…	sil	q

4. Aligning phonetic transcription

At this point we have a chunked phonetic transcription: each spoken chunk of the signal is annotated with a unit of phonetic transcription. The next stage is to break up the transcription into individual segment labels and roughly align the labels to the signal. See Figure C.4.1. A basic level of alignment can be performed by the SFS analign program.

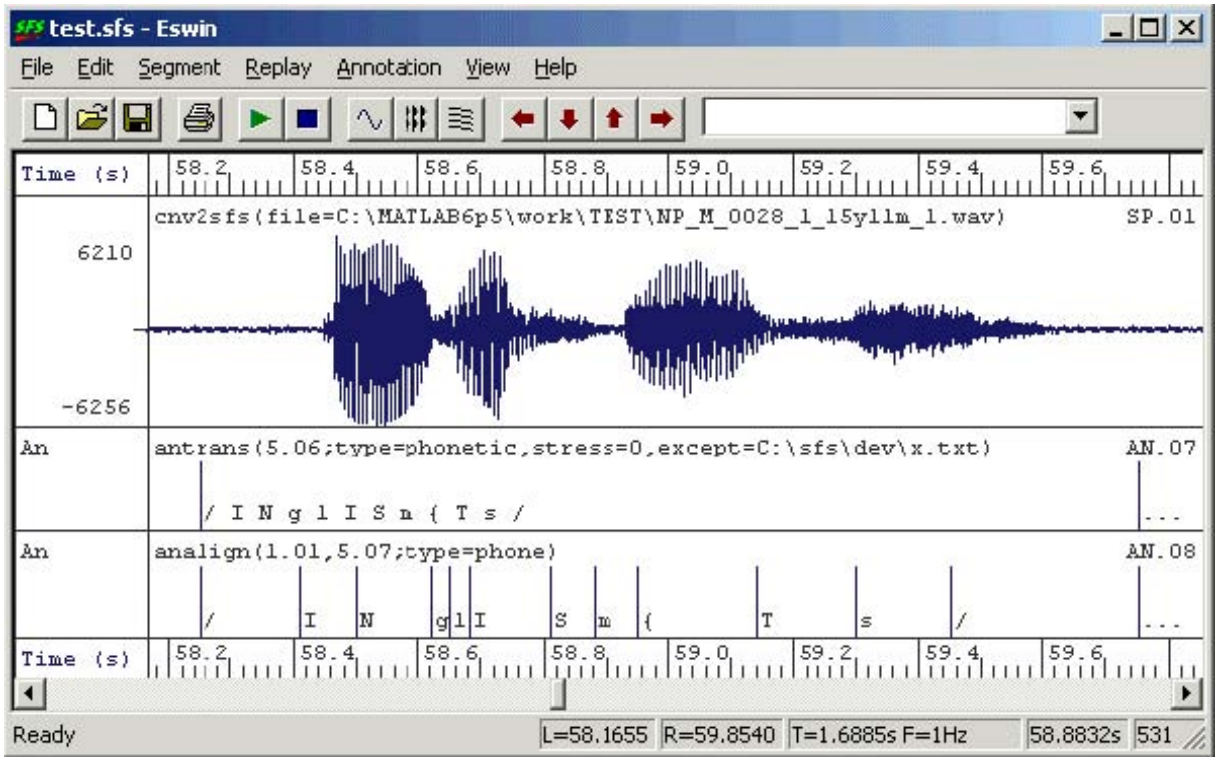


Figure C.4.1 - Chunked vs. aligned transcription

Automatic alignment

Analign has two modes of operation. In the first mode, input is a set of transcribed chunks in which the start and end points of the chunks are fixed. The program then finds an alignment between the segments in the transcription and the signal region identified by the chunk. In the second mode, the program chooses chunks on the basis of pause labels, and all phonetic annotations between the pauses are realigned. By default pauses are identified by labels containing the SAMPA pause symbol "...". You can use the first mode to get a basic alignment, then you can use the second mode to refine the alignment by adding or deleting phonetic annotations and re-running analign.

To align chunked phonetic transcription, select the input speech and annotation items and choose

Tools|Annotation|Auto-align phone labels. Choose option "Fixed label boundaries" to only perform alignment *within* a label. See Figure C.4.2.

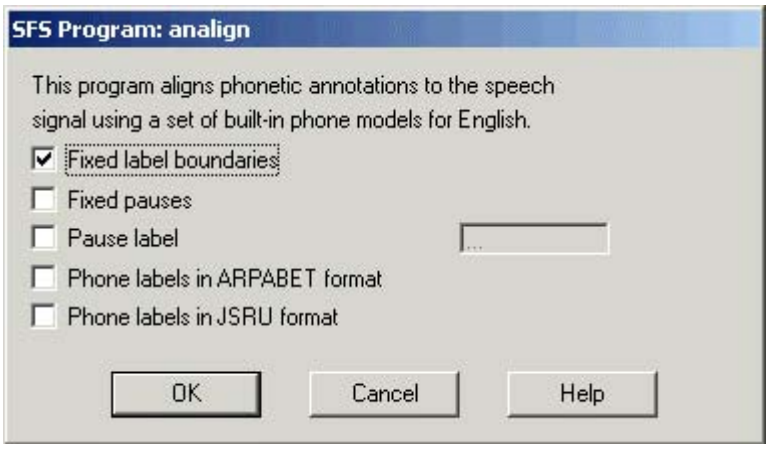


Figure C.4.2 - SFSWin Align Labels dialog (1)

The automatic alignment is performed using a set of phone hidden-Markov models which have been trained on Southern British English. You may need to replace these for other languages and accents. Look at the manual page of analign for details. The HMMs that come with SFS have been built using the Cambridge hidden Markov modelling toolkit HTK (also see Appendix E).

Automatic alignment is an approximate process, and you will almost certainly see places in the aligned transcription where the alignment is not satisfactory. Common kinds of problems are:

1. Segments stretched over unmarked pauses.
2. Segments compressed when smoothed or elided in rapid speech.
3. Poor alignment in consonant clusters and unstressed syllables in rapid speech.
4. Poor identification of speech-to-silence boundaries.
5. Poor alignment for syllable-initial glides and syllable-final nasals.

You can either correct the alignment manually or you can make changes to the transcription and run analign again. We'll describe these in turn.

Manual editing of transcription alignment

To edit a set of annotations, select a speech signal and the annotations to be edited and choose Item|Display. The waveform and the input annotations are displayed within SFS program eswin. Then right-click with the mouse in the box at the left of the annotations and choose menu option "Edit annotations". An editable copy of the set of annotations will then appear at the bottom of the screen.

Eswin has a number of special facilities to help in the correction of annotation alignments. To demonstrate these, zoom into a region of the signal so that individual annotations are clearly visible. Then click the left mouse button to display the vertical cursor. You will then find that:

- the **left and right arrow keys [<−] and [−>]** shift the left cursor one pixel to the left and right;
- pressing **[Ctrl] together with the arrow keys** will cause the left cursor to jump from one annotation to the previous/next annotation. The annotation label is also copied into the annotation edit box;
- with the left cursor on an annotation, pressing **[Shift] together with the arrow keys** will slide the annotation one pixel left and right;
- with the left cursor on an annotation, pressing the **[Delete] key** will delete an annotation.

You can also delete an annotation by deleting the contents of the annotation edit box and pressing [Return] while the cursor is positioned on an annotation. Figure C.4.3 shows an annotation being moved using the arrow keys.

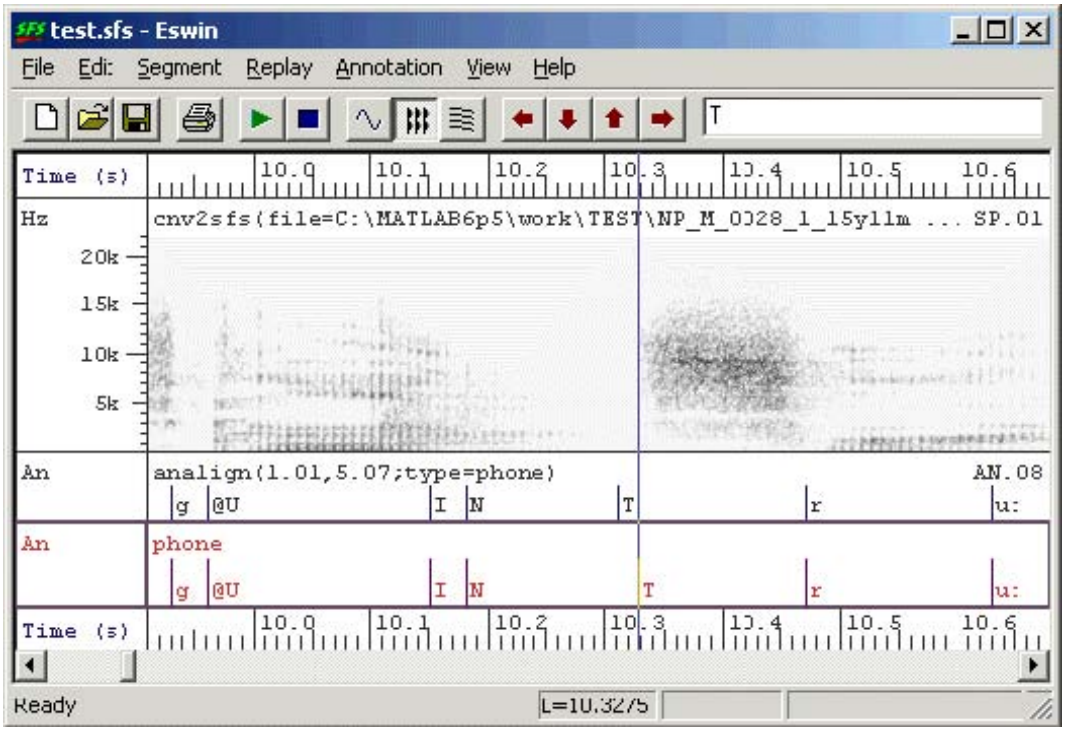


Figure C.4.3 - Manual editing of annotations

Semi-automatic alignment correction

If the automatic alignment has failed for fairly obvious reasons, it may be more efficient to redo the alignment with the problem fixed than to reposition every annotation manually. For example, a common problem is a failure to mark short pauses that occur within utterances. It is easy to add these pauses as new annotations (with "/" symbols) and to re-do the automatic alignment.

Because we have aligned the transcription once, we do not want analign to preserve the current annotation label boundaries. Instead we probably want to preserve the position of major pauses in the transcription (marked with "..." symbols). To re-do the alignment this way, select the speech signal and the edited aligned annotations and choose Tools|Annotations|Auto-align phone labels, but choosing option "Fixed pauses", see Figure C.4.4. If you use a different symbol to "..." for pauses, enter the symbol as the "Pause label" parameter.

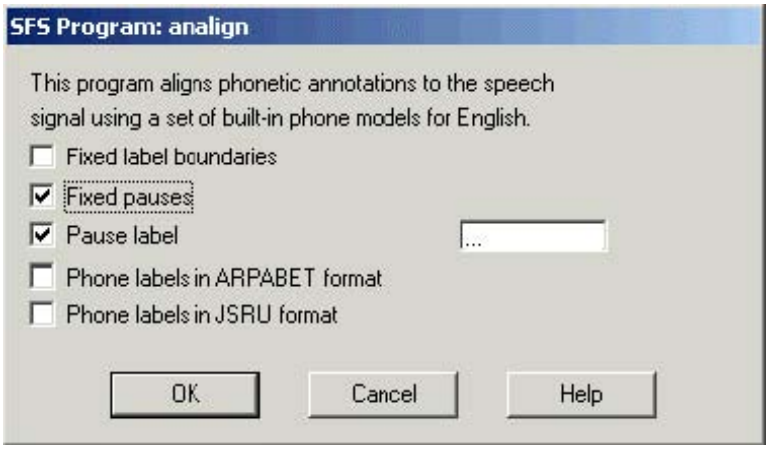


Figure C.4.4 - SFSWin Align Labels dialog (2)

5. Verification and Post-processing

One of the final steps in annotating a signal is to verify that the annotation labels match your normal conventions for labeling. For example, you may want to check that only labels from a given inventory are present. Another step in the final processing may be to collapse adjacent silences/pauses into single labels.

These kinds of operation can be most easily performed with an SML script. We will present two scripts: the first checks labels against an inventory stored in a file, the second collapses silences and pauses.

Verification

We assume that an inventory of symbols is saved in a text file with one symbol per line. The following script then reports the name and location of all symbols not in the inventory.

```
/* anverify - verify annotation labels come from known inventory */

/* inventory */
file    ip;
string    itab[1:1000];
var    icnt;

/* load inventory from file */
init {
   string    s;
   openin(ip,"c:/sfs/dev/sampa.lst");

   input#ip s;
   while (compare(s,s)) {
      icnt = icnt+1;
      itab[icnt] = s;
      input#ip s;
   }

   close(ip);
}

/* process an annotation item */
main {
   var    i,num;

   num = numberof(".");
   for (i=1;i<=num;i=i+1) {
      if (!entry(matchn(".",i),itab)) {
  print $filename,"\t";
         print timen(".",i):8:4,"\t";
         print matchn(".",i)," – illegal symbol\n";
      }
   }
}
```

To run this script, copy and paste it into a file "anverify.sml" and create the inventory file "sampa.lst". Then select the annotation item to check and run Tools|Run SML script, see Figure C.5.1.

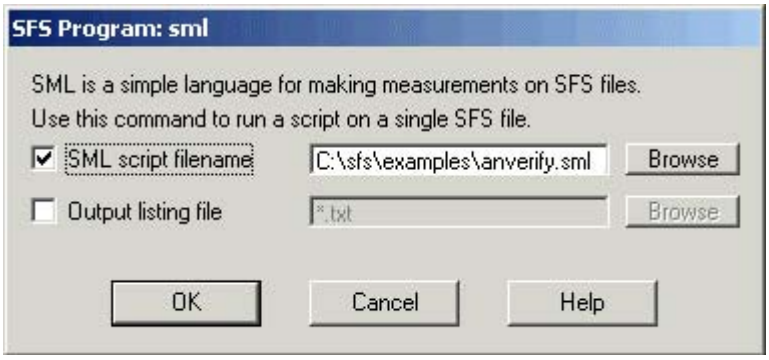


Figure C.5.1 - SFSWin Run SML script dialog

Post-processing

In this script we collapse adjacent annotations if they both label silence or pause. Specifically:

First	Second	Result
…	…	…
…	/	…
/	…	…
/	/	/

The processed annotation item is saved back into the same file.

```
/* ansilproc – collapse adjacent silence annotations */

item    ian;    /* input annotations */
item    oan;    /* output annotations */

/* check annotation for silence */
function var issil(lab)
string lab
{
   if (compare(lab,"/")==0) return(1);
   if (compare(lab,"...")==0) return(1);
   return(ERROR);
}

main {
   var    i,j,numf;
   var    size,cnt;
   string    lab,lab2;

   /* get input & output */
   sfsgetitem(ian,$filename,str(selectitem(AN),4,2));
   numf=sfsgetparam(ian,"numframes");
   sfsnewitem(oan,AN,sfsgetparam(ian,"frameduration"),
      sfsgetparam(ian,"offset"),1,numf);

   /* process annotations */
   i=0;
   cnt=0;
   while (i < numf) {
      lab = sfsgetstring(ian,i);
      if ((i<numf-1) && issil(lab)) {
         /* is a non-final silence */
         size=sfsgetfield(ian,i,1);
         j=i+1;
         lab2 = sfsgetstring(ian,j);
         while ((j<numf) && issil(lab2)) {
            if (compare(lab2,"...")==0) lab = lab2;
            size=size + sfsgetfield(ian,j,1);
            j=j+1;
            if (j<numf) lab2 = sfsgetstring(ian,j);
         }
         sfssetfield(oan,cnt,0,sfsgetfield(ian,i,0));
         sfssetfield(oan,cnt,1,size);
         sfssetstring(oan,cnt,lab);
         i=j;
      }
      else {
         /* final or non-silence, just copy */
         sfssetfield(oan,cnt,0,sfsgetfield(ian,i,0));
         sfssetfield(oan,cnt,1,sfsgetfield(ian,i,1));
         sfssetstring(oan,cnt,lab);
         i=i+1;
      }
      cnt = cnt + 1;
   }

   /* save result */
   sfsputitem(oan,$filename,cnt);
}
```

Copy and paste this script into ansilproc.sml, and run it using Tools|Run SML script. An example of the effect of the script is shown in Figure C.5.3

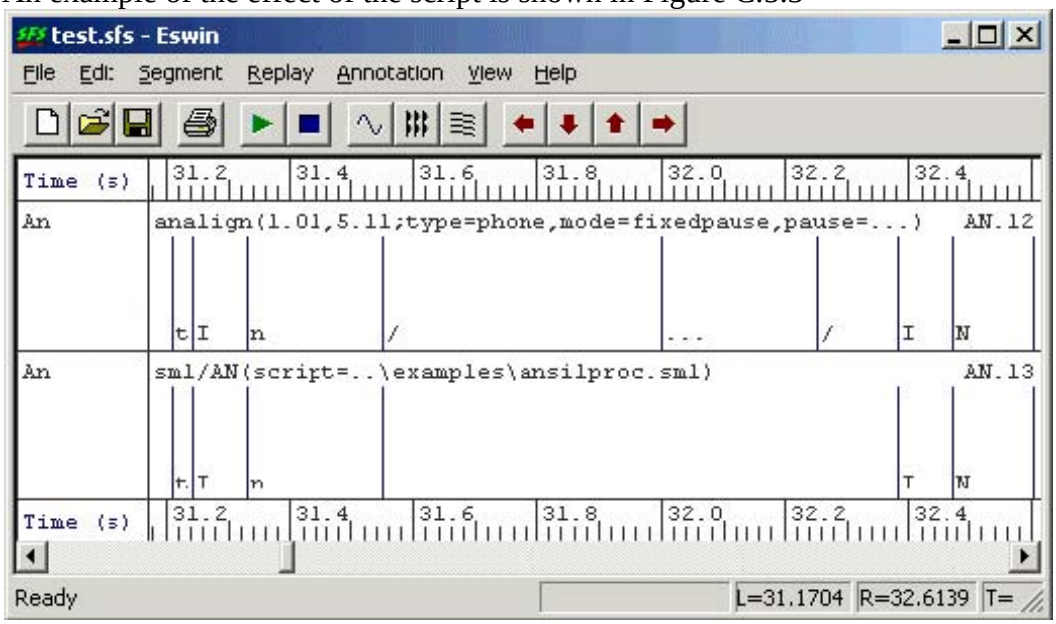


Figure C.5.2 - Post-processing of silences

6. Other special processing

This section refers specifically to annotated recordings of dysfluent speech made available by the Speech Group of the Department of Psychology at UCL (www.psychol.ucl.ac.uk).

Description of UCL Psychology phonetic annotation system

Below is a summary of the phonetic mark-up developed by the Speech group and used on the dysfluent speech database. The basic phonetic symbol set is the JSRU symbol set described in Appendix B.

Word Boundaries

Word boundaries are indicated in the phonetic transcription with a symbol placed before the first syllable in the word:

- A forward slash "/" is used to mark a function Word
- A colon ":" is used to mark a content Word

Function words are closed class words (only about 300 in English) which perform grammatical functions while content words are open class words which carry meaning.

Function Words	
Prepositions:	of, at, in, without, between
Pronouns:	he, they, anybody, it, one
Determiners:	the, a, that, my, more, much, either, neither
Conjunctions:	and, that, when, while, although, or
Modal verbs:	can, must, will, should, ought, need, used
Auxilliary verbs:	be (is, am, are), have, got, do
Particles:	no, not, nor, as
Content Words	
Nouns:	John, room, answer, Selby

Adjectives:	happy, new, large, grey
Full verbs:	search, grow, hold, have
Adverbs:	really, completely, very, also, enough
Numerals:	one, thousand, first
Interjections:	eh, ugh, phew, well
Yes/No answers:	yes, no (as answers)

Beware that the same lexical word can function as either content or function word depending on its function in an utterance:

1. have

    a. "I have come to see you" = Function Word (Auxillary)

    b. "I have three apples" = Content Word (Full Verb)

2. one

    a. "One has one's principles" = Function Word (Pronoun)

    b. "I have one apple" = Content Word (Numeral)

3. no

    a. "I have no more money" = Function Word (Negative Particle)

    b. "No. I am not coming" = Content Word (Yes/No Answer)

Examples with the word boundary markers:

- "I saw him in the school." = /ie :saw /him /in /dha :skuul.
- "I have come to see you." = /ie /haav :kam /ta :see /yuu.

Syllable Boundaries

The appropriate stress marker from the list below is placed at the start of each syllable, to mark syllable boundaries as well as stress:

- Exclamation mark ! prior to emphatically-stressed syllable.
- Double quote " prior to primary-stressed syllable.
- Single quote ' prior to secondary-stressed syllable.
- Hyphen - prior to unstressed syllable not in word-initial position.

In the case of a word-initial syllable, the stress marker is positioned immediately after the word marker. The only exception is in the case of an unstressed first syllable, which does not receive a dash but instead only receives the word marker. The dash, by default, indicates that a syllable is not word initial, as well as indicating that it is unstressed.

Examples:

- :"sen-ta (centre)
- :di"tekt (detect)
- :'in-ta"naa-sha-nl (international)
- :"in-ta'naa-sha-na-lie-zai-shn (internationalization)
- :in-ta'naa-sha-na-lie"zai-shn (internationalization)

Dysfluencies within words

All dysfluent phones are entered in UPPER-CASE at a finer-grained level of transcription wherein each upper-case symbol represents 50ms duration estimates.

Multiple upper-case phones may be represented with an explicit repetition count: {x *num*}, e.g. if the duration of a prolonged F were 5 times 50ms, it could be transcribed either as "FFFFF" or F{x 5}. The latter is helpful in transcribing very long prolongations like F{x 30}.

A "Q" is used to indicate a pause within a word of 100ms, e.g. a 300ms dysfluent pause would be transcribed as either QQQ or Q{x 3}.

Examples:

- Prolongations, e.g. /dhaats :FFFFFaan"taa-stik or /dhaats :F{x 5}aan"taa-stik.
- Repetitions, e.g. dhaats /a :"load /av :"BA BA BA BA Bawl-da'daash or /dhaats /a :"load /av :"KQ KQ KQ Ko-bl-az.

Note that a space does not indicate pausing. In the first repetition example above, there is no pause between the repetitions of the "BA" sound. There are, however, brief pauses (100ms) between the "K" sounds in the second example

For ambiguous phonetic transcription sequences the {x…} convention is used when the symbol is repeated, e.g. the transcription "AAAAA" refers to a prolonged "A", but "AA{x5}" refers to a prolonged "AA".

Other dysfluencies which cannot be transcribed are entered in the form of a comment at the place where it occurs. For example, a block can be entered as {U block}. All dysfluencies are marked in the phonetic transcriptions.

Marking of supralexical dysfluencies

Word repetitions are transcribed using the syllable or word repetition convention described below (++|++), with the exception that a monosyllabic word that is repeated with no pausing, or very little, and is judged to be 'stuttered' can be transcribed within one word, thus:

- /AAND /aand, or, /AANDQQ/aand

Any repeated monosyllabic words that are separated by significant pausing (more than two Qs) are transcribed using the convention below.

In the transcribed speech, the section of "replaced" and "replacement" speech are

enclosed by two "+" signs and the two sections are separated by a vertical bar "|". For example:

• Syllable repetition:

/dhei /waz :"noa + :"ree Q + | + :"ree + -zan /fa /him ta :"duu /it.

• Word repetition:

/dheiz :"noa :"u-dha + /dhat + | + /dhat + /ie :'noa /ov.

• Backtracking:

/dheiz + :"noa :"u-dha /dhat Q + | + :"noa :"u-dha /dhat + /ie :'noa/ov.

• Backtracking + elaboration:

/dheiz + :"noa :"u-dha /dhat Q + | + :"noa :"u-dha :'i-di-at /dhat + /ie :'noa /ov.

Marking of pauses between words

Pauses are marked with lower-case "q" if they are part of fluent speech intonation. Dysfluent pauses are marked with upper-case "Q".

Marking of other comments do for f in $d/???C*.PHN

Other comments by the transcriber are entered into the transcription using the convention {U ...text...}. This might be used for speech that could not be transcribed or for other sound events. g=`echo $f | sed s/.PHN//`

if test -e $g.WV1

Division into tiers then

In this section we will look at how the dysfluent transcription may be divided into two tiers: the first describing the word and dysfluency events, the second describing the phonetic sequence. The advantage of this separation is that the phonetic symbol sequence may then be time-aligned with the speech signal. Figure C.6.1 shows an example of the database annotation prior to processing. h=`echo $g | sed s%c:/data/wsjcam0/si_tr/%%`

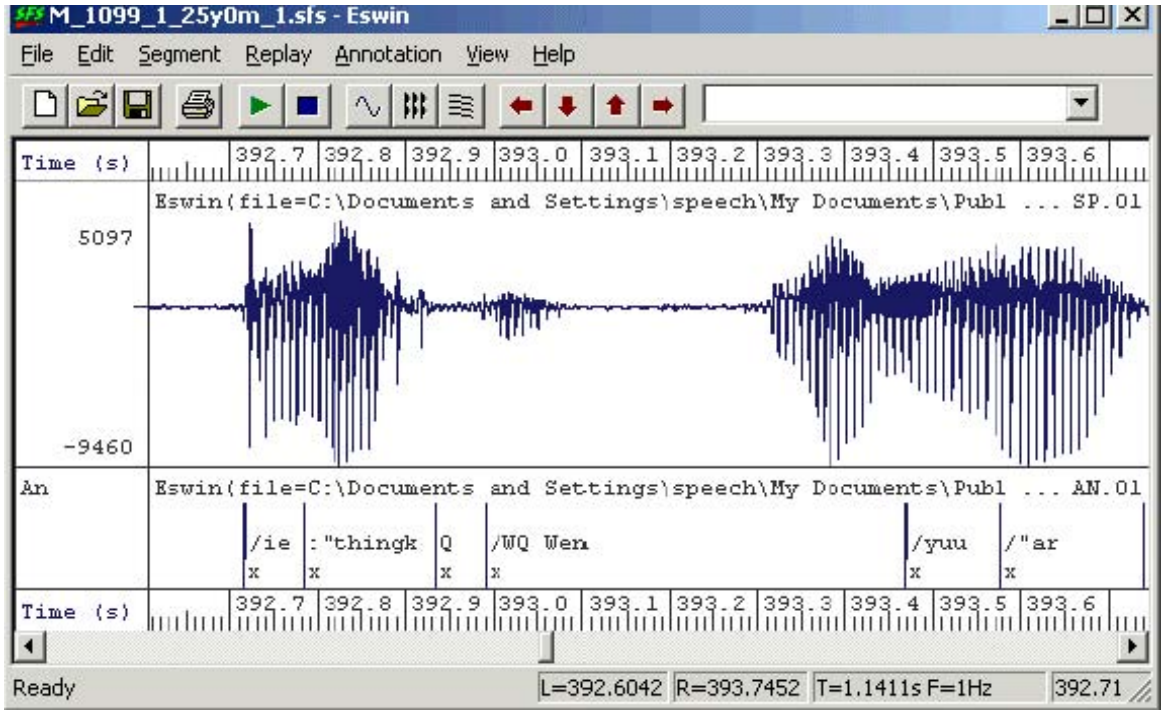


Figure C.6.1 - Example dysfluency mark-up before processing

echo $h >>basetest.lst

fi

The following script takes as input an annotation item marked up using the system above which has been time-aligned at the level of individual syllables, as can be seen in Figure C.6.1. The output of the script is two further annotation sets. The basic principle of operation is that the input transcription is parsed symbol by symbol, while some symbols are directed into the word tier and others into the phone tier. In addition, redundant annotations that only mark the ends of syllables (and are shorter than about 5ms long) are removed. done

The marking of dysfluency is changed so that conventional phonetic annotation is used in the phone tier, and a new marker "{D}" is added to the word tier. This allows us to process the phone tier using the automatic alignment procedure described in section 4. done

At present no processing of "multiplier" markers is performed, so that "AA{x 5}" is divided up into "aa" on the phone tier and "{x 5} {D}" on the word tier.

/* anfluency - process phonetic annotations used on fluency data */ The script looks for phonetic annotation files of the form "???C*.PHN" and as long as there is a matching audio signal ".WV1" it adds the basename of the file to a list. This script creates a file called "basetrain.lst"; which has the speaker directory and base filename for each training file, and a file called "basetest.lst", which has the speaker directory and base filename for each test file. For the data used, there are about 3000 files in basetrain.lst and 900 in basetest.lst.

We can now load the audio signal and the source phonetic annotations into an SFS file using the following script:

```
/* version 1.0 - June 2004 # domakesfs.sh
 * #
 * This script take a set of phonetic annotations # 1. Make training and testing directories
 * from the UCL Psychology Speech Group fluency #
 * database and normalises them to be consistent mkdir train test
 * with SFS conventions. for d in 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 * do
 */
item    ian;    /* input annotations */
item    oanp;    /* output annotations - phonetic */
var     oanpcnt;
item    oanw;    /* output annotations - word */
var     oanwcnt;

/* check for uppercase */
function var isupper(str)
{
   string str;
   if ((ascii(str)>=65)&&(ascii(str)<=90)) return(1);
   return(ERROR);
}

/* convert to lower case */
function string tolower(src)
{
   string src;
   string dst;
   var       i;
   dst="";
   for (i=1;i<=strlen(src);i=i+1) {
      if ((ascii(src:i:i)>=65)&&(ascii(src:i:i)<=90)) {
         dst = dst ++ char(ascii(src:i:i)+32);
      }
      else {
         dst = dst ++ src:i:i;
      }
   }
   return(dst);
}

/* check next character for digraph */
function string checknext(prefix,ch,label)
string label;
{
   string prefix,ch;
   if (strlen(label)==0) return(prefix);
   if (index(ch,label:1)==1) {
      prefix=prefix++(label:1);
      label=label:2:strlen(label);
   }
   return(prefix);
}

/* strip next symbol from front of string */
function string nextsymbol(label)
```

```
 * Input is transcription of syllables or words   mkdir train/C0$d
 * in JSRU format with additional markers showing done
 * word category and dysfluency for d in 0 1 2 3 4 5 6 7 8 9
 * do
 * Output is two new annotation sets: one containing   mkdir test/C1$d
 * only the phonetic labels and stress markers, parsed done
 * with spaces between the symbols; and one with the #
 * word category and dysfluency mark-up # 2. Convert audio and labels to SFS
 * #
for f in `cat basetrain.lst`
  hed -n train/$f.sfs
  cnv2sfs c:/data/wsjcam0/si_tr/$f.wv1 train/$f.sfs
  anload -f 16000 -s c:/data/wsjcam0/si_tr/$f.phn train/$f.sfs
done
for f in `cat basetest.lst`
do
  hed -n test/$f.sfs
  cnv2sfs c:/data/wsjcam0/si_tr/$f.wv1 test/$f.sfs
  anload -f 16000 -s c:/data/wsjcam0/si_tr/$f.phn test/$f.sfs
done
```

do

An example of the processing performed by the script can be seen in Figure C.6.2.

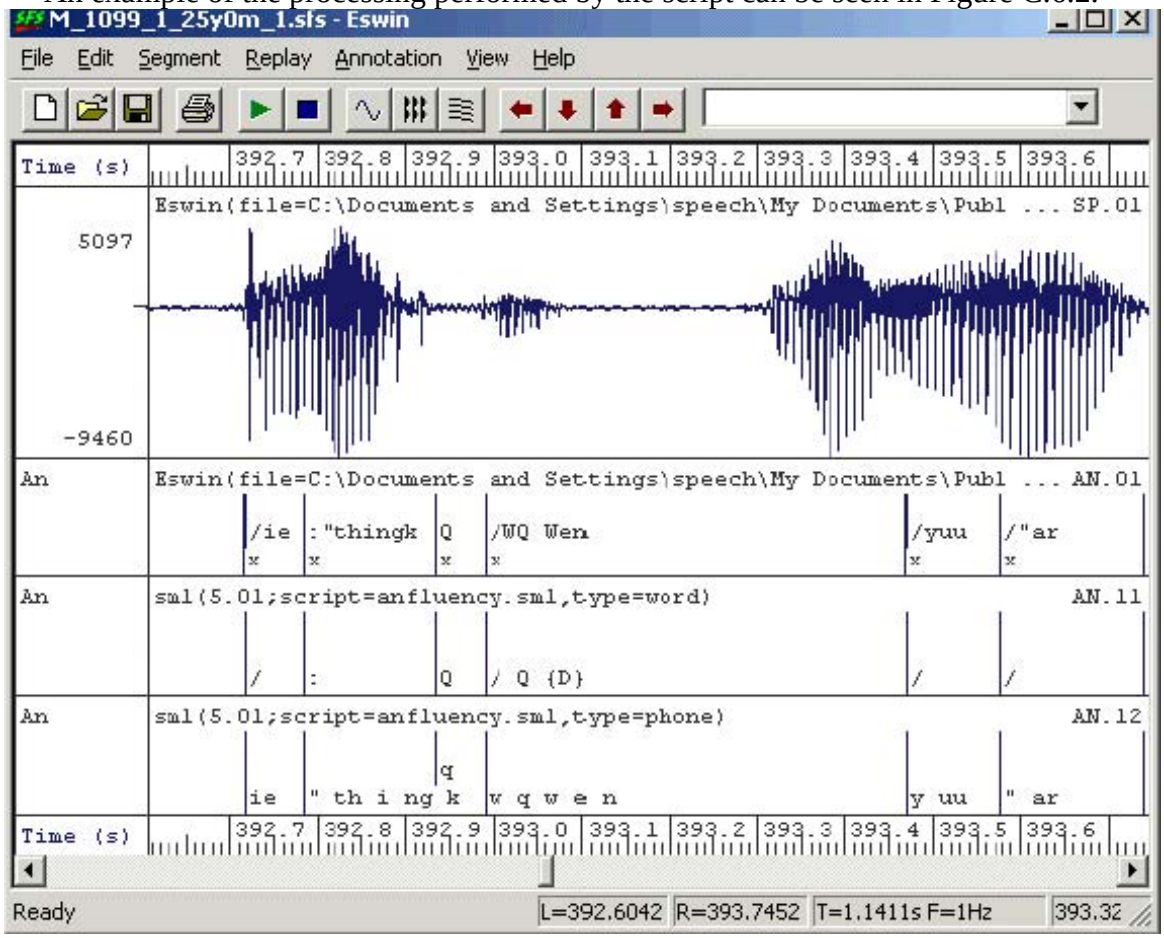


Figure C.6.2 - Example dysfluency mark-up divided across two tiers

Subsequent Processing

Phonetic alignment of phone tier

The automatic phonetic alignment of the phone tier can be performed using the tools describe in section 4 of this appendix. Selecting a suitable speech and annotation item, choose menu option Tools|Annotations|Auto align phone labels. For the phone tier annotations above we only want to align within a phone label and to use JSRU symbols. See Figure C.6.3.

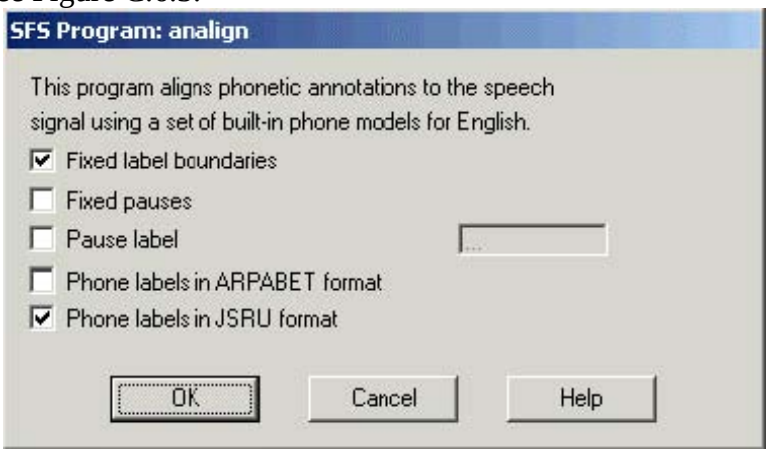


Figure C.6.3 - Automatic phonetic alignment on JSRU symbols

Dysfluency statistics

Finally, we will show how the identification of dysfluencies in the word tier can be used to collect some statistics about their occurrence. This script counts where dysfluencies occurred and their typical duration.

```
/* dysstats.sml - measure some statistics about dysfluencies */

/* counts */
var    ncontent;    /* # in content words */
var    nfunction;    /* # in function words */
var    npause;        /* # after pause */

/* stats */
stat    sdur;        /* stats on duration */

/* for each input file */
main {
   var    num,i;
   string    last;
   string    lab;

   num=numberof(".");

   last="";
   for (i=1;i<=num;i=i+1) {
      lab=matchn(".",i);
      if (index("{D}",lab)) {
         if (index("/",lab)) {
            nfunction=nfunction+1;
         }
         else if (index(":",lab)) {
            ncontent=ncontent+1
         }
         if (index("Q",last)) {
            npause=npause+1;
         }
         sdur += lengthn(".",i);
      }
      last=lab;
   }
}

/* summarise */
summary {
   print "Files processed          : ",$filecount:1,"\n";
   print "Number of dysfluencies   : ",nfunction+ncontent:1,"\n";
   print "Dysfluent function words : ",nfunction:1,"\n";
   print "Dysfluent content words  : ",ncontent:1,"\n";
   print "Dysfluencies after pause : ",npause:1,"\n";
   print "Mean dysfluent duration  : ",sdur.mean," +/- ",sdur.stddev,"s\n";
}
```

An example run of the script on one 15 min recording is shown below:

Files processed : 1
Number of dysfluencies : 28
Dysfluent function words : 18
Dysfluent content words : 10
Dysfluencies after pause : 7
Mean dysfluent duration : 0.4118 +/- 0.3442s

Files processed : 1
Number of dysfluencies : 28
Dysfluent function words : 18
Dysfluent content words : 10
Dysfluencies after pause : 7

Bibliography

- BEEP British English pronunciation dictionary at ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/dictionaries/beep.tar.gz.
- Hidden Markov modeling toolkit at http://htk.eng.cam.ac.uk/.
- International Phonetic Alphabet at http://www.arts.gla.ac.uk/IPA/ipachart.html.
- SAMPA Phonetic Alphabet http://www.phon.ucl.ac.uk/home/sampa/.
- Speech Filing System at http://www.phon.ucl.ac.uk/resource/sfs/.
- UCL Psychology Department at http://www.psychol.ucl.ac.uk/.
- VoiScript at http://www.phon.ucl.ac.uk/resource/voiscript/.

Appendix D Audio analysis with SFS

Researchers might be interested in the way that production of particular sounds is affected by a disorder such as stammering. They may resort to audio analysis to do this. This Appendix describes some basic ways in which audio analysis of speech can be performed using SFS utilities. The main topic covered is formant analysis, which is a way of representing how speech output changes over time as the articulators move to produce the consonant and vowel sounds described in Appendix B. Some background on 1) articulatory phonetics and spectrographic analysis of speech and 2) statistical analysis are assumed. For readers needing the background for, or those who wish to revise the concepts behind 1,) reference can be made to Rosen and Howell (1991) which provides an elementary, non-mathematical introduction to this area. Hyperlinks to websites that cover some of the critical concepts behind the statistical topics are given at the end. Some software is presented and described in this appendix but programming experience is not assumed. This tutorial refers to versions 4.6 and later of SFS and appears in the documentation on the SFS website. Visit the SFS website to obtain your software (http://www.phon.ucl.ac.uk/)

1. Formant Analysis Strategy

Perhaps the most obvious way to do formant analysis with SFS is to load up an audio signal, choose Tools|Speech|Display|Cross-section, then make measurements of formant frequencies interactively, writing the results down on a piece of paper, see figure D.1.1. You can then type your results into a statistics package and make whatever comparisons you need. This is **not** the strategy we will be presenting in this tutorial.

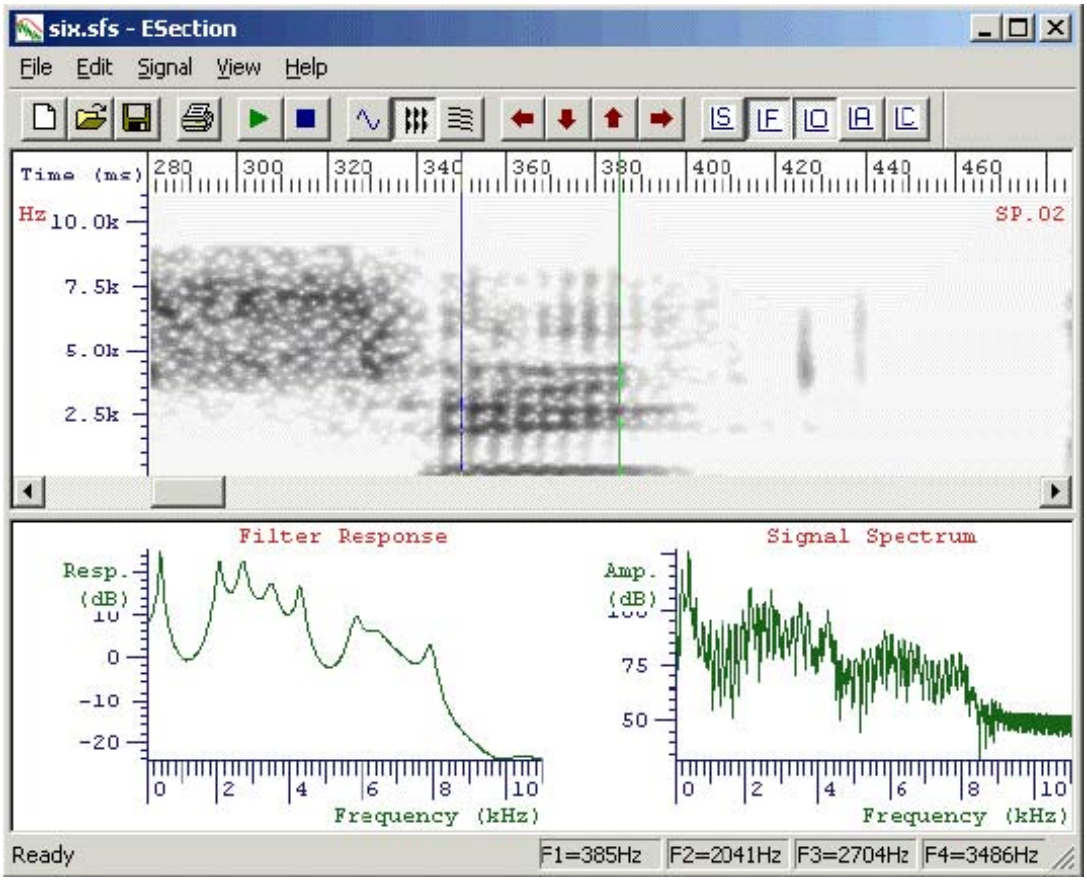


Figure D.1.1 - Interactive formant measurement (frequencies in status bar)

There are a number of deficiencies in the direct, interactive route:

- Lack of consistency: how do you know that you are positioning the cursors in a consistent fashion every time?
- Potential bias: are you sure you haven't chosen the position of the cursors to obtain the expected formant frequency values?
- Inflexibility: if you want to go back and change the way the formants are measured, or if you want to go back and collect other data (e.g. durations), you have to go through all the data again *without knowing exactly how you made the first set of measurements*.
- Cost: measuring interactively is slow and time-consuming. If you can use a speech corpus that has been phonetically labeled, then you can be much more productive by exploiting those labels.
- Amount of data: a semi-automatic procedure can analyse more data and provide a wider range of statistics on the distribution of formant frequencies.

The strategy we will be presenting in this tutorial follows these steps:

1. Annotate the signal;

2. Perform automatic formant analysis of all the speech data;

3. Use a script to extract the distribution of formant values;

4. Analyse the formant distributions.

2. Annotating the audio signal

If you are lucky you will find ready-annotated material suitable for your purposes (11 such files from speakers who stammer are available from the UCL data archive described in Appendix A). Phonetically-annotated speech corpora are becoming more common, though they are still rare for speech disorders like stammering. You will probably also require data on fluent speakers for control purposes and if you are thinking of analysing one of the major languages of the world you should investigate whether annotated recordings are available for fluent speakers at least. Often these are supplied to speech researchers at a much lower price than they are made available to speech technology companies. The two major world suppliers of speech corpora are the Linguistic Data Consortium (www.ldc.upenn.edu) and the European Language Resource Association (www.elra.info).

We will not concern ourselves here with converting corpus data to be compatible with SFS, but there exist tools in SFS (such as cnv2sfs and anload) to help make this easier. Instead we will briefly discuss the use of SFS to add annotations to the signal. We assume that we will only be annotating the regions of the signal where we want to make formant measurements, rather than performing an aligned phonetic annotation of the whole signal (see the sister tutorial in Appendix C on Phonetic annotation).

Typically formant measurements are made on syllabic nuclei, where there is likely to be voicing and a relatively unobstructed vocal tract. We will describe the annotation of monophthongal vowels, although the procedure could easily be adapted to deal with more complex elements.

From within SFSWin, select the speech item to annotate and choose Tools|Speech|Annotate|Manually. Enter a suitable name for the annotations (say, "labels"). The speech signal will be displayed, with a box at the bottom of the display where the annotations may be added/edited. Adjust the display to show waveforms and/or spectrograms as you wish. Use the cursors to isolate regions of the signal until you find the first vowel segment you want to annotate. Zoom in so that the vowel region is clearly visible - perhaps filling about one-quarter of the display. See Figure D.2.1.

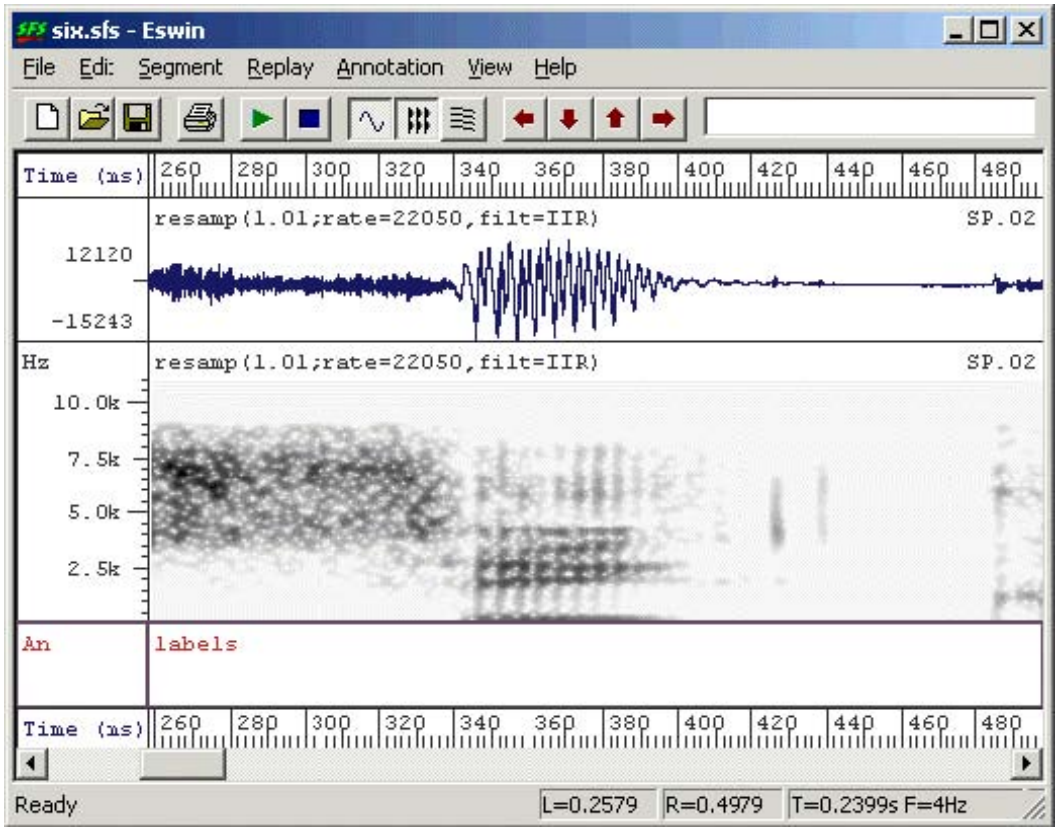


Figure D.2.1 - Ready for annotation

The next question is how best to position the annotations. Should one attempt to determine the "centre" of the vowel, or its "edges", or where the formants are "stationary"? The problem is that none of these have clear, unambiguous definitions. The best choice is the one that makes the least assumptions and has the least potential for bias. It is suggested that a strategy for labeling is chosen that is easy and reliable. In the case of Figure D.2.1, where the vowel is preceded and followed by a voiceless consonant, then the labels should go at the start and end of voicing. In circumstances where the voiced region is shared with another voiced segment, consider estimating the point which is acoustically half-way between the segments. It is then reasonable to propose that one segment dominates on one side of the label, while the other segment dominates on the other side. Once the labels are positioned we can try various programmed strategies for reliably extracting formant frequencies from the region.

To add an annotation, position the left cursor at the start of the region to annotate, and the right cursor at the end. Then using the keyboard, type the following:

[A] [*label*] [RETURN] [B] [/] [RETURN]

The [A] key is a keyboard shortcut meaning label the left cursor, the [B] key is a keyboard shortcut meaning label the right cursor. In Figure D.2.2 we have labeled a segment as being "I_six", that is the vowel /I/ in the word "six". We have marked the end of the segment with "/".

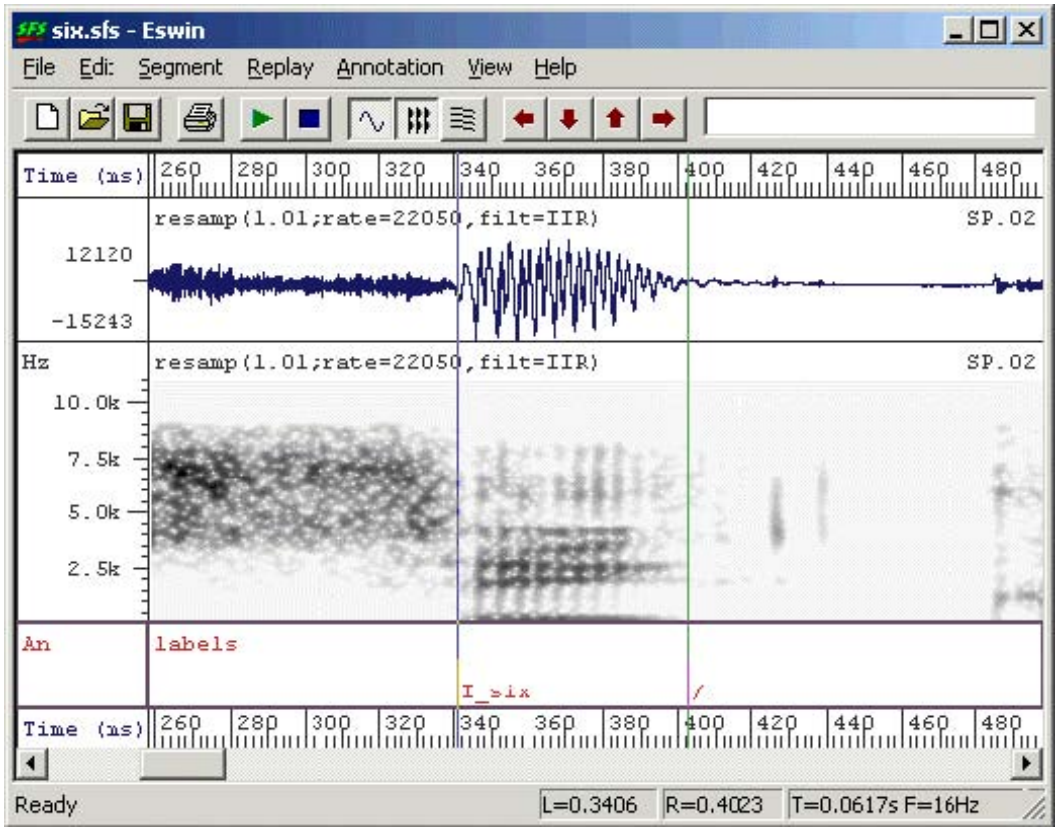


Figure D.2.2 - Annotated vowel segment

Deciding how to label the regions will depend on the kind of phonetic analysis you are planning to undertake. Since adding information to the label is easy when you are doing the labeling, and very difficult to do retrospectively, consider labeling with information about the context as well as the identity of the segment. You may find in your analysis that your assumptions about allophonic variation are wrong, and that identically-labelled segments actually belong to different phonological categories!

One final useful piece of advice is to fill in information about the identity of the speaker and the recording session in the SFS file header. This information may be useful in allowing us to find files and label graphs later. To do this, select option File|Properties in SFSWin and complete the form shown in Figure D.2.3.

Figure D.2.3 - SFSWin File properties

3. Formant Analysis

Introduction

It is worth mentioning at the outset that formant analysis is not an exact science. The task the computer is trying to do is to estimate the natural frequency of vocal tract resonances given a short section of speech signal picked up by a microphone. The task is made complex because the only way information about the resonances gets into the microphone signal is if the resonances are excited with sounds generated elsewhere in the vocal tract - typically from the larynx. Thus the program has to make assumptions about the nature of this source signal to determine how that signal has been modified by the vocal tract resonances. It may be the case that peaks in the spectrum of the sound are caused by vocal tract resonances, but they may be properties of the source. Likewise, it may be the case that every formant is excited, but it may be that the source simply had no energy at a resonant frequency and it was not excited.

In addition formant analysis is made difficult by the following factors: the articulators are constantly moving and the source is changing while producing speech; the sound signal generated by the vocal tract is possibly contaminated by noise and reverberation before it enters the microphone: and sometimes formants can get very close together in frequency - so that two resonances can give rise to a single spectral peak. All this without even mentioning the problems that arise at high fundamental frequencies, when formant frequencies are likely to be biased towards the nearest harmonic frequency.

In all, we should expect that our formant analysis will give rise to "errors", and rather than ignoring them, or hoping that they have no effect, we should build in the possibility of measurement error into our procedures.

Fixed frame analysis

The most common means to obtain formant frequency measurements from a speech signal is through linear prediction on short fixed-length sections of the signal - typically 20-30ms windows. These windows are stepped by 10ms to give spectral peak estimates 100 times per second of signal. Typically each frame delivers about 6 spectral peaks from a signal sampled at 10,000 samples/sec. Not all these peaks are caused by formants, and so a post-processing stage is required to label some of the peaks as being caused by "F1", "F2", etc. This post-processing stage usually makes assumptions about the typical frequency and bandwidth range of vocal tract resonances and their rate of change.

Currently the best program in SFS to perform fixed-frame formant analysis is the formanal program. This can be found in SFSWin under Tools|Speech|Analysis|Formant estimates track. The formant analysis code in this program was originally written by David Talkin and John Shore as part of the Entropic Signal Processing System and is used under licence from Microsoft. The current SFS implementation does not have any user-changeable signal processing parameters, see Figure D.3.1.

Figure D.3.1 - SFSWin Formant estimates dialog

The formanal program performs the following processing steps:

1. Downsamples the signal to 10,000 samples/sec;
2. High-pass filters at 75Hz;
3. Pre-emphasises the signal;
4. Performs linear prediction by autocorrelation on 50ms windows;
5. Root solves the linear prediction polynomial to obtain spectral peaks;
6. Finds the best assignment of peaks to formants over each voiced region of the signal using a dynamic programming algorithm.

Example output is shown in Figure D.3.2.

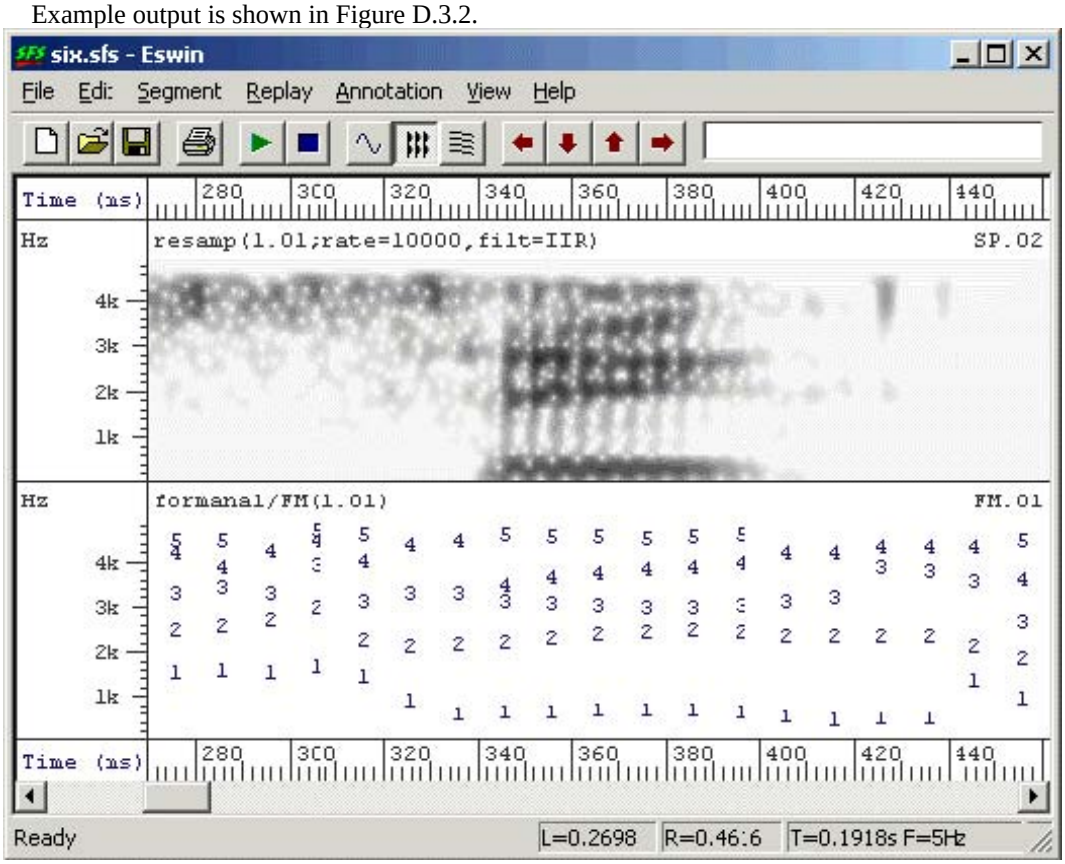


Figure D.3.2 - Example of formant estimation output

Pitch-synchronous analysis

Fixed-frame analysis works well in many situations, and you should certainly try the formanal program on your recordings before attempting anything more sophisticated.

However, with its large windows and DP tracking, formanal will tend to produce rather smooth formant contours which may not reflect accurately the moment-to-moment changes of the vocal tract. A potentially more exact means of obtaining formant frequency measurements is to analyse the data *pitch-synchronously*. Pitch synchronous formant analysis divides the signal up into windows according to a set of pitch epoch markers, such that each analysis window is simply one pitch period long. The result is a set of formant estimates that are output at a rate of one frame per pitch period rather than one frame per 10ms. There are other technical reasons why we expect individual pitch periods as being a better basis for formant estimation.

To perform pitch-synchronous formant analysis, we can use the SFS fmanal program. This program is less sophisticated than formanal and we have to do some careful preparation of the signal before running it. In particular we need to downsample the signal to about 10,000 samples/sec and we need to generate a set of pitch epoch markers. We will discuss these in turn:

Downsampling

If the signal is sampled at a rate higher than about 12,000 samples/sec, it is suggested that you first downsample the signal to about 10,000 samples/sec. To do this, select the signal in SFSWin and choose Tools|Speech|Process|Resample, see Figure D.3.3. Put in a sampling rate of 10,000 samples/sec (or if the original signal is at 22050 or 44100 samples/sec, put in 11025 samples/sec).

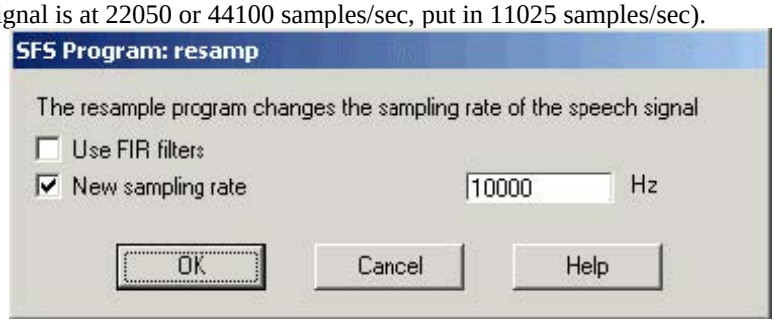


Figure D.3.3 - SFSWin resampling dialog

Pitch epoch marking if Laryngograph signal available

The most reliable way to obtain pitch epoch markers is to make a Laryngograph recording at the same time as the speech signal is recorded. The Laryngograph (www.laryngograph.com) is a specialist piece of equipment that uses two neck electrodes to monitor vocal fold contact area. The resulting stereo signal can be recorded directly into SFSWin or imported from a file using Item|Import|Lx.

From the Laryngograph signal, a set of pitch epoch markers (Tx) can be found from Tools|Lx|Pitch period estimation. Laryngograph recordings are not available for the sample of speech described in Appendix A because we did not want to run the remote risk that attaching the electrodes affected the speech samples obtained. Future research on stammering may use laryngograph signals and allow a convenient way of performing pitch synchronous analysis.

Pitch epoch marking without a Laryngograph signal

A less reliable means to get pitch epoch markers is to analyse the speech signal for periodicity. This can work well for clean, non-reverberant audio signals. To do this, select the speech signal in SFSWin and choose Tools|Speech|Analysis|Fundamental frequency|Pitch epoch location and track (see Figure D.3.4).

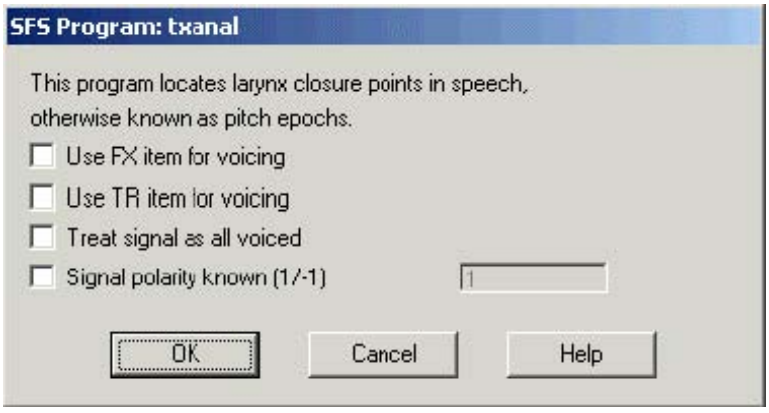


Figure D.3.4 - SFSWin pitch epoch track dialog

The result of the preparation should be two items in the SFS file: a downsampled speech signal and a set of pitch epoch markers (Tx). To perform pitch-synchronous formant analysis, select these two items and choose Tools|Speech|Analysis|Formants estimation. Then select the option "Use Tx for pitch synchronous at offset", leaving the offset value as the default of 0. See Figure D.3.5.

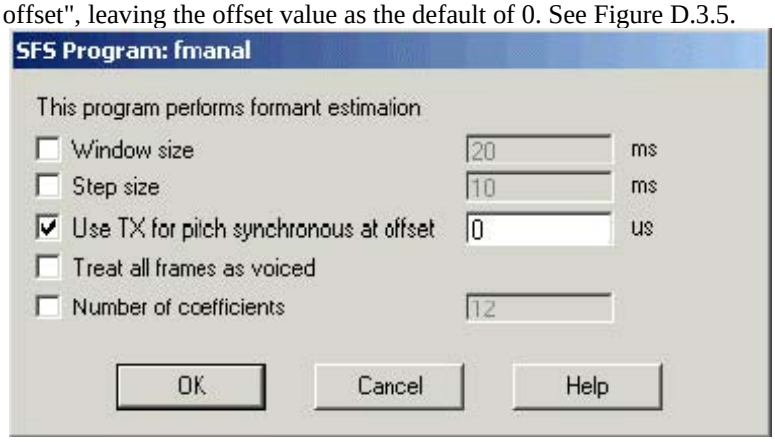


Figure D.3.5 - SFSWin pitch-synchronous formant analysis dialog

An example of pitch synchronous formant analysis is shown in Figure D.3.6. You can see that the formant frames now occur once per pitch period rather than once per 10ms.

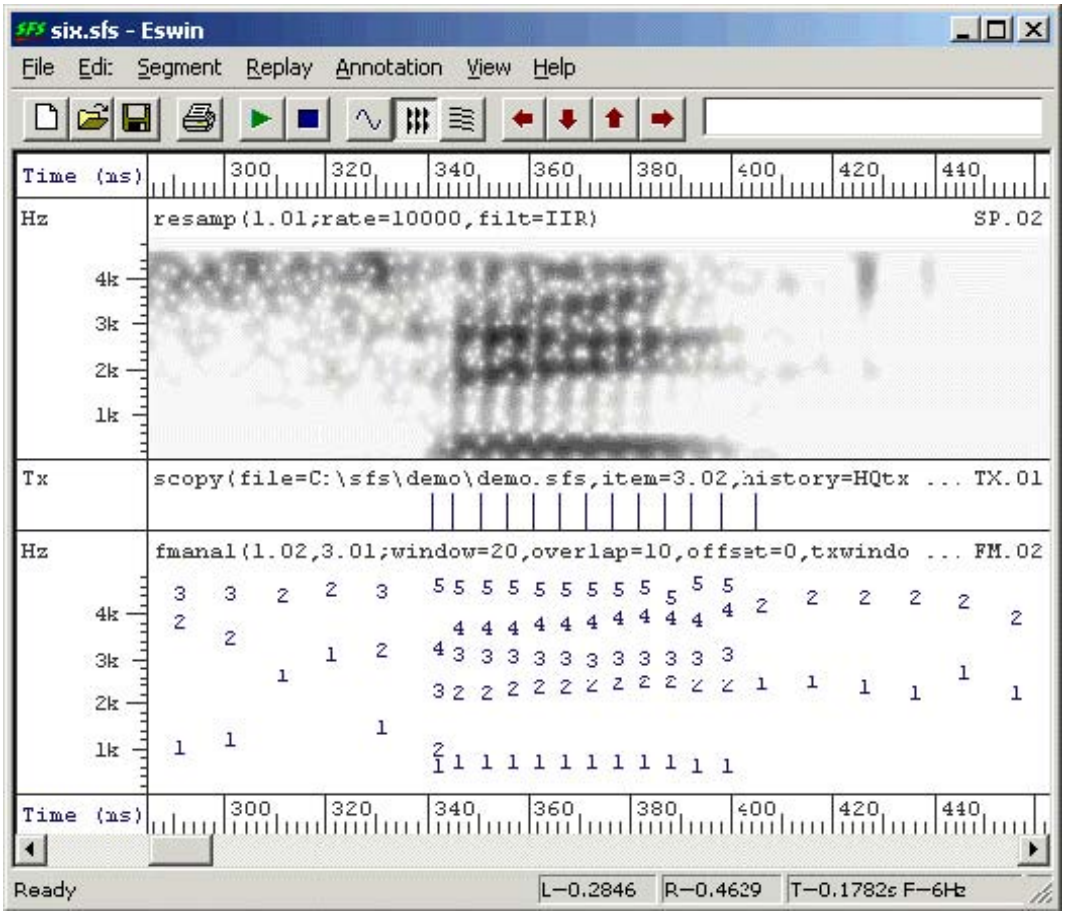


Figure D.3.6 - Example of pitch-synchronous formant estimation output

4. Finding average formant frequencies

We are now in a position to find average formant frequencies in our data. There are three types of average we could consider: (i) the average within a vowel segment, (ii) the average over all segments of a given type for one speaker, or (iii) the average over all vowel segments of a given type spoken by multiple speakers. We will look at these in turn.

Average within a segment

We have annotated our speech signal with labels identifying the vocalic regions where we would like to make a single formant frequency measurement. However, within that region the formant analysis program may have delivered a number of frames of formant estimates. Also the region may cover the whole vocalic segment, while we are interested in a single value which "characterises" the vowel segment. Thus we need to decide how to calculate a characteristic value over what part of the annotated region. In the process we need to take into account the typical contextual changes that occur to vowel formant frequencies in syllables and the typical errors made by formant frequency estimation techniques.

Method 1. Mean over whole segment

We'll start with the most obvious: taking a mean over the whole segment. To demonstrate this, we'll write a script to extract the mean F1, F2 and F3 of each annotated region and save these to a text file in "comma-separated value" (CSV) format. The script is as follows:

```
/* fmsummary.sml – summarise formant measurements from labels */

/* get mean formant value for a segment */
function var measure_mean(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;        /* time */
   var    sum;    /* sum of values */
   var    cnt;    /* # values */

   /* calculate mean over whole segment */
   sum=0;
   cnt=0;
   t=next(FM,stime);
   while (t < etime) {
      sum = sum + fm(pno,t);
      cnt = cnt + 1;
      t = next(FM,t);
   }

   return(sum/cnt);
}

/* for each file to be processed */
main {
   var    num;    /* # annotated regions */
   var    i;
   var stime,etime;
   var    vf1,vf2,vf3;

   num = numberof(".");
   /* for each annotation */
   for (i=1;i<=num;i=i+1) if (compare(matchn(".",i),"/")!=0) {
      stime = timen(".",i);
      etime = stime + lengthn(".",i);
      vf1 = measure_mean(stime,etime,5);
      vf2 = measure_mean(stime,etime,8);
      vf3 = measure_mean(stime,etime,11);
      /* output in CSV format */
      print "\"",$filename,"\",\"",matchn(".",i),"\","
      print vf1,",",vf2,",",vf3,"\n"
   }
}
```

This script calls a function measure_mean() for each annotated region for each formant parameter. The script assumes that there is already a FORMANT item in the file, which can be either fixed-frame or pitch synchronous. To run this script, select the annotation item and the formant item to be processed and choose menu option Tools|Run SML script. Enter the file containing the script above and the name of a text file to receive the output, see Figure D.4.1.

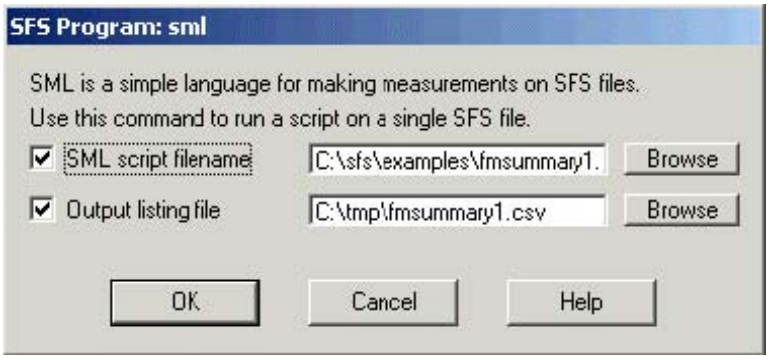


Figure D.4.1 - SFSWin run SMl script dialog

The result of running this script looks like this:

```
"C:\data\ABI\short\brm_f_01_01.sfs","sil", 1120.2540, 2913.2937, 4104.5054
"C:\data\ABI\short\brm_f_01_01.sfs","k", 1146.8917, 2651.2722, 3779.6567
"C:\data\ABI\short\brm_f_01_01.sfs","ae",  789.7859, 1805.1616, 2465.6825
"C:\data\ABI\short\brm_f_01_01.sfs","ng",  662.5625, 1246.0687, 2189.2784
"C:\data\ABI\short\brm_f_01_01.sfs","g",  395.1142,  918.3024, 2061.2703
"C:\data\ABI\short\brm_f_01_01.sfs","ax",  375.5445, 1579.8998, 2311.1436
"C:\data\ABI\short\brm_f_01_01.sfs","r",  438.3379, 1350.3599, 2410.2007
"C:\data\ABI\short\brm_f_01_01.sfs","uw",  462.2808, 1760.6830, 2486.4146
...
```

This CSV format is convenient to use because many spreadsheet and statistics packages can read data in this format. Figure D.4.2 shows this data loaded into Excel.

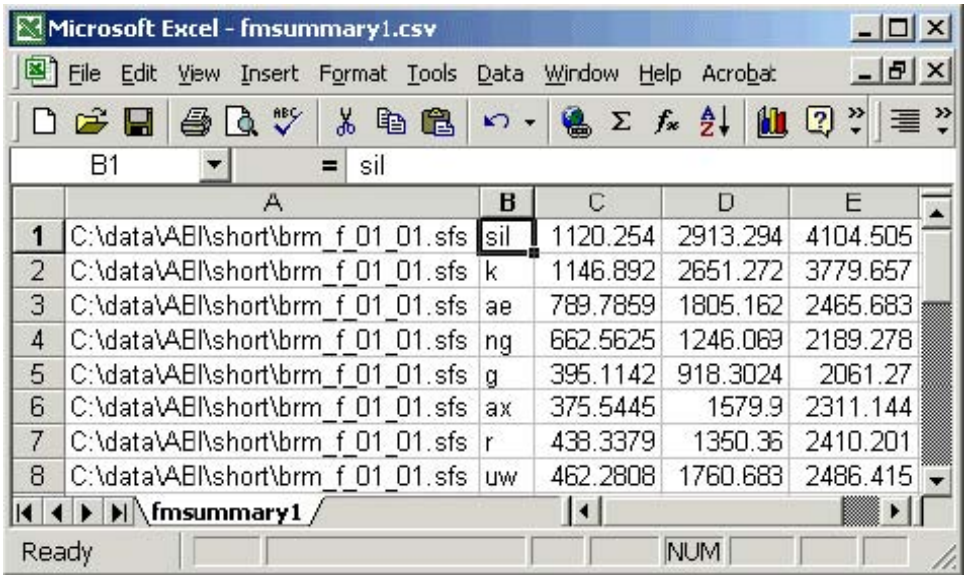


Figure D.4.2 - Formant data loaded into Excel

Method 2. Mean over middle third of segment

Since we expect formant values at the edges of the segment to be less characteristic of the segment than values towards the middle, a refinement of method 1 would be to restrict the analysis to the central third of the segment in time. Here is a replacement measurement function for the script above:

```
/* get mean formant value for a segment */
function var measure_mean_third(stime,etime,pno)
{
   var   stime;   /* start time */
   var   etime;   /* end time */
   var   pno;   /* FM parameter # */
   var   t;      /* time */
   var   sum;   /* sum of values */
   var   cnt;   /* # values */
   var   len;

   /* adjust times to central third */
   len = etime - stime;
   stime = stime + len/3;
   etime = etime - len/3;

   /* calculate mean */
   sum=0;
   cnt=0;
   t=next(FM,stime);
   while (t < etime) {
      sum = sum + fm(pno,t);
      cnt = cnt + 1;
      t = next(FM,t);
   }

   return(sum/cnt);
}
```

Method 3. Median over whole segment

The disadvantage of the mean is that we know that formant tracking errors can occasionally produce wildly inaccurate frequency values. For example, a common tracking error is to relabel F2 as F1, and F3 as F2, and so on. It would seem to be a good idea to remove from the calculation any outlier values. One easy way to do this is to calculate the median over the segment rather than the mean. Here is the adjustment to our script:

```
/* calculate a median */
function var median(table,len)
{
   var table[];    /* array of values */
   var len;        /* # values */
   var    i,j,tmp;

   /* sort table */
   for (i=2;i<=len;i=i+1) {
      j = i;
      tmp = table[j];
      while (table[j-1] > tmp) {
         table[j] = table[j-1];
         j = j - 1;
         if (j==1) break;
      }
      table[j] = tmp;
   }

   /* return middle value */
   if ((len%2)==1) {
      return(table[1+len/2])
   }
   else {
      return((table[len/2]+table[1+len/2])/2);
   }
}

/* get median formant value for a segment */
function var measure_median(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;        /* time */
   var af[1:1000];    /* array of values */
   var    nf;        /* # values */

   /* calculate median */
   nf=0;
   t=next(FM,stime);
   while ((t < etime)&&(nf < 1000)) {
      nf = nf+1;
      af[nf] = fm(pno,t);
      t = next(FM,t);
   }

   return(median(af,nf));
}
```

Method 4. Trimmed mean over whole segment

The disadvantage of the median is that it only picks one value as representative of the formant contour for the segment. One way to get a smoothed estimate but disregard outliers is to use the "trimmed mean" - that is the mean of the values at the middle of the distribution. In the following variation we calculate a trimmed mean of the central 60% (disregarding the lowest 20% and the highest 20%) - but adjust as you see fit.

```
/* calculate trimmed mean */
function var trimmean(table,len)
{
   var table[];    /* array of values */
   var len;        /* # values */
   var    I,j,tmp;
   var    lo,hi;

   /* sort table */
   for (i=2;i<=len;i=i+1) {
      j = i;
      tmp = table[j];
      while (table[j-1] > tmp) {
         table[j] = table[j-1];
         j = j - 1;
         if (j==1) break;
      }
      table[j] = tmp;
   }

   /* find mean over middle portion */
   lo = trunc(0.5 + 1 + len/5);    /* lose bottom 20% */
   hi = trunc(0.5 + len - len/5);    /* lose top 20% */
   j=0;
   tmp=0;
   for (i=lo;i<=hi;i=i+1) {
      tmp = tmp + table[i];
      j = j + 1;
   }
   return(tmp/j);
}

/* get median formant value for a segment */
function var measure_trimmed_mean(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;        /* time */
   var af[1:1000];    /* array of values */
   var    nf;        /* # values */

   /* calculate trimmed mean */
   nf=0;
   t=next(FM,stime);
   while ((t < etime)&&(nf < 1000)) {
      nf = nf+1;
      af[nf] = fm(pno,t);
      t = next(FM,t);
   }

   return(trimmean(af,nf));
}
```

Method 5. Find straight line of best fit

Since we don't expect the formant frequency to be constant over the segment, another approach is to fit a line to the formant values and choose the value of that line at the centre point of the segment as representative of the segment as a whole. To fit a line, we perform a least-squares procedure as follows:

```
/* calculate last-squares fit and return value at time */
function var lsqfit(at,af,nf,t)
{
   var    at[];    /* array of times */
   var    af[];    /* array of frequencies */
   var    nf;       /* # values */
   var    t;       /* output time */
   var    I
   stat x,y,xy
   var    a,b;    /* coefficients */

   /* collect parameters */
   for (i=1;i<=nf;i=i+1) {
      x += at[i];
      y += af[i]
      xy += at[i]*af[i]
   }

   /* find coefficients of line */
   b = (nf*xy.sum-x.sum*y.sum)/(nf*x.sumsq-x.sum*x.sum);
   a = (y.sum - b*x.sum)/nf;

   return(a + b*t);
}

/* get mid point of formant trajectory for a segment */
function var measure_linear(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;       /* time */
   var at[1:1000];    /* array of time */
   var af[1:1000];    /* array of values */
   var    nf;       /* # values */

   /* calculate trajectory */
   nf=0;
   t=next(FM,stime);
   while ((t < etime)&&(nf < 1000)) {
      nf = nf+1;
      at[nf] = t;
      af[nf] = fm(pno,t);
      t = next(FM,t);
   }

   return(lsqfit(at,af,nf,(stime+etime)/2));
}
```

Method 6. Find quadratic of best fit

Finally, we refine the last approach by fitting a quadratic rather than a straight line to the formant values. This accommodates the fact that formant trajectories are often curved through a segment. A possible disadvantage is that we may become over sensitive to outliers. Here are the modifications needed:

```
/* calculate last-squares fit of quadratic and return value at time */
function var quadfit(at,af,nf,t)
{
   var    at[];    /* array of times */
   var    af[];    /* array of frequencies */
   var    nf;       /* # values */
   var    t;       /* output time */
   var    i;
   var    a,b,c;    /* coefficients */
   var    mat1[1:4];    /* normal matrix row 1 */
   var    mat2[1:4];    /* normal matrix row 2 */
   var    mat3[1:4];    /* normal matrix row 3 */

   /* collect parameters */
   for (i=1;i<=nf;i=i+1) {
      mat1[1] = mat1[1] + 1;
      mat1[2] = mat1[2] + at[i];
      mat1[3] = mat1[3] + at[i] * at[i];
      mat1[4] = mat1[4] + af[i];
      mat2[1] = mat2[1] + at[i];
      mat2[2] = mat2[2] + at[i] * at[i];
      mat2[3] = mat2[3] + at[i] * at[i] * at[i];
      mat2[4] = mat2[4] + at[i] * af[i];
      mat3[1] = mat3[1] + at[i] * at[i];
      mat3[2] = mat3[2] + at[i] * at[i] * at[i];
      mat3[3] = mat3[3] + at[i] * at[i] * at[i] * at[i];
      mat3[4] = mat3[4] + at[i] * at[i] * af[i];
   }

   /* reduce lines 2 and 3, column 1 */
   for (i=4;i>=1;i=i-1) {
      mat2[i] = mat2[i] - mat1[i]*mat2[1]/mat1[1];
      mat3[i] = mat3[i] - mat1[i]*mat3[1]/mat1[1];
   }

   /* reduce line 3 column 2 */
   for (i=4;i>=2;i=i-1) {
      mat3[i] = mat3[i] - mat2[i]*mat3[2]/mat2[2];
   }

   /* calculate c */
   c = mat3[4]/mat3[3];

   /* back substitute to get b */
   b = (mat2[4] - mat2[3]*c)/mat2[2];

   /* back substitute to get a */
   a = (mat1[4] - mat1[3]*c - mat1[2]*b)/mat1[1];

   return(a + b*t + c*t*t);
}

/* get mid point of formant trajectory for a segment */
function var measure_quadratic(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;       /* time */
   var at[1:1000];    /* array of time */
   var af[1:1000];    /* array of values */
   var    nf;       /* # values */
```

In the next section we will apply these approaches to the study of the distribution of formant values for a particular segment type for a single speaker, and investigate which gives us the least variable results.

Average within a speaker

So far we have shown a number of ways in which to extract a characteristic formant frequency value from each annotated region of the signal. In this section we will look at the distribution of those values for a number of instances of a single type of annotated region for a single speaker. This will not only demonstrate how to collect data across a number of files, but it will also allow us to make a simple empirical study of the performance of the six different methods. Attention aimed at obtaining the most accurate method may reap particular benefits in disordered speech where the formant values estimated by different methods may be more variable than with fluent speakers.

The script below calls the measure_mean() function on all instances of a given labeled region found in the input files. It then collects the values into a histogram and plots the histogram and a modelled normal distribution for each formant. It also reports the mean and standard deviation of the estimated characteristic formant frequencies for the segment.

```
/* fmplot1.sml -- plot distribution of formant frequency averages */
/* - uses mean over whole annotated region */

stat    f1;          /* f1 distribution */
stat    f2;          /* f2 distribution */
stat    f3;          /* f3 distribution */
var     hf1[0:100];    /* f1 histogram (50Hz bins) */
var     hf2[0:100];    /* f2 histogram (50Hz bins) */
var     hf3[0:100];    /* f3 histogram (50Hz bins) */

string    label;    /* annotation label to measure */
file    gop;    /* graphics output */

/* get mean formant value for a segment */
function var measure_mean(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;        /* time */
   var    sum;    /* sum of values */
   var    cnt;    /* # values */

   /* calculate mean over whole segment */
   sum=0;
   cnt=0;
   t=next(FM,stime);
   while (t < etime) {
      sum = sum + fm(pno,t);
      cnt = cnt + 1;
      t = next(FM,t);
   }

   if (cnt > 0) return(sum/cnt) else return(ERROR);
}

/* normal distribution */
function var normal(st,x)
stat st;
{
   var x;
   x = x - st.mean;
   return(exp(-0.5*x*x/st.variance)/sqrt(2*3.14159*st.variance));
}

/* plot histogram overlaid with normal distribution */
function var plotdist(st,hs)
stat st;
var hs[];
{
   var    i;
   var    xdata[1:2];
   var ydata[0:4000];

   /* set up x-axes */
   xdata[1]=0;
   xdata[2]=4000;
   plotxdata(xdata,1)

   /* plot histogram */
   plotparam("type=hist");
   for (i=0;i<=80;i=i+1) ydata[i] = hs[i]/st.count;
   plot(gop,1,ydata,81);
```

We will run this script over 200 phonetically annotated sentences that form part of the SCRIBE corpus. We will just look at the distribution of the formant frequencies among 65 instances of /A:/ in those sentences. The graphical output of the script is shown in Figure D.4.3.

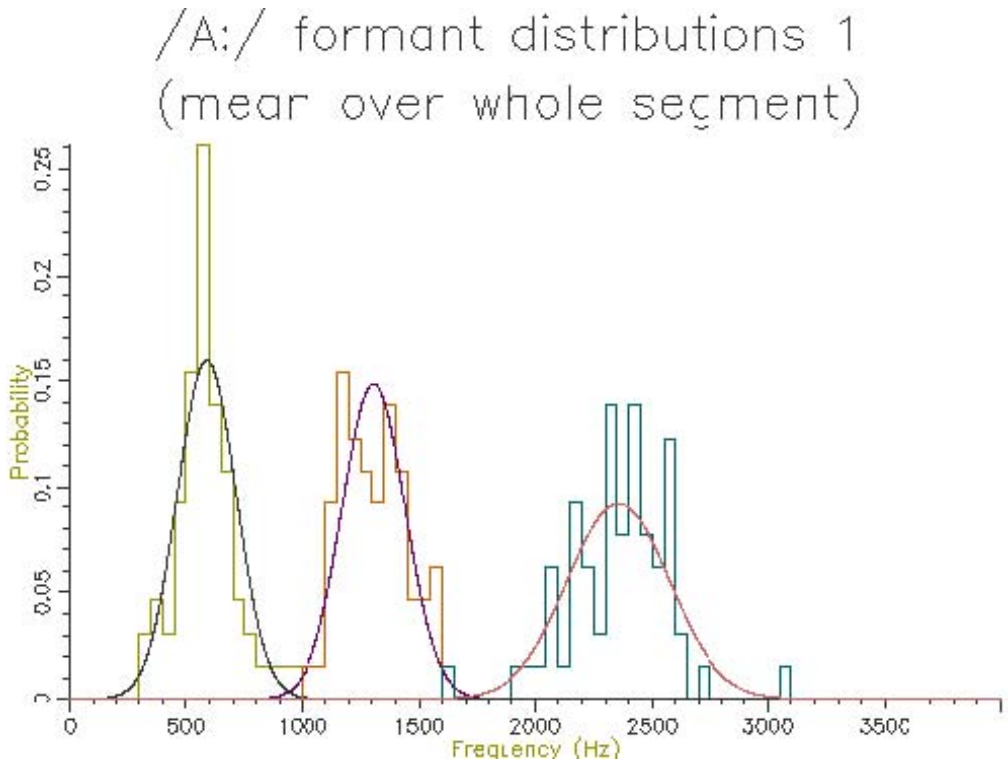


Figure D.4.3 - Analysis of 65 /A:/ vowels, method 1

This figure shows quite clearly the breadth of the formant frequency distributions even when all the vowels are from the same speaker. The estimated characteristic formant frequencies for /A:/ for this speaker are also output by the script:

F1 = 579.2066 +/- 112.6763Hz (65)
F2 = 1278.4254 +/- 127.9527Hz (65)
F3 = 2332.7904 +/- 210.0033Hz (65)

Figures D.4.4 to D.4.8 show the output of similar scripts set up to use each of the other methods described in the last section

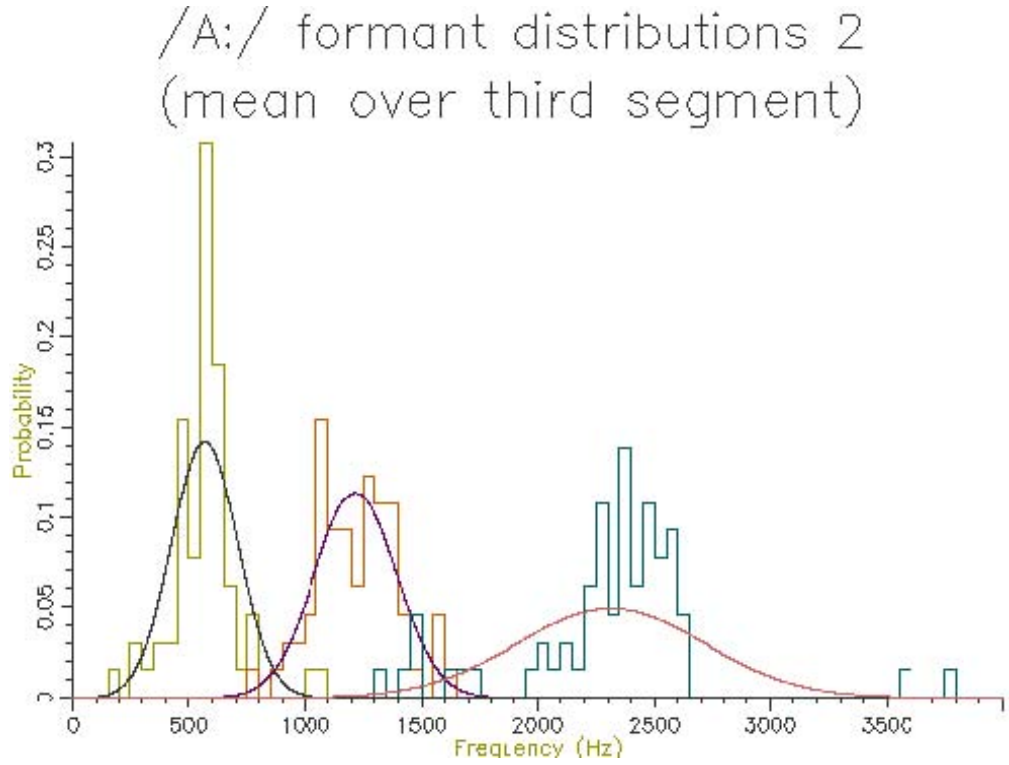


Figure D.4.4 - Analysis of 65 /A:/ vowels, method 2

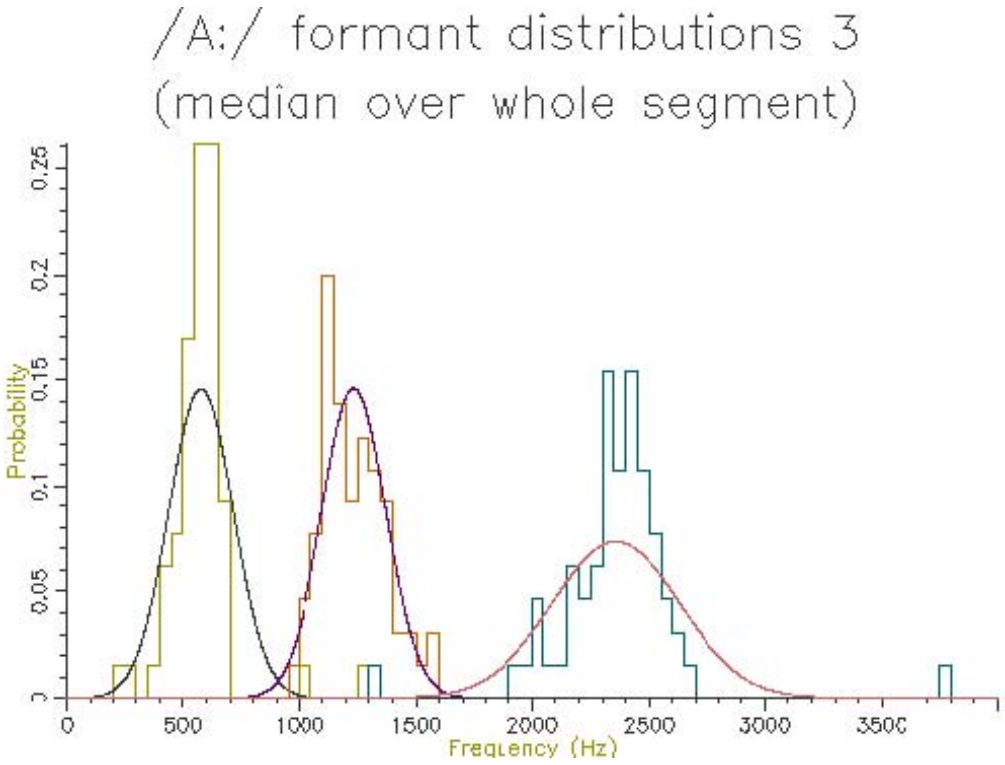


Figure D.4.5 - Analysis of 65 /A:/ vowels, method 3

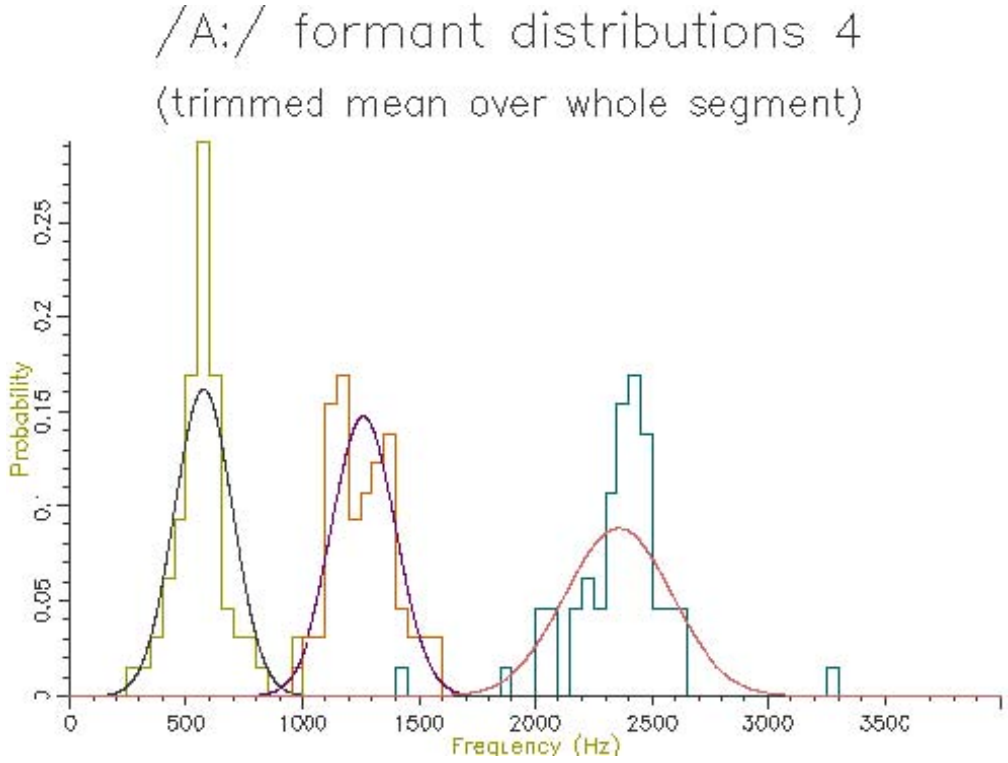


Figure D.4.6 - Analysis of 65 /A:/ vowels, method 4

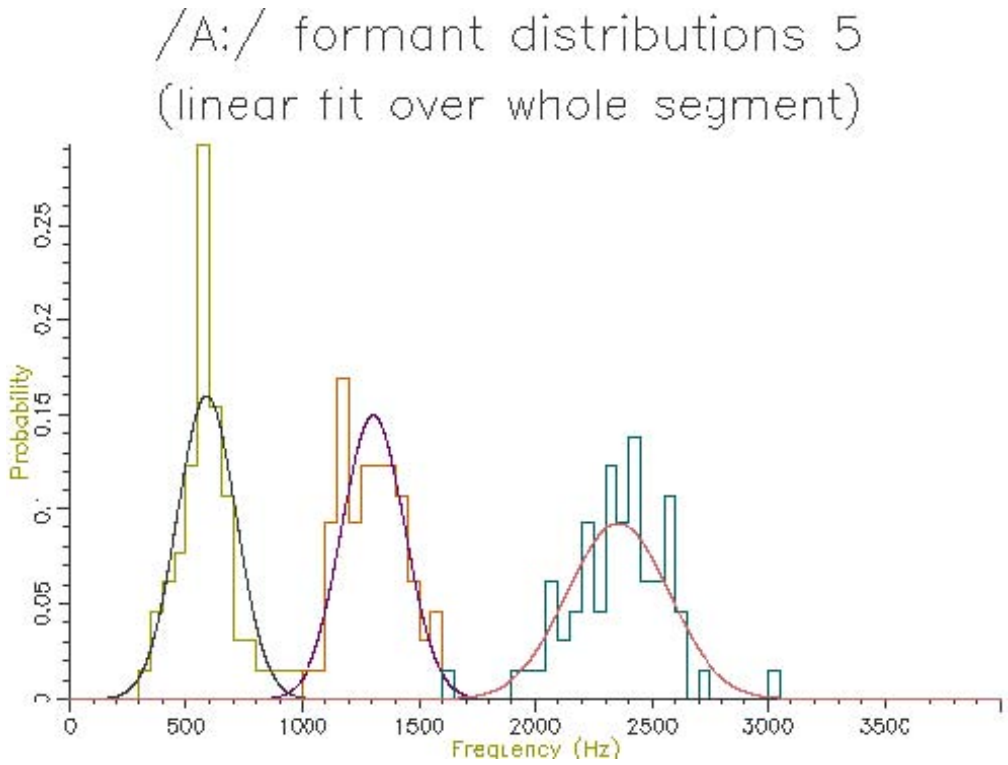


Figure D.4.7 - Analysis of 65 /A:/ vowels, method 5

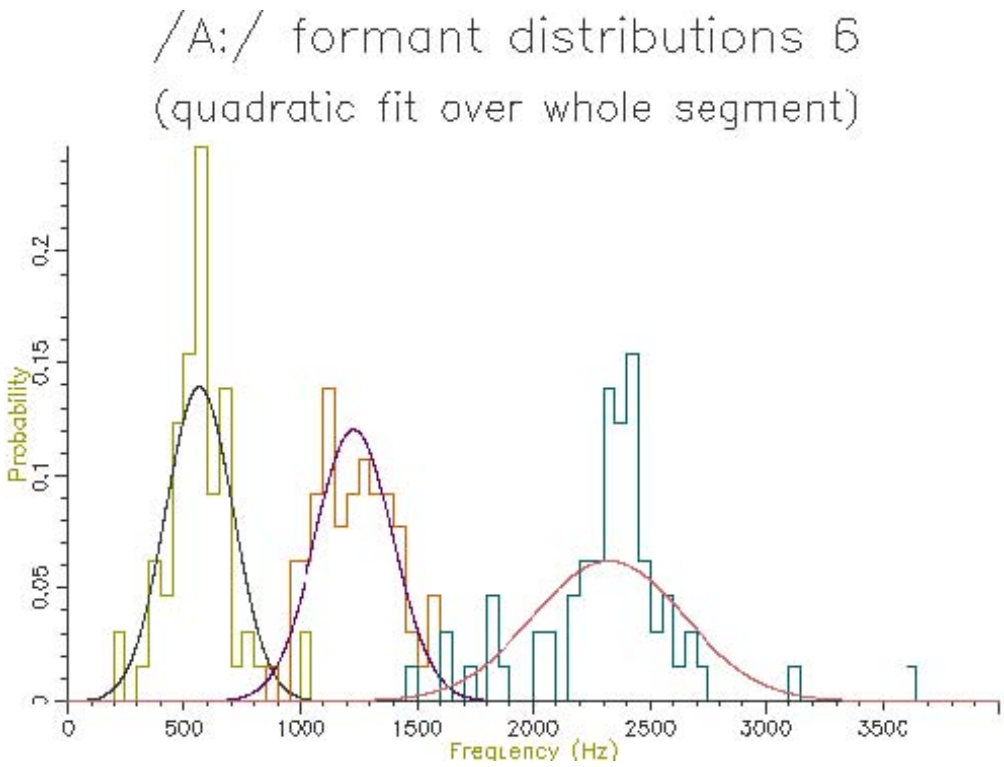


Figure D.4.8 - Analysis of 65 /A:/ vowels, method 6

The table below shows the mean and standard deviation of the characteristic formant frequency for each formant for each of the analysis methods:

Method	F1		F2		F3	
	Mean (Hz)	Dev (Hz)	Mean (Hz)	Dev (Hz)	Mean (Hz)	Dev (Hz)
1. Mean whole	588.6	125.1	1306.2	134.3	2356.0	217.7
2. Mean third	567.8	140.4	1216.6	175.6	2314.1	404.6
3. Median	576.9	136.9	1234.2	136.7	2357.8	272.2
4. Trimmed mean	575.3	123.4	1263.0	135.3	2357.2	227.2
5. Fit line	588.2	125.0	1305.0	133.6	2355.1	217.0
6. Fit quadratic	565.9	143.3	1230.5	165.6	2324.0	323.1

Looking at the table, the lowest variance for F1 comes through using the trimmed mean, while the lowest variance for F2 and F3 comes from the straight line fit. Which is the best method? The problem is choosing a method that is robust to the typical errors in formant estimation. The trimmed mean seems a simple and robust measure (at least for /A:/).

When studying stammered speech, the differences may be subtle, so care should be

taken to use the most accurate procedure. The next section considers requirements for representing audio data across groups of speakers.

Speaker Normalisation

So far we have looked at collecting formant measurements within a segment and across copies of a segment within one speaker. The data in Appendix A come from speakers who are heterogeneous in gender, age and accent. In this section we look at the problems of collecting formant measurements across speakers. The biggest challenge we face is the standardisation of the range of formant values for each speaker prior to averaging across speakers. Because speakers are of physically different sizes, the absolute value of their formant frequencies will vary because of their size as well as because of any change in accent or style.

We will only look at a simple means for standardising or normalising formant frequencies. As well as collecting formant measurements from a collection of recordings of a speaker for a single segment type, we will also collect measurements for all related types for the speaker. We can then represent the characteristic formant frequencies for a segment for a speaker in terms of their relationship to the overall distribution of frequencies for the speaker.

To demonstrate the idea we will first show how to collect segment specific and general measurements for vowels from a number of annotated recordings of a single speaker, delivering a normalised formant estimate. We will use the trimmed mean to get a characteristic value for each segment.

```
/* fmnorm.sml – calculate normalised formant measurements for segment */

/* global distribution */
stat    gf1,gf2,gf3;

/* segment specific distribution */
stat    sf1,sf2,sf3;

/* label to find */
string    label;

/* calculate trimmed mean */
function var trimmean(table,len)
{
   var table[];    /* array of values */
   var len;        /* # values */
   var    i,j,tmp;
   var    lo,hi;

   /* sort table */
   for (i=2;i<=len;i=i+1) {
      j = i;
      tmp = table[j];
      while (table[j-1] > tmp) {
         table[j] = table[j-1];
         j = j - 1;
         if (j==1) break;
      }
      table[j] = tmp;
   }

   /* find mean over middle portion */
   lo = trunc(0.5 + 1 + len/5);    /* lose bottom 20% */
   hi = trunc(0.5 + len - len/5);    /* lose top 20% */
   j=0;
   tmp=0;
   for (i=lo;i<=hi;i=i+1) {
      tmp = tmp + table[i];
      j = j + 1;
   }
   if (j > 0) return(tmp/j) else return(ERROR);
}

/* get trimmed mean formant value for a segment */
function var measure_trimmed_mean(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;        /* time */
   var af[1:1000];    /* array of values */
   var    nf;        /* # values */

   /* calculate trimmed mean */
   nf=0;
   t=next(FM,stime);
   while ((t < etime)&&(nf < 1000)) {
      nf = nf+1;
      af[nf] = fm(pno,t);
      t = next(FM,t);
   }

   return(trimmean(af,nf));
```

In this script we collect the mean value of F1, F2 and F3 for a single segment type, and also collect the mean and variance of F1, F2 and F3 over all vowel segments. We then express F1, F2 and F3 for the given segment as *z-score* positions of the segment mean with respect to the mean and variance of all vowels. The table below shows the output of the script over 200 sentences spoken by one person for two different vowels:

/A:/ vowel

Speaker means: (1847 segments)
F1 = 426.0644 +/- 151.5595Hz
F2 = 1589.4008 +/- 314.8747Hz
F3 = 2496.9879 +/- 239.9074Hz
Segment means: (65 segments)
F1 = 575.2579 +/- 123.4225Hz
F2 = 1263.0107 +/- 135.3098Hz
F3 = 2357.1892 +/- 227.2396Hz
Normalised segment means: (65 segments)
F1 = 0.9844 z-score
F2 = -1.0366 z-score
F3 = -0.5827 z-score

/i:/ vowel

Speaker means: (1847 segments)
F1 = 426.0644 +/- 151.5595Hz
F2 = 1589.4008 +/- 314.8747Hz
F3 = 2496.9879 +/- 239.9074Hz
Segment means: (192 segments)
F1 = 336.3615 +/- 83.0778Hz
F2 = 1936.3488 +/- 211.8316Hz
F3 = 2621.4353 +/- 209.6236Hz
Normalised segment means: (192 segments)
F1 = -0.5919 z-score
F2 = 1.1019 z-score
F3 = 0.5187 z-score

We can now apply this idea to compare formant frequencies across speakers. In this demonstration we will plot the mean F1 and F2 for 5 long monophthongs (/i:/, /u:/, /3:/, /A:/, /O:/) over 10 male and 10 female speakers of a single accent. We will do this first without normalisation, then with normalisation.

Note: in the data used for this demonstration, speakers can be identified from the filename: the first 8 characters of the filename are specific to the speaker. We will use this to collect the F1 and F2 data on a speaker-dependent basis. Also this database is labelled with ARPABET symbols, not JSRU or SAMPA.

```
/* f1f2speaker.sml – F1-F2 diagram for vowels across speakers */

/* list of speakers */
string    stab[1:100];
var    scnt;

/* general vowel distributions */
stat    gf1[1:100];
stat    gf2[1:100];

/* specific vowel distributions */
stat    s1f1[1:100],s1f2[1:100];
stat    s2f1[1:100],s2f2[1:100];
stat    s3f1[1:100],s3f2[1:100];
stat    s4f1[1:100],s4f2[1:100];
stat    s5f1[1:100],s5f2[1:100];

/* output file */
file    gop;

/* function to find speaker code from filename */
function var speakercode(name)
{
   var    code;
   string name;

   name = name:8;    /* first eight characters */
   code = entry(name,stab);
   if (code) return(code);
   scnt=scnt+1;
   stab[scnt]=name;
   return(scnt);
}

/* calculate trimmed mean */
function var trimmean(table,len)
{
   var table[];    /* array of values */
   var len;        /* # values */
   var    i,j,tmp;
   var    lo,hi;

   /* sort table */
   for (i=2;i<=len;i=i+1) {
      j = i;
      tmp = table[j];
      while (table[j-1] > tmp) {
         table[j] = table[j-1];
         j = j - 1;
         if (j==1) break;
      }
      table[j] = tmp;
   }

   /* find mean over middle portion */
   lo = trunc(0.5 + 1 + len/5);    /* lose bottom 20% */
   hi = trunc(0.5 + len - len/5);    /* lose top 20% */
   j=0;
   tmp=0;
   for (i=lo;i<=hi;i=i+1) {
      tmp = tmp + table[i];
      j = j + 1;
   }
```

The outputs of the script on 10 male and 10 female speakers are shown in Figures D.4.9 and D.4.10. The normalised results show considerably less variation across speakers and also less overlap across segment types. In the next section we will look at how we can perform statistical comparisons on these kind of data across speakers and types.

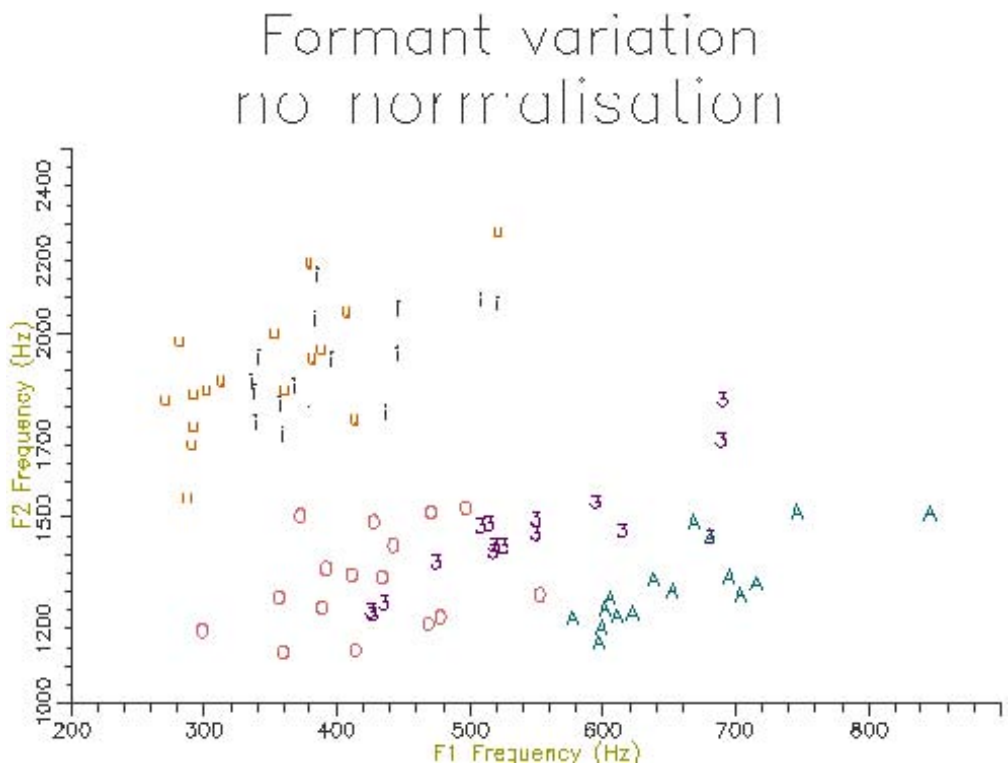


Figure D.4.9 - Vowel analysis without normalization

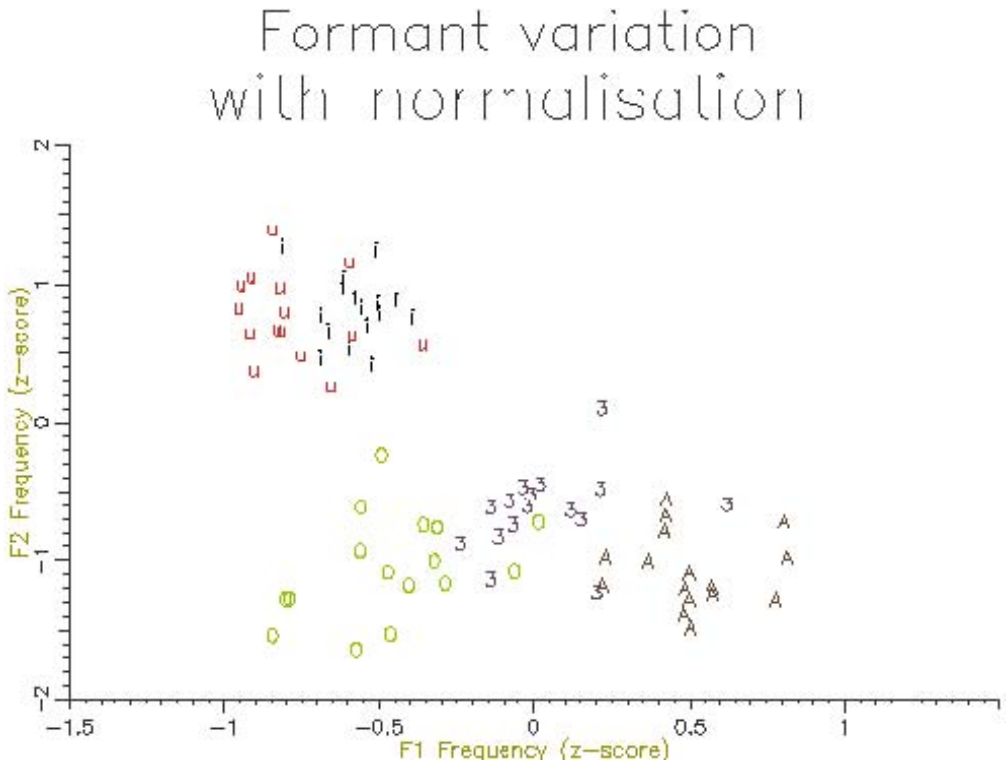


Figure D.4.10 - Vowel analysis with normalization

5. Data Analysis

We have collected together a range of tools for measuring formant frequencies from annotated speech signals. In this section we will look at how we might use those tools to establish the likelihood of various experimental hypotheses using techniques of *inferential statistics*.

+Comparison of means (1D)

As can be seen from the formant frequency distributions shown in Figures D.4.3 to D.4.8, typical formant distributions for a single segment show a fairly normal shape. Thus it is reasonable for us to use a parametric method for comparing the means of two samples. We will show this kind of analysis through a number of worked example cases.

Is a vowel the same in two different contexts?

In this example we look at some instances of /i:/ vowels spoken in word-final and word-medial positions. We can use one of the scripts from section D.4 (e.g. fmsummary4.sml) to extract this data from annotated recordings of a single speaker. Here we have divided it into two sets according to the context in which each vowel occurs:

Word final

"sse_f_02_02.sfs","iy/we", 418.8033, 1774.0710, 2394.6035
"sse_f_02_03.sfs","iy/security", 627.6675, 1651.4409, 2598.3047
"sse_f_02_04.sfs","iy/be", 463.7514, 1803.9850, 2568.7997
"sse_f_02_05.sfs","iy/be", 659.1691, 2073.7795, 2667.1369
"sse_f_02_14.sfs","iy/agency", 577.9793, 2376.8602, 2939.6063
"sse_f_02_16.sfs","iy/Gary", 538.9950, 2037.0008, 2349.4858
"sse_f_02_19.sfs","iy/tea", 572.1907, 2190.4421, 2744.6243

Word medial

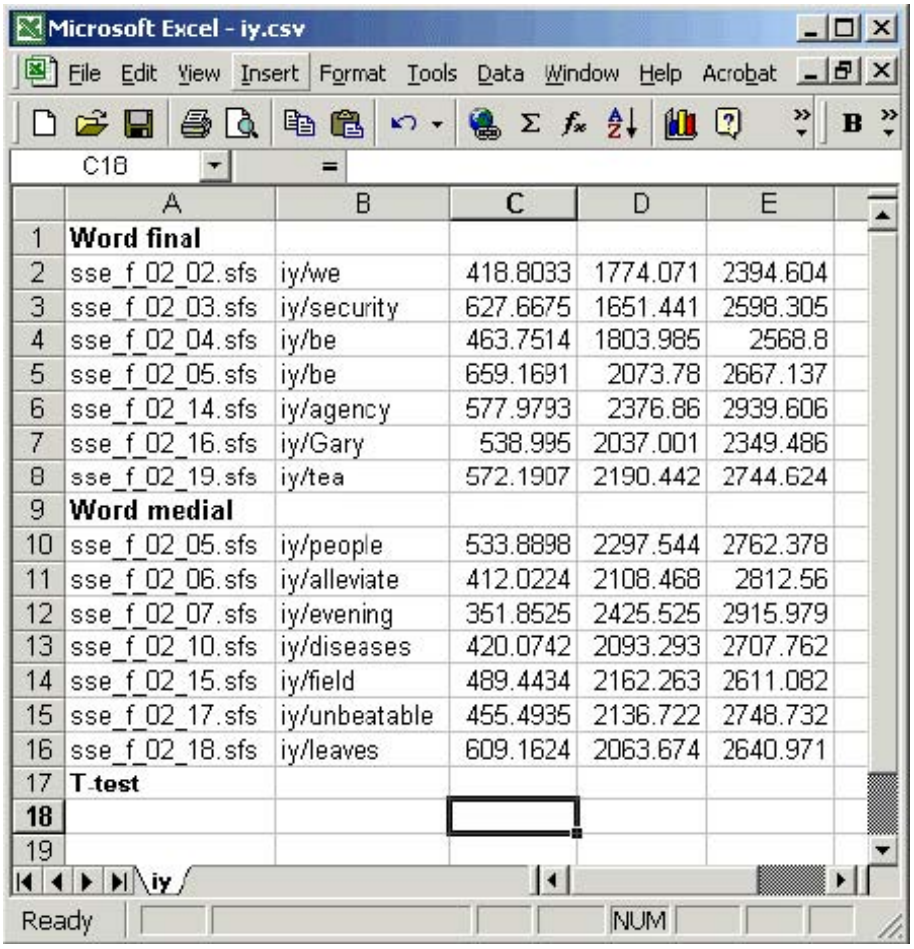
"sse_f_02_05.sfs","iy/people", 533.8898, 2297.5439, 2762.3784
"sse_f_02_06.sfs","iy/alleviate", 412.0224, 2108.4677, 2812.5599
"sse_f_02_07.sfs","iy/evening", 351.8525, 2425.5254, 2915.9789
"sse_f_02_10.sfs","iy/diseases", 420.0742, 2093.2929, 2707.7623
"sse_f_02_15.sfs","iy/field", 489.4434, 2162.2629, 2611.0818
"sse_f_02_17.sfs","iy/unbeatable",455.4935, 2136.7220, 2748.7320
"sse_f_02_18.sfs","iy/leaves", 609.1624, 2063.6738, 2640.9705

A reasonable question to ask is whether there is a systematic difference in the formant frequencies of /i:/ across these two contexts (certainly F1 looks a bit higher and F2 looks a bit lower in word final position). For these data, the question we are asking is how likely it is that these two samples would have their means if they were really just two samples of the same underlying population of vowels. Thus the null hypothesis is that the observed variation in sample means is due to chance. If it turns out that the difference in means is unlikely just to be due to chance, then we can say that it is likely that there is a real effect.

To obtain the likelihood that a difference in sample means arose by chance we can simulate drawing samples of appropriate size from a single population and find out what proportion have a difference in means as large as the difference we observe in our data. This is just the calculation that is performed by the t-test.

We could perform a t-test on these data using a statistics package, but here we will just use the Excel spreadsheet program (the OpenOffice Calc spreadsheet has the same functions). You may need to install the optional "Analysis ToolPak" (sic) to get the statistical functions.

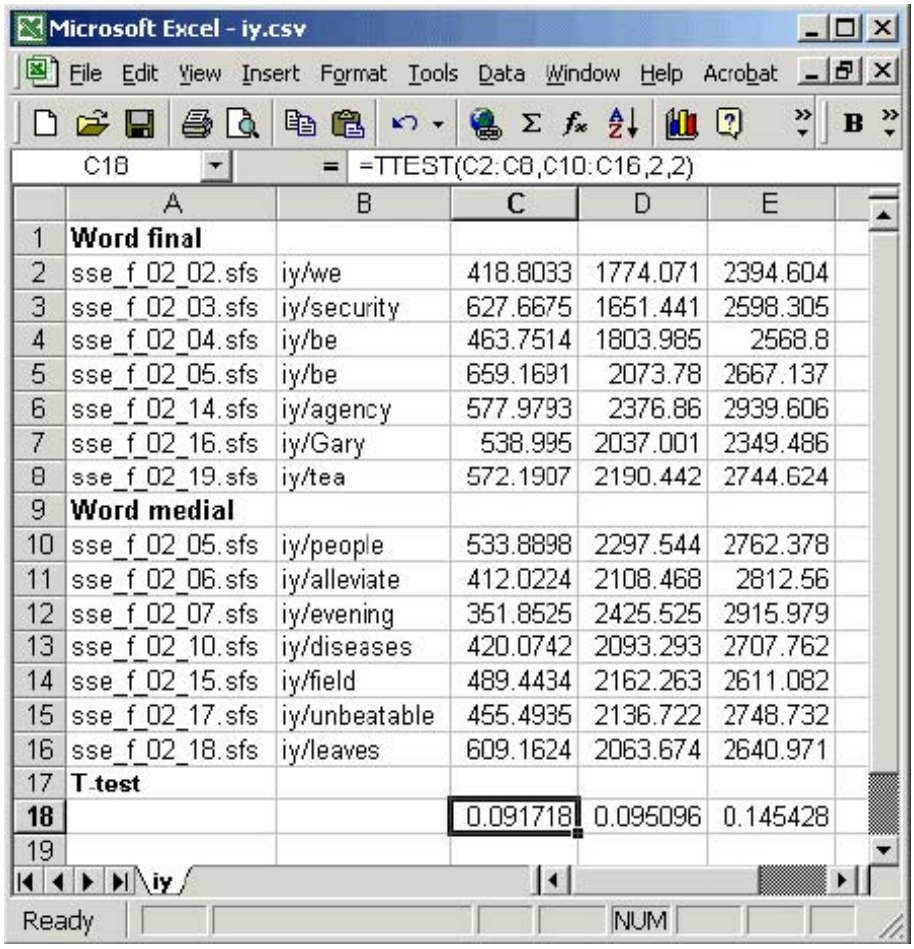
Figure D.5.1 shows the data in Excel, ready for the t-test values to be calculated:



	A	B	C	D	E
1	**Word final**				
2	sse_f_02_02.sfs	iy/we	418.8033	1774.071	2394.604
3	sse_f_02_03.sfs	iy/security	627.6675	1651.441	2598.305
4	sse_f_02_04.sfs	iy/be	463.7514	1803.985	2568.8
5	sse_f_02_05.sfs	iy/be	659.1691	2073.78	2667.137
6	sse_f_02_14.sfs	iy/agency	577.9793	2376.86	2939.606
7	sse_f_02_16.sfs	iy/Gary	538.995	2037.001	2349.486
8	sse_f_02_19.sfs	iy/tea	572.1907	2190.442	2744.624
9	**Word medial**				
10	sse_f_02_05.sfs	iy/people	533.8898	2297.544	2762.378
11	sse_f_02_06.sfs	iy/alleviate	412.0224	2108.468	2812.56
12	sse_f_02_07.sfs	iy/evening	351.8525	2425.525	2915.979
13	sse_f_02_10.sfs	iy/diseases	420.0742	2093.293	2707.762
14	sse_f_02_15.sfs	iy/field	489.4434	2162.263	2611.082
15	sse_f_02_17.sfs	iy/unbeatable	455.4935	2136.722	2748.732
16	sse_f_02_18.sfs	iy/leaves	609.1624	2063.674	2640.971
17	**T-test**				
18					
19					

Figure D.5.1 - Ready for t-test calculation in Excel

To enter the calculation for a t-test, use the TTEST(array1,array2,tails,type) function: array1 is the column of F1 values for word final, array2 is the column of F1 values for word medial, tails is 2 for a test in which we don't know whether the frequencies should be higher or lower, type is 2 for when we believe that both sets have the same variance. Figure D.5.2 shows the data in Excel, with the formula entered and copied under the F2 and F3 columns.



	A	B	C	D	E
1	**Word final**				
2	sse_f_02_02.sfs	iy/we	418.8033	1774.071	2394.604
3	sse_f_02_03.sfs	iy/security	627.6675	1651.441	2598.305
4	sse_f_02_04.sfs	iy/be	463.7514	1803.985	2568.8
5	sse_f_02_05.sfs	iy/be	659.1691	2073.78	2667.137
6	sse_f_02_14.sfs	iy/agency	577.9793	2376.86	2939.606
7	sse_f_02_16.sfs	iy/Gary	538.995	2037.001	2349.486
8	sse_f_02_19.sfs	iy/tea	572.1907	2190.442	2744.624
9	**Word medial**				
10	sse_f_02_05.sfs	iy/people	533.8898	2297.544	2762.378
11	sse_f_02_06.sfs	iy/alleviate	412.0224	2108.468	2812.56
12	sse_f_02_07.sfs	iy/evening	351.8525	2425.525	2915.979
13	sse_f_02_10.sfs	iy/diseases	420.0742	2093.293	2707.762
14	sse_f_02_15.sfs	iy/field	489.4434	2162.263	2611.082
15	sse_f_02_17.sfs	iy/unbeatable	455.4935	2136.722	2748.732
16	sse_f_02_18.sfs	iy/leaves	609.1624	2063.674	2640.971
17	**T-test**				
18			0.091718	0.095096	0.145428
19					

Figure D.5.2 - t-test calculation in Excel

What is the interpretation of the t-test probabilities? The t-test shows us that for each formant there is about a 1 in 10 chance (p about 0.1) that the difference in the sample means could have arisen even if they were actually from the same underlying population. This is not good evidence that there is a real effect here: we would expect the difference in observed means once in every ten experiments even if there were no effect to measure.

In this demonstration there was no obvious connection between the vowel contexts in each of the two sets, they were just two random samples of words in the recording. If we had planned the recording more carefully we could have constructed *paired contexts*, so that one of the pair would give us values in the first set, and one would give us a value in the second. For example we might have words such as "happy/happiness", "silly/sillyness" where we could compare the vowel formant frequencies with and without an additional affix. This is called a "paired" test and is fundamentally more sensitive than the independent samples test we reported above. In a paired test you are only looking for a systematic difference between members of each pair rather than a difference of means across the sets. In effect you are looking at the distribution of the hertz difference between the members of the pair, and the t-test establishes whether the mean of that difference across all pairs is significantly different from zero.

Comparison of means (2D)

A problem with the analysis of the last section is that we analysed separately any difference in the means of F1 and F2 and F3. If you think about it you can see that greater the number of parameters we check the more likely it is that we will find a low-probability random fluctuation. In other words we cannot just use a probability of 0.05 (say) to identify a significant event if we then apply the same significance separately to multiple parameters. The use of multiple parameters obliges us to look for a greater level of significance.

Another problem with treating F1 and F2 separately is that there may be a real effect which makes a small difference to **both** F1 and F2, but which is not significant when we test either separately.

Generally we need to consider a method to compare F1 and F2 together. First we'll look at graphing two samples on the F1-F2 plane, plotting a contour expressing one standard deviation in two-dimensions.

The following script collects F1 and F2 frequencies from the five long monophthongs from a single speaker. It then plots these on a scatter graph and estimates the shape and size of an ellipse that characterises the distribution of values.

```
/* f1f2distributions.sml – collect and plot F1-F2 distributions */

/* raw data tables - one per vowel type */
var       t1f1[1:1000],t1f2[1:1000],t1cnt;
var       t2f1[1:1000],t2f2[1:1000],t2cnt;
var       t3f1[1:1000],t3f2[1:1000],t3cnt;
var       t4f1[1:1000],t4f2[1:1000],t4cnt;
var       t5f1[1:1000],t5f2[1:1000],t5cnt;

/* output file */
file    gop;

/* calculate trimmed mean */
function var trimmean(table,len)
{
   var table[];    /* array of values */
   var len;        /* # values */
   var    i,j,tmp;
   var    lo,hi;

   /* sort table */
   for (i=2;i<=len;i=i+1) {
      j = i;
      tmp = table[j];
      while (table[j-1] > tmp) {
         table[j] = table[j-1];
         j = j - 1;
         if (j==1) break;
      }
      table[j] = tmp;
   }

   /* find mean over middle portion */
   lo = trunc(0.5 + 1 + len/5);    /* lose bottom 20% */
   hi = trunc(0.5 + len – len/5);    /* lose top 20% */
   j=0;
   tmp=0;
   for (i=lo;i<=hi;i=i+1) {
      tmp = tmp + table[i];
      j = j + 1;
   }
   if (j > 0) return(tmp/j) else return(ERROR);
}

/* get trimmed mean formant value for a segment */
function var measure_trimmed_mean(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;        /* time */
   var af[1:1000];    /* array of values */
   var    nf;        /* # values */

   /* calculate trimmed mean */
   nf=0;
   t=next(FM,stime);
   while ((t < etime)&&(nf < 1000)) {
      nf = nf+1;
      af[nf] = fm(pno,t);
      t = next(FM,t);
   }
```

Figure D.5.3 shows the output of the script. These data seem rather noisy.

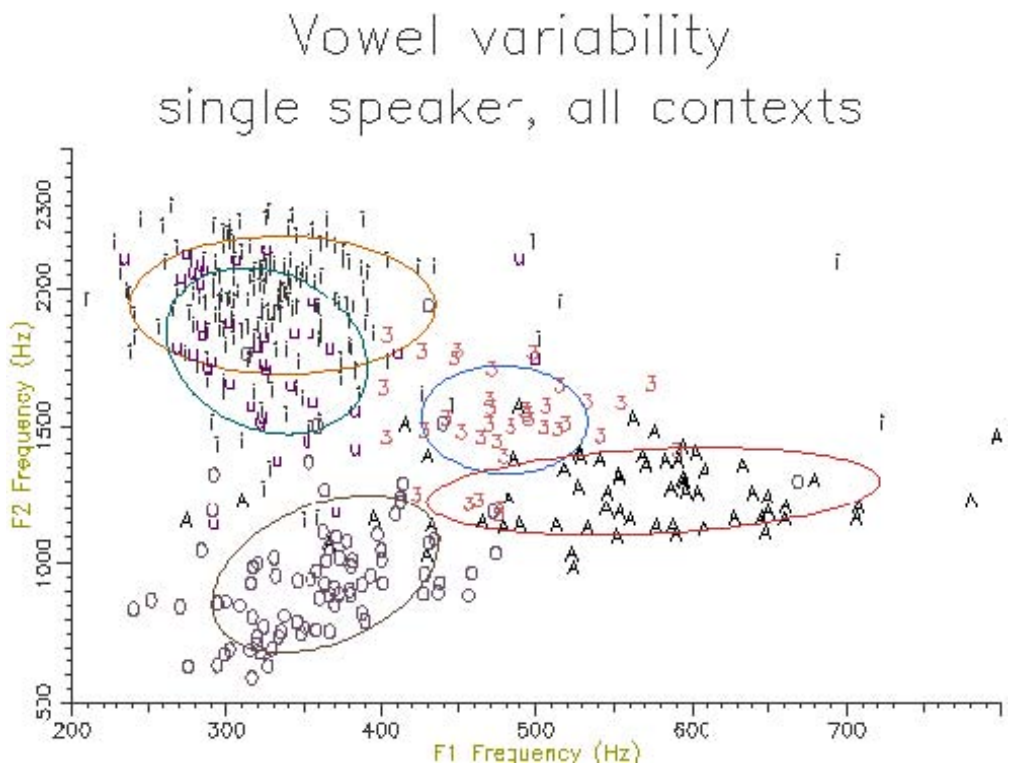


Figure D.5.3 - Vowel distribution on F1-F2 plane

Comparing the centroids of two vowel samples on the F1-F2 plane

Looking carefully at Figure D.5.3 you will see that there is a large overlap in the distributions of /i:/ and /u:/ on the F1-F2 plane. We will now consider how we might perform a statistical test that determines whether these two samples are really distinct or come from the same underlying population (we are pretty sure that /i:/ and /u:/ are different, of course!). Rather than use two t-tests on the F1 values and the F2 values separately, we will show how to perform a single test that uses F1 and F2 *jointly*.

A statistical test on one random variable is called a "univariate" test, while a statistical test on 2 or more variables is called "multivariate". In this case we need to use the multivariate equivalent of the t-test, and this is called "Hotelling's T^2 test".

The script below collects information about the F1 and F2 distribution of two vowels and calculates the value of Hotelling's T^2 statistic on the two samples.

```
/* f1f2compare.sml - calculate Hotelling's T-squared on vowel pair */

/* vowels to measure */
string    label1;
string    label2;

/* raw data tables - one per vowel type */
var       t1f1[1:1000],t1f2[1:1000],t1cnt;
var       t2f1[1:1000],t2f2[1:1000],t2cnt;

/* calculate trimmed mean */
function var trimmean(table,len)
{
   var table[];    /* array of values */
   var len;        /* # values */
   var    i,j,tmp;
   var    lo,hi;

   /* sort table */
   for (i=2;i<=len;i=i+1) {
      j = i;
      tmp = table[j];
      while (table[j-1] > tmp) {
         table[j] = table[j-1];
         j = j - 1;
         if (j==1) break;
      }
      table[j] = tmp;
   }

   /* find mean over middle portion */
   lo = trunc(0.5 + 1 + len/5);    /* lose bottom 20% */
   hi = trunc(0.5 + len - len/5);    /* lose top 20% */
   j=0;
   tmp=0;
   for (i=lo;i<=hi;i=i+1) {
      tmp = tmp + table[i];
      j = j + 1;
   }
   if (j > 0) return(tmp/j) else return(ERROR);
}

/* get trimmed mean formant value for a segment */
function var measure_trimmed_mean(stime,etime,pno)
{
   var    stime;    /* start time */
   var    etime;    /* end time */
   var    pno;    /* FM parameter # */
   var    t;        /* time */
   var af[1:1000];    /* array of values */
   var    nf;        /* # values */

   /* calculate trimmed mean */
   nf=0;
   t=next(FM,stime);
   while ((t < etime)&&(nf < 1000)) {
      nf = nf+1;
      af[nf] = fm(pno,t);
      t = next(FM,t);
   }

   return(trimmean(af,nf));
}
```

When this script is run on the data used to produce Figure D.5.3 on the vowels /i:/ and /u:/, the result is:

Analysis summary:
Number of files = 30
Number of instances of /i:/ = 192
Number of instances of /u:/ = 39
Hotelling T-squared statistic = 18.80
For significance find probability that F(2,228) > 9.36

From this output you can see that the Hotelling's T^2 statistic is 18.8 for these two vowels. To interpret this number we need to know how often this value would occur for samples of the size we used if there were no underlying difference between these vowels. To turn the T^2 statistic into a probability we can use the fact that its distribution follows the same shape as the F distribution of a particular configuration (basically for a *p*-variate sample of df= *n*, the F statistic is (n-p)/p(n-1) times the T^2 statistic). So in this case, the likelihood of a T^2 of 18.8 is equivalent to the likelihood of an F(2,228) statistic of 9.36. Since in general we are only interested in the significance of the statistic, here are a few critical values from the F-distribution F(2,n):

F statistic	p < 0.05	p < 0.01
F(2,10)	4.1	7.56
F(2,20)	3.49	5.85
F(2,50)	3.19	5.08
F(2,100)	3.10	4.85
F(2,200)	3.06	4.77
F(2,500)	3.04	4.71

Since the likelihood of F(2,200) being greater than 4.77 is less than 0.01, we can be sure that the likelihood of F(2,228) being greater than 9.36 is much less than 0.01. Thus there is a significant difference between these two distributions (phew!).

The procedure above could be extended to work with F1, F2 and F3 if required. But if you wanted to test if 3 or more samples came from the same population (rather than just 2 in this case) you would need to perform a multivariate analysis of variance or MANOVA.

Bibliography

The following have been used in the development of this tutorial:

- Maria Isabel Ribeiro, Gaussian Probability Density Functions: Properties and Error Characterization at http://omni.isr.ist.utl.pt/~mir/pub/probability.pdf
- Hotelling's T_2 test at http://www.itl.nist.gov/div898/handbook/pmc/section5/pmc543.htm
- Critical F Distribution Calculator at http://www.psychstat.smsu.edu/introbook/fdist.htm

Appendix E HTK Hidden Markov modelling toolkit with SFS

This document provides a tutorial introduction to the use of SFS in combination with the Cambridge Hidden Markov modelling toolkit (HTK) for pattern processing of speech signals. The tutorial covers installation, file conversion, phone and phone-class recognition, phonetic alignment, pronunciation variation analysis and, of particular interest for current purposes, automatic dysfluency analysis. The work preceding automatic dysfluency analysis contains material that is essential for understanding this topic and could be used in developments of the recognizer (e.g. pronunciation variant analysis). Some background understanding of the Unix command line interpreter is assumed and basic knowledge of how Hidden Markov models work. This tutorial refers to versions 4.6 and later of SFS with version 3.3 of HTK.

1. Installation, Acquiring and Chunking the audio signal

These installation instructions refer to Windows computers. However most of the tutorial applies to other platforms where HTK and SFS command-line programs can be run under a Unix-like shell program.

Installation of CYGWIN

CYGWIN provides a Unix-like programming environment for Windows computers. This environment will be used in the tutorial so that scripts for processing multiple files using the BASH shell language can be used. This is useful because the shell language is simple yet powerful and runs on many different computing platforms.

CYGWIN can be downloaded from the CYGWIN home page at www.cygwin.com. From there download and run the program setup.exe which manages the installation of CYGWIN. This program first collects information about your nearest CYGWIN distribution site then presents you with a list of components to install. Finally it downloads and installs the selected components. The same program can be used to update your installation as new software versions become available and to add/delete components. The setup program goes through these steps:

1. **Choose installation type**. Choose "Install from Internet" if you have a reliable internet connection. Otherwise choose "Download from Internet" to copy the files onto your computer and then "Install from Local Directory" to install them.
2. **Choose installation directory**. We suggest you leave this as "C:/cygwin" unless you know what you are doing.
3. **Select local package directory**. Put directory (aka "folder") here, where CYGWIN will put the downloaded files before installation. You could enter a temporary directory name. We use "C:/download/cygwin". You may need to make the folders first using Windows Explorer.
4. **Select Internet connection**. Leave as default.
5. **Choose a download site**. Highlight an address in the list that seems to come from your own country. We choose "ftp://ftp.mirror.ac.uk/".
6. **Select packages**. Use this page to investigate what packages (components) are available for download. Many are rather old and obscure elements of the Unix operating system. For the purposes of this tutorial you should download at least the following:
    1. All components in the "Base" category.
    2. Devel|BINUTILS: The GNU assembler, linker and binary utilities
    3. Devel|GCC: C Compiler
    4. Devel|GCC-G++: GCC C++ compiler
    5. Devel|MAKE: the GNU version of the 'make' utility

    But feel free to install any of the other goodies that take your fancy.
7. **Download**. The program then downloads and installs the selected packages.
8. **Installation complete**. Choose both boxes to put a CYGWIN icon on your desktop and put a CYGWIN entry in the Start Menu.

After installation your should see a CYGWIN icon on the desktop and a Start menu option Start|Programs|Cygwin|Cygwin BASH shell. Either of these will start up a command window which provides a Unix-like environment in which we will be demonstrating SFS and HTK.

A good introduction to programming the Unix environment can be found in the old but essential "The Unix Programming Environment" by Kernighan and Pike.

Available at Amazon.co.uk.

It is worth exploring the CYGWIN environment to get used to the way it maps the names of the Windows disks and folders. A folder like "C:\WINDOWS" is referred to as "c:/windows" in CYGWIN, or as "/cygdrive/c/windows". Your home directory in CYGWIN (referred to as "~") will actually be a subdirectory of the windows folder c:\cygwin\home.

Installation of a Text Editor

You will need a suitable text editor for editing scripts and other text files in this tutorial. Our recommendation is **TextPad** which can be downloaded from www.textpad.com. This is a shareware program which requires registration if you use it extensively.

Installation of HTK

The Cambridge University Hidden Markov modelling toolkit (HTK) can be downloaded from htk.eng.cam.ac.uk. To check that you have read the licence conditions, they ask you first to register your name and e-mail address with them. They will then send you a password to use to download the HTK sources from htk.eng.cam.ac.uk/ftp. For the purposes of this tutorial we downloaded http://htk.eng.cam.ac.uk/ftp/beta/HTK-3.3-alpha1.tar.gz into our cygwin home directory. By the time you read this tutorial it is very likely that there will be a new release with a different filename. The CYGWIN command to unpack this is just:

```
$ tar xvfz HTK-3.3-alpha1.tar.gz
```

When unpacked, a sub-directory called "htk" will be created under your home directory. You can now delete the downloaded distribution file.

To build HTK for CYGWIN you first need to set a number of environment variables. We suggest you create a file called "htk.env" in your home directory containing the following:

```
export HTKCF='-O2 –DCYGWIN'
export HTKLF='-o a.out'
export HTKCC='gcc'
export HBIN='..'
export Arch=ASCII
export CPU=cygwin
export PATH=~/htk/bin.cygwin:$PATH
```

Then each time you want to use HTK you can just type

```
$ source htk.env
```

Alternatively you can put these commands in a file ".bash_login" in your home directory so that they will be executed each time you log in.

Unfortunately, as of the date of writing this tutorial, the HTK distribution needs patching before it can be compiled under CYGWIN. These are the follwing edits that you need to make using a text editor:

1. Edit HTKLib/HShell.h and include at the end of the file:

2. #ifdef CYGWIN
3. #include <asm/socket.h>

4. #endif

5. Edit HTKLib/HGraf.null.c and include at the end of the file:

```
/* EXPORT HTextHeight: return the height of s in pixels */
int HTextHeight(char *str)
{
   return 0;
}
```

11. Edit HTKTools/makefile and remove the reference to "-lX11" in the instructions for HSLab:

```
HSLab:  $(hlib)/HTKLib.$(CPU).a HSLab.o
        $(CC) HSLab.o $(HLIBS) -lm $(HTKLF)
        mv a.out $(HBIN)/bin.$(CPU)/HSLab
```

We can now make HTK with the following instructions:

```
$ source htk.env
$ cd ~/htk
$ mkdir bin.cygwin
$ cd HTKLib
$ cp HGraf.null.c HGraf.c
$ make
$ cd ../HTKTools
$ make
$ cd ../HLMLib
$ make
$ cd ../HLMTools
$ make
```

Installation of SFS

As mentioned earlier, SFS can be downloaded from www.phon.ucl.ac.uk/resource/sfs/. Run the installation package and select the option "Add SFS to command-line path" to add the SFS program directory to the search path for programs to run from the command prompt and the CYGWIN shell. You may need to reboot for this change to take effect.

2. Phone-class recognition

In this section we will describe a "warm-up" exercise to show how SFS and HTK can be used together to solve a simple problem. The idea is to demonstrate the software tools rather than to achieve ultimate performance on the task.

We will demonstrate the use of SFS and HTK to build a system that automatically labels an audio signal with annotations which divide the signal into regions of "silence", "voiced speech", and "voiceless speech".

Source data

For this demonstration we will use some annotated data that are part of the SCRIBE corpus (see www.phon.ucl.ac.uk/resource/scribe). These data are interesting because they contain some "acoustic" level annotations - that is phonetic annotation at a finer level of detail than normal. In particular the annotation marks voiced and voiceless regions *within* phonetic segments. An example is shown in Figure E.2.1.

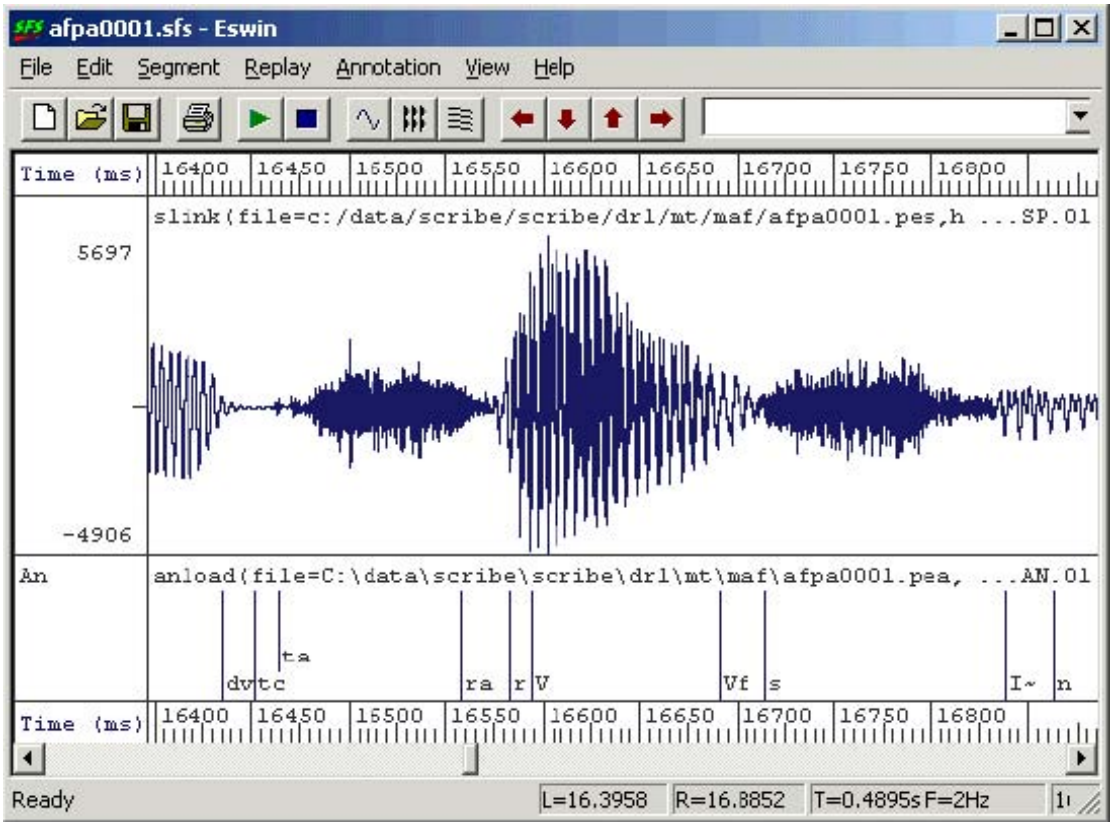


Figure E.2.1 - Acoustic level annotations

The files we will be using are from the 'many talker' sub-corpus and are as follows

Speaker	**Training**		**Testing**	
Signal	**Label**	**Signal**		**Label**
mac	Acpa0002.pes acpa0003.pes acpa0004.pes	Acpa0002.pea acpa0003.pea acpa0004.pea	acpa0001.pes	acpa0001.pea
mae	Aepa0001.pes aepa0003.pes aepa0004.pes	Aepa0001.pea aepa0003.pea aepa0004.pea	aepa0002.pes	aepa0002.pea
maf	Afpa0001.pes afpa0002.pes afpa0004.pes	Afpa0001.pea afpa0002.pea afpa0004.pea	afpa0003.pes	afpa0003.pea
mah	Ahpa0001.pes ahpa0002.pes ahpa0003.pes	Ahpa0001.pea ahpa0002.pea ahpa0003.pea	ahpa0004.pes	ahpa0004.pea
mam			ampa0001.pes ampa0002.pes ampa0003.pes ampa0004.pes	ampa0001.pea ampa0002.pea ampa0003.pea ampa0004.pea

You can see from this table that we have reserved one quarter of each training speaker's recording for testing, and one whole unseen speaker. This means that we can test our parser on material that has not been used for training and also on a speaker that has not been used for training. This should give us a more robust estimate of the

recognizer’s performance.

Loading source data into SFS

We will start by setting up SFS files which point to the SCRIBE data. We will make a new directory in our cygwin directory and run a script which makes the SFS files. The SCRIBE audio files are in a raw binary format at a sampling rate of 20,000 samples/sec. We "link" these into the SFS files rather than waste disk space by copying them. The SCRIBE label files are in SAM format, which the SFS program anload can read (with "-S" switch). The shell script is as follows:

```
# doloadsfs.sh - load scribe data into new SFS files
for s in c e f h m
do
  for f in 0001 0002 0003 0004
  do
    hed -n ma$s.$f.sfs
    slink -isp -f 20000 c:/data/scribe/scribe/dr1/mt/ma$s/a${s}pa$f.pes \
                                          ma$s.$f.sfs
    anload -S c:/data/scribe/scribe/dr1/mt/ma$s/a${s}pa$f.pea ma$s.$f.sfs
  done
done
```

We'd run this in its own subdirectory as follows:

$ mkdir tutorial1
$ cd tutorial1
$ sh doloadsfs.sh

Data Preparation

There are two data preparation tasks: designing and computing a suitable acoustic feature representation of the audio files so they are suitable for the recognition task and mapping the annotation labels into a suitable set of symbols.

For the first task, a simple spectral envelope feature set would seem to be adequate. We will try this first and develop alternatives later. The SFS program voc19 performs a 19-channel filterbank analysis on an audio signal. It consists of 19 band-pass filters spaced on a bark scale from 100 to 4000Hz. The outputs of the filters are rectified, low-passed filtered at 50Hz, resampled at 100 frames/second and finally log-scaled. To run voc19 on all our training and testing data we type:

$ apply voc19 ma*.sfs

For the second task, we are aiming to label the signal with three different labels, according to whether there is silence, voiced speech or voiceless speech. Let us lable these three types as SIL, VOI, UNV. Our annotation preparation task is to map existing annotations to these types. In this case we are not even sure of the inventory of symbols used by the SCRIBE labelers, so we write a script to collect the names of all the different annotations they used:

```
/* ancollect.sml -- collect inventory of labels used */

/* table to hold annotation labels */
string    table[1:1000];
var    tcount;

/* function to check/add label */
function var checklabel(str)
{
   string str;

   if (entry(str,table)) return(0);
   tcount=tcount+1;
   table[tcount]=str;
   return(1);
}

/* for each input file */
main {
   var    i,num;

   num=numberof(".");
   for (i=1;i<=num;i=i+1) checklabel(matchn(".",i));
}

/* output sorted list */
summary {
   var    i,j;
   string    t;

   /* insertion sort */
   for (i=2;i<=tcount;i=i+1) {
      j=i;
      t=table[j];
      while (compare(t,table[j-1])<0) {
         table[j] = table[j-1];
         j=j-1;
         if (j==1) break;
      }
      table[j]=t;
   }

   /* output list */
   for (i=1;i<=tcount;i=i+1) print table[i],"\n";
}
```

To run this script from the CYGWIN shell, we type:

$ sml ancollect.sml ma*.sfs >svumap.txt

We now need to edit the file svumap.txt so as to assign each input annotation with a new SIL, VOI or UNV annotation. Here is the start of the file after editing:

```
# SIL
## SIL
%tc SIL
+ SIL
/ SIL
3: VOI
3:? UNV
3:a UNV
3:af UNV
3:f UNV
3:~ VOI
=l VOI
=lx VOI
=lx? VOI
=lxf UNV
=lxf0 UNV
=m VOI
=mf UNV
=n VOI
...
```

We now translate the SCRIBE labels to the new 3-way classification. We use the SFS anmap program with the "-m" option to collapse adjacent repeated symbols into one instance:

$ apply "anmap -m svumap.txt" ma*.sfs

The result of the mapping can be plainly seen in Figure E.2.2.

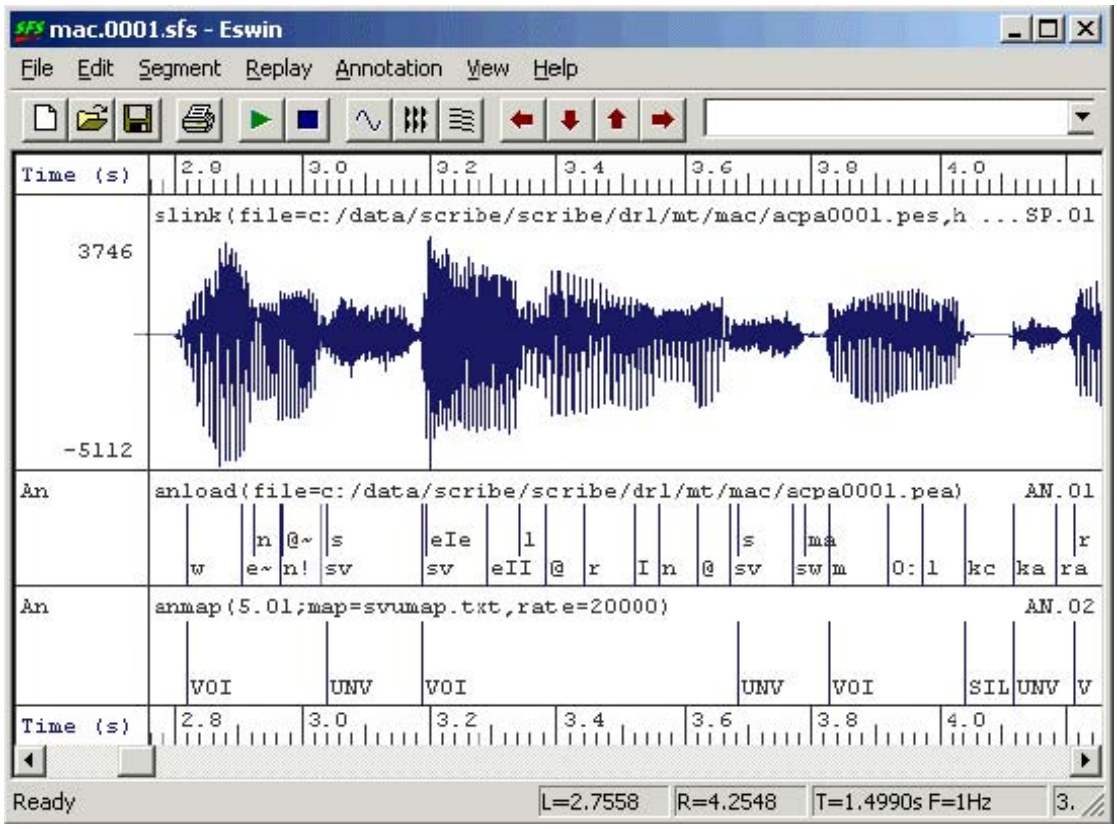


Figure E.2.2 - Mapped annotations

Export of data to HTK format

Unfortunately, HTK cannot (as yet) read SFS files directly, so the next step is to export the data into files formatted so that HTK can read them. Fortunately, SFS knows how to read and write HTK formatted files. To write a set of HTK data files from the voc19 analysis we performed, just type:

$ apply "colist -H" ma*.sfs

This creates a set of HTK format data files with names modelled after the SFS files. Similarly to write a set of HTK format label files, use:

$ apply "anlist -h -O" ma*.sfs

We now have a set of data files ma*.dat and a set of label files ma*.lab in HTK format ready to train some HMMs.

HTK configuration

For training HMMs, HTK requires us to build some configuration files beforehand. The first file is a general configuration of all HTK tools. We will put this into a file called config.txt

```
# config.txt - HTK basic parameters
SOURCEFORMAT = HTK
TARGETKIND = FBANK
NATURALREADORDER = T
```

In this file we specify that the source files are in HTK format, that the training data are already processed into filterbank parameters, and that the data are stored in the natural

byte order for the machine.

The second configuration file we need is a prototype hidden Markov model which we will use to create the models for the three different labels. This configuration file is specific to a one-state HMM with a 19-dimensional observation vector, so we save it in a file called proto-1-19.hmm

<BeginHMM>
<NumStates> 3 <VecSize> 19 <FBANK>
 <State> 2
  <Mean> 19
   0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  <Variance> 19
   1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <TransP> 3
  0.0 1.0 0.0
  0.0 0.9 0.1
  0.0 0.0 1.0
<EndHMM>

It is confusing that an HTK configuration file for a 1-state HMM has 3 states, but the HTK convention is that the first state and the last state are *non-emitting*, that is they are part of the network of HMMs but do not describe any of the input data. Only the middle state of the three emits observation vectors. Briefly this HMM definition file sets up a model with a single state which holds the mean and variance of the 19 filterbank channel parameters. The transition probability matrix simply says that state 1 always jumps to state 2, that state 2 jumps to state 3 with a probability of 1 in 10, and that state 3 always jumps to itself.

HTK training

To train the HMMs we need a file containing a list of all the data files we will use for training. Here we call it train.lst, and it has these contents:

```
mac.0002.dat
mac.0003.dat
mac.0004.dat
mae.0001.dat
mae.0003.dat
mae.0004.dat
maf.0001.dat
maf.0002.dat
maf.0004.dat
mah.0001.dat
mah.0002.dat
mah.0003.dat
```

We can now use the following script to train the HMMs:

```
# dotrain.sh
for s in SIL VOI UNV
do
   cp proto-1-19.hmm $s.hmm
   HRest -T 1 -C config.txt -S train.lst -l $s $s.hmm
Done
```

In this script we copy our prototype HMM into SIL.hmm, VOI.hmm, UNV.hmm and then train these HMMs individually using the portions of the data files that are labeled with SIL, VOI and UNV respectively.

HTK testing

To evaluate how well our HMMs can label an unseen speech signal with the three labels, we recognise the data we reserved for testing and compare the recognised labels with the labels we generated. We will show how to do the recognition in this section, and look at performance evaluation in the next.

To recognise a data file using our trained HMMs we need three further HTK configuration files and a list of test files. The first configuration file we need is just a list of the HMMs; store this in a file called phone.lst:

```
SIL
VOI
UNV
```

The next file we need is a dictionary that maps word pronunciations to a sequence of phone pronunciations. This sounds rather odd in this application, but is necessary because HTK is set up to recognise words rather than phones. Since we are really building a kind of phone recogniser, we solve the problem by having a dictionary of exactly three "words" each of which is "pronounced" by one of the phone labels. Put this into a file called phone.dic:

SIL SIL
VOI VOI
UNV UNV

The last configuration file we need is a recognition grammar that describes which HMM sequences are allowed and what the probability is that each model should follow one of the others. For now, we will just use a default grammar consisting of a simple "phone loop" where the symbols can come in any order and with equal likelihood. This kind of grammar can be built using the HTK program HBuild from the phone list, as follows:

$ HBuild phone.lst phone.net

The resulting grammar file phone.net looks like this:

```
VERSION=1.0
N=7    L=9
I=0    W=!NULL
I=1    W=!NULL
I=2    W=SIL
I=3    W=VOI
I=4    W=UNV
I=5    W=!NULL
I=6    W=!NULL
J=0    S=0    E=1    l=0.00
J=1    S=5    E=1    l=0.00
J=2    S=1    E=2    l=-1.10
J=3    S=1    E=3    l=-1.10
J=4    S=1    E=4    l=-1.10
J=5    S=2    E=5    l=0.00
J=6    S=3    E=5    l=0.00
J=7    S=4    E=5    l=0.00
J=8    S=5    E=6    l=0.00
```

Do not worry too much about this file. It looks more complex than it really is - basically the lines starting with "I" represent nodes of a simple transition network, while the lines starting with "J" represent arcs that run from one node to another and have a transition probability (stored as a log likelihood).

The last thing we need is a list of test filenames; put this in train.lst:

```
mac.0001.dat
mae.0002.dat
maf.0003.dat
mah.0004.dat
mam.0001.dat
mam.0002.dat
mam.0003.dat
mam.0004.dat
```

Recognition can be performed with the following script. We run HVite to generate a set of recognised label files (with a .rec file extension) then load them into the SFS files for evaluation:

```
# dotest.sh
Hvite -T 1 -C config.txt -w phone.net -o S -S test.lst phone.dic phone.lst
for f in `cat test.lst`
do
   g=`echo $f | sed s/.dat//`
   anload -h $g.rec $g.sfs
done
```

In this script, the HVite option "-w phone.net" instructs it to perform word recognition, while the option "-o S" requests it not to put numeric scores in the recognised label files. The loop at the bottom takes the name of each test data file in turn, strips off the .dat from its name and loads the .rec file into the .sfs file with the SFS program anload.

Performance evaluation

We are now in a position to evaluate how well our system is able to divide a speech signal up into SIL-VOI-UNV. The inputs to the evaluation process are the SFS files for the test data. These now have the original acoustic annotations, the mapped annotations and the recognised annotations, see Figure E.2.3.

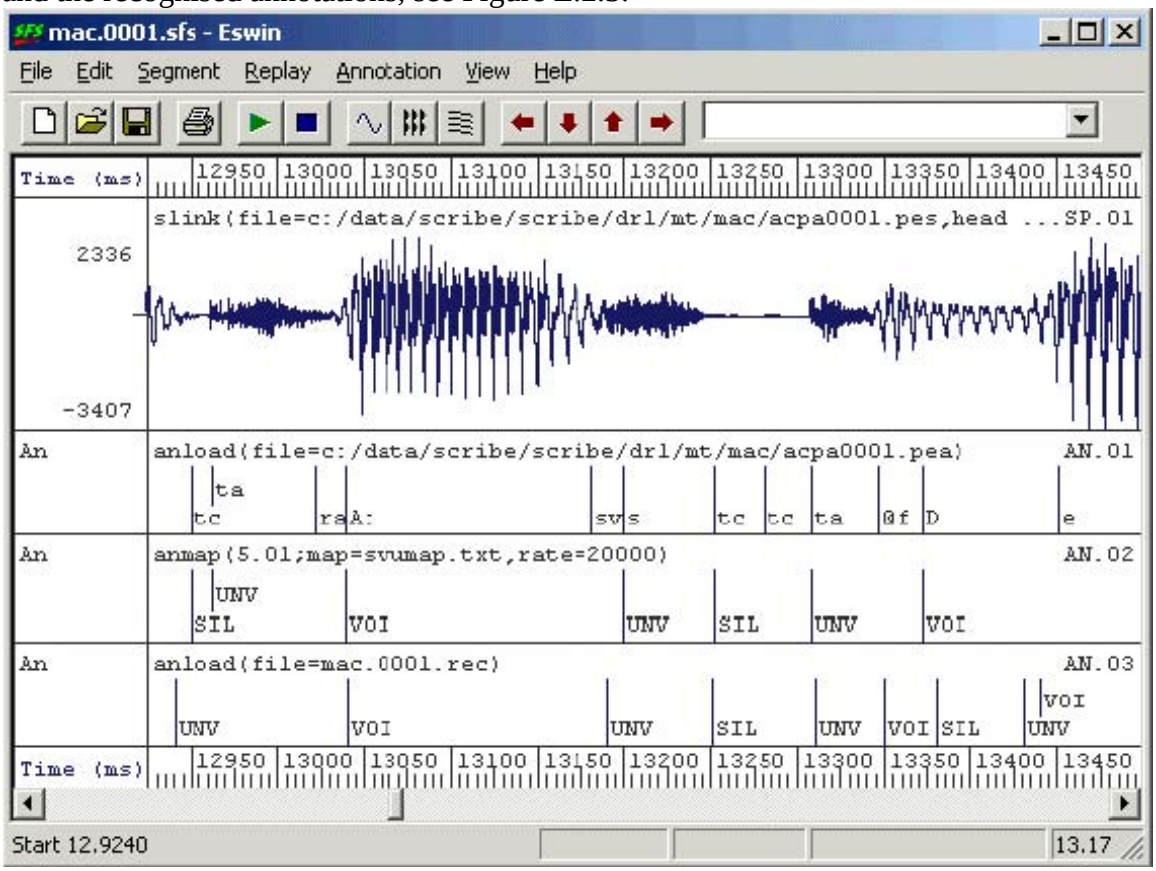


Figure E.2.3 - Recognition results

The figure shows that the recognised annotations are not bad, but there are some mistakes. We now need to consider what numerical measure we might use to describe the performance of the recogniser. In this case a suitable measure is the *frame labeling rate* (the percentage of frames of the signal which have been classified correctly). Since the recognised labels have been based on a frame rate of 100 frames per second, it also makes sense to evaluate the recognition performance at this rate.

The SFS program ancomp compares two annotation sets in various ways. It is supplied with a reference set and a test set and it can compare the timing of the labels, the content of the labels or the sequence of the labels. Here we want to look at the content of

the labels every 10ms and build a confusion matrix that identifies which input (=reference) labels have been mapped to which output (=test) labels with what frequency. The command and some output for a single test file follows:

```
$ ancomp -r an.02 -t an.03 -f mac.0001.sfs
      SIL UNV VOI
SIL:  1114   31    6
UNV:    57  639  100
VOI:   140  138 1514
```

The "-f" switch to ancomp selects the frame labeling mode of comparison, while the switches "-r" and "-t" specify the reference and the test annotation items respectively. This way of running ancomp delivers performance on a single file only however, and we would like to get the performance over all test files. We can do this by asking ancomp to dump its raw label comparisons to its output and then combine the outputs from the program across all test files. This output can then be sent to the SFS program conmat which is a general purpose program for producing confusion matrices. The script is as follows:

```
# doperf.sh
(for f in `cat test.lst`
do
   g=`echo $f|sed s/.dat//`
   ancomp -r an.02 -t an.03 -f -m - $g.sfs
done) | conmat –esl
```

If we run this on all the test files, we get this overall performance assessment:

```
$ sh doperf.sh
Processing date     : Mon Jun 28 12:44:05 2004
Confusion data from : stdin

   Confusion Matrix

      | SIL   UNV     VOI
-----+------------------------
SIL  | 8130    855    151   9136 total  88%
UNV|  888    5010   1631   7529 total  66%
VOI |   736    863 12298  13897 total  88%

Number of matches = 30562
Recognition rate  =  83.2%
```

A way of diagnosing where the recognition is failing is to look at the mapping from the original acoustic annotations to the recognised SIL-VOI-UNV labels. We can do this for a single file as follows:

```
$ ancomp -r an.01 -t an.03 -f mac.0001.sfs
      SIL UNV  VOI
  #:   12    1 0
 ##:  903    2 0
  +:   10    7 0
```

```
 /:    0    0 0
 3::   0    0 28
3:?:   0    0 4
 =n:   15   6 25
=nf:   0    0 1
 @:    1    6 144
 @?:   2    7 15
@U@:   0    1 28
@UU:   0    0 18
 @f:   0    8 11
 @~:   0    1 65
 A::   0    1 54
A:f:   0    0 1
A:~:   0    0 3
 D:    11   5 14
...
```

We could study this output to see whether we had made mistakes in our mapping from acoustic labels to classes.

Our little project could be extended in a number of ways:

1. **Use a different acoustic feature set**. A popular spectral envelope feature set is mel-scaled cepstral coefficients (MFCCs). These can be calculated with the SFS program mfcc.

2. **Use a different HMM configuration**. It is possible that a 3-state rather than 1-state HMM would perform better on this task, although care would have to be taken that it was still capable of recognising segments which were shorter than 3 spectral frames.

3. **Use a different density function**. Since we are mapping a wide range of spectral vectors to a few classes, it is likely that the spectral density function for each class will not be normally distributed. The use of Gaussian mixtures within the HMMs may help.

4. **Use of full covariance**. The 19 channels of the filterbank have a significant degree of covariation, which may affect the accuracy of the probability estimates from the HMM. It may be better to use a full covariance matrix in the HMM rather than just a diagonal covariance.

5. **Use of symbol sequence constraints**. Although unlikely to make much of an impact in this application, in many tasks there are constraints on the likely sequences of recognised symbols. HTK allows us to put estimated sequence probabilities in the recognition network.

3. Phone recognition

In this section we will build a simple phone recogniser. Again the aim is not to get ultimate performance, but to demonstrate the steps and the tools involved.

Source Data

Training a phone recogniser requires a lot of data. For a speaker-dependent system you need several hundred sentences, while for a speaker-independent system you need several thousand. For this tutorial we will use the WSJCAM0 database. This is a database of British English recordings modeled after the Wall Street Journal database. The WSJCAM0 database is available from the Linguistic Data Consortium (www.ldc.upenn.edu). This database is large and comes with a phone labeling that makes it very easy to

train a phone recogniser.

For the purposes of this tutorial we will just use a part of the speaker-independent training set. We will use speakers C02 to C0Z for training, and speakers C10 to C19 for testing. Within each speaker we only use the WSJ sentences, which are coded with a letter 'C' in the fourth character of the filename. The first thing to do is to obtain a list of file 'basenames' - just the directory name and basename of each file we will use for training and for testing. The following script will do the job:

```
# dogetnames.sh – get basenames of files for training and testing
rm -f basetrain.lst
for d in c:/data/wsjcam0/si_tr/C0*
do
  echo processing $d
  for f in $d/???C*.PHN
  do
    g=`echo $f | sed s/.PHN//`
    if test -e $g.WV1
    then
      h=`echo $g | sed s%c:/data/wsjcam0/si_tr/%%`
       echo $h >>basetrain.lst
    fi
  done
done
rm -f basetest.lst
for d in c:/data/wsjcam0/si_tr/C1[0-9]
do
  echo processing $d
```

This script decompresses the audio signals in the ".WV1" files and loads in the phonetic annotations. The SFS files are created in "train" and "test" subdirectories of the tutorial folder:

```
$ mkdir tutorial2
$ cd tutorial2
$ sh dogetnames.sh
$ sh domakesfs.sh
```

Figure E.3.1 shows an example source data file with an audio signal and phonetic annotations.

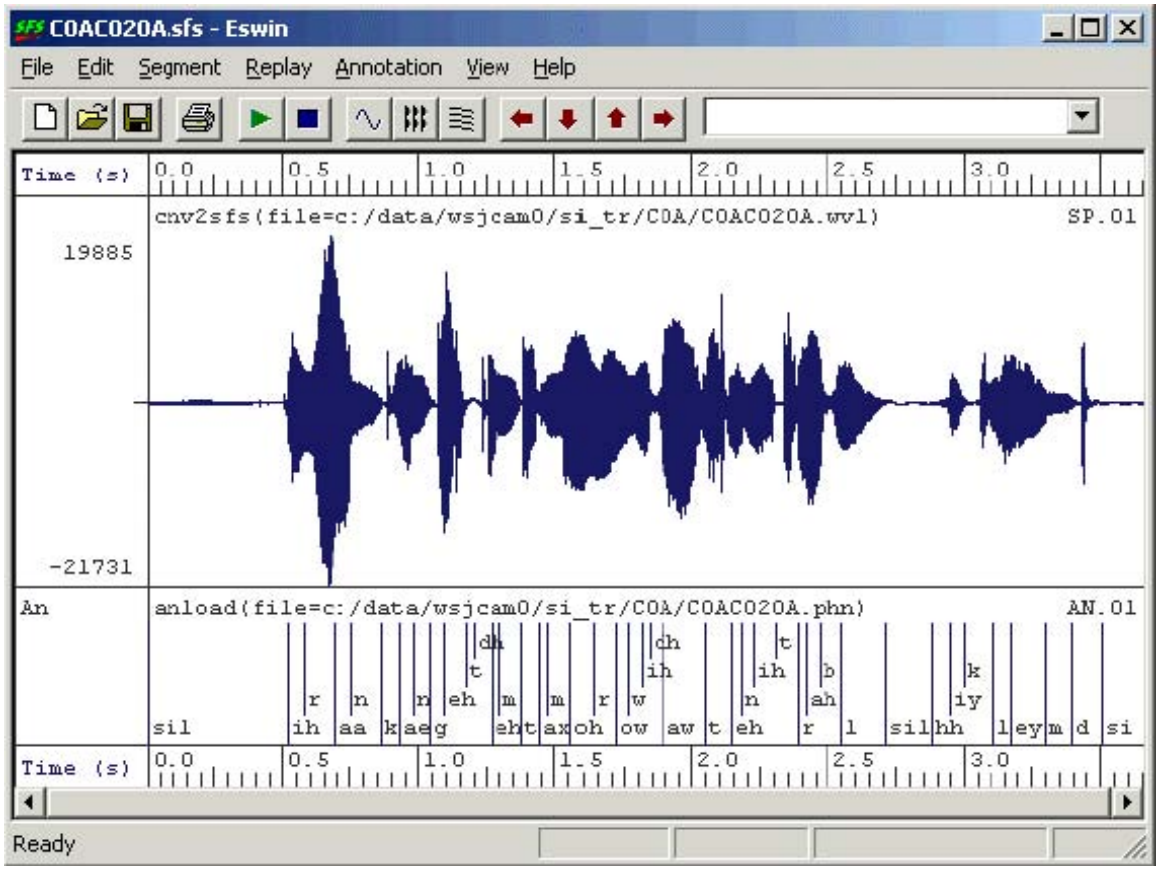


Figure E.3.1 - Source data from WSJCAM0

Making HTK data files

We can now choose an acoustic feature set for the data and generate suitable HTK format data files and label files.

A very common type of acoustic feature set for phone recognition is based on mel-scaled cepstral coefficients (MFCCs). To keep things simple, we will use 12 MFCC parameters plus one parameter that is the log energy for each 10ms frame of signal. We will use the SFS program mfcc for this, but in fact HTK has its own tool for calculating MFCCs. Also for speed we will not use delta or delta-delta coefficients although these have been shown to improve recognition performance in some circumstances. The following shell script performs the MFCC calculation, saves the data into an HTK format file and records the HTK file names in 'train.lst' and 'test.lst' for us to use when we are training and testing HMMs.

```
# domakedat.sh
#
rm -f train.lst
for f in `cat basetrain.lst`
do
  mfcc -n12 -e -l100 -h6000 train/$f.sfs
  colist -H train/$f.sfs
  echo train/$f.dat >>train.lst
done
rm -f test.lst
for f in `cat basetest.lst`
do
  mfcc -n12 -e -l100 -h6000 test/$f.sfs
  colist -H test/$f.sfs
  echo test/$f.dat >>test.lst
done
```

To save producing one HTK label file per data file, we will create an HTK "Master Label File", which will hold all the phone labels for all files. Master label files can be built using the HTK HLEd program, but here we will use an SML script. This is easier to run and allows us to collect a list of the phone names and build a phone dictionary at the same time. The SML script is as follows:

```
/* makemlf.sml – make HTK MLF file from files */

/* table to hold annotation labels */
string    table[1:1000];
var    tcount;

/* MLF file */
file    op;

/* function to check/add label */
function var checklabel(str)
{
   string str;

   if (entry(str,table)) return(0);
   tcount=tcount+1;
   table[tcount]=str;
   return(1);
}

/*initialise */
init {
   openout(op,"phone.mlf");
   print#op "#!MLF!#\n";
}

/* for each input file */
main {
   var       i,num;
   string     basename;
   string    label;

   /* print filename */
   print $filename,"\n"
   i=index("\.",$filename);
   if (i) basename=$filename:1:i-1 else basename=$filename;
   print#op "\"",basename,".lab\"\n";

   /* print annotations */
   num=numberof(".");
   for (i=1;i<=num;i=i+1) {
      label = matchn(".",i);
      print#op label,"\n";
      checklabel(label);
   }
   print#op ".\n"
}

/* output phone list and dictionary */
summary {
   var    i,j;
   string    t;

   /* insertion sort */
   for (i=2;i<=tcount;i=i+1) {
      j=i;
      t=table[j];
      while (compare(t,table[j-1])<0) {
         table[j] = table[j-1];
         j=j-1;
         if (j==1) break;
      }
      table[j]=t;
```

This script is run as follows:

$ sml -f makemlf.sml train test

The "-f" switch to SML means that it ignores non SFS files when it searches directories and sub-directories for files. The output is "phone.mlf" - the HTK master label file, "phone.lst", a list of the phone labels used in the data, "phone+.lst", a list of the phones augmented with enter and exit labels, and "phone.dic", a dictionary in which phone "word" symbols are mapped to phone pronunciations (see section 2 for explanation!).

Training HMMs

To start we will need an HTK global configuration file just as we had in section E.2. Here it is again - put this in "config.txt":

```
# config.txt - HTK basic parameters
SOURCEFORMAT = HTK
TARGETKIND = MFCC_E
NATURALREADORDER = T
```

We are now in a position to train a set of phone HMMs, one model per phone type. We will construct these in a fairly conventional way with 3 states and no skips, with one gaussian mixture of 13 dimensions per state. Put the following in a file "proto-3-13.hmm":

```
<BeginHMM>
<NumStates> 5 <VecSize> 13 <MFCC_E>
 <State> 2
  <Mean> 13
   0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  <Variance> 13
   1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <State> 3
  <Mean> 13
   0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  <Variance> 13
   1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <State> 4
  <Mean> 13
   0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  <Variance> 13
   1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <TransP> 5
  0.0 1.0 0.0 0.0 0.0
  0.0 0.6 0.4 0.0 0.0
  0.0 0.0 0.6 0.4 0.0
  0.0 0.0 0.0 0.7 0.3
  0.0 0.0 0.0 0.0 1.0
<EndHMM>
```

We now initialise this model to be "flat" - that is with each state mean set to the global data mean and each state variance set to the global data variance. The HTK tool HCompV will do this, as follows:

$ HCompV -T 1 -C config.txt -m -S train.lst -o proto.hmm proto-3-13.hmm

This creates a file "proto.hmm" with the <MEAN> and <VARIANCE> sections initialised to appropriate values. We now need to duplicate this prototype into a model for each phone symbol type. We can do this with a shell script as follows:

```
# domakehmm.sh
HCompV -T 1 -C config.txt -m -S train.lst -o proto.hmm proto-3-13.hmm
Head -3 proto.hmm > hmmdefs
for s in `cat phone.lst`
do
  echo "~h \"$s\"" >>hmmdefs
  gawk '/BEGINHMM/,/ENDHMM/ { print $0 }' proto.hmm >>hmmdefs
done
```

Run this script to creates a file "hmmdefs" which has a model entry for each phone all initialised to the same mean values.

We can now train the phone models using the HTK embedded re-estimation tool HERest. The basic command is as follows:

```
$ HERest -C config.txt -I phone.mlf -S train.lst -H hmmdefs phone.lst
```

This command re-estimates the HMM parameters in the file hmmdefs, returning the updated model to the same file. This command needs to be run several times, as the re-estimation process is iterative. To decide how many cycles of re-estimation to perform it is usual to monitor the performance of the recogniser as it trains and to stop training when performance peaks. For this to work we need to test the recogniser on material that hasn't been used for training, and for honesty won't be used for the final performance evaluation either.

To estimate the performance of the recogniser, we use it to recognise our reserved test data and compare the recognised transcriptions to the ones distributed with the database. To perform recognition we need the phone.lst file, which lists the names of the models, the phone.dic file, which maps the phone names onto themselves, and a phone.net file, which contains the recognition grammar. In this application it makes sense to use a bigram grammar in which we record the probabilities that one phone can follow another. We can build this with the commands:

```
$ HLStats -T 1 -C config.txt -b phone.big -o phone.lst phone.mlf
$ HBuild -T 1 -C config.txt -n phone.big phone+.lst phone.net
```

The HLStats command collects bigram statistics from our master label file and stores them in phone.big. The HBuild command converts these into a network grammar suitable for recognition. The phone+.lst file is the list of phone models augmented with the symbols "!ENTER" and "!EXIT".

The basic recognition command, now, is just:

```
$ HVite -T 1 -C config.txt -H hmmdefs -S test.lst -i recout.mlf \
-w phone.net phone.dic phone.lst
```

This command runs the recogniser and stores its recognition output in the recout.mlf master label file. To compare the recognised labels to the distributed labels, we can use the HResults program:

```
$ HResults -I phone.mlf phone.lst recout.mlf
====================        HTK        Results        Analysis
======================
  Date: Thu Jul  1 09:10:58 2004
  Ref : phone.mlf
```

```
  Rec : recout.mlf
------------------------ Overall Results -------------------------
SENT: %Correct=0.00 [H=0, S=903, N=903]
WORD: %Corr=48.03, Acc=41.52 [H=31884, D=10898, S=23605, I=4319,
N=66387]
============================================================
=========
```

We can put this all together in a script which runs through 10 cycles of re-estimation and collects the performance after each cycle in a log file:

```
# dotrainrec.sh
rm -f log
for n in 1 2 3 4 5 6 7 8 9 10
do
  HERest -T 1 -C config.txt -I phone.mlf -S train.lst -H hmmdefs phone.lst
  Hvite -T 1 -C config.txt -H hmmdefs -S test.lst -i recout.mlf \
  -w phone.net phone.dic phone.lst
  echo "Cycle $n:" >>log
  Hresults -I phone.mlf phone.lst recout.mlf >>log
Done
```

Figure E.3.2 shows how the Accuracy figure changes with training cycle on our data:

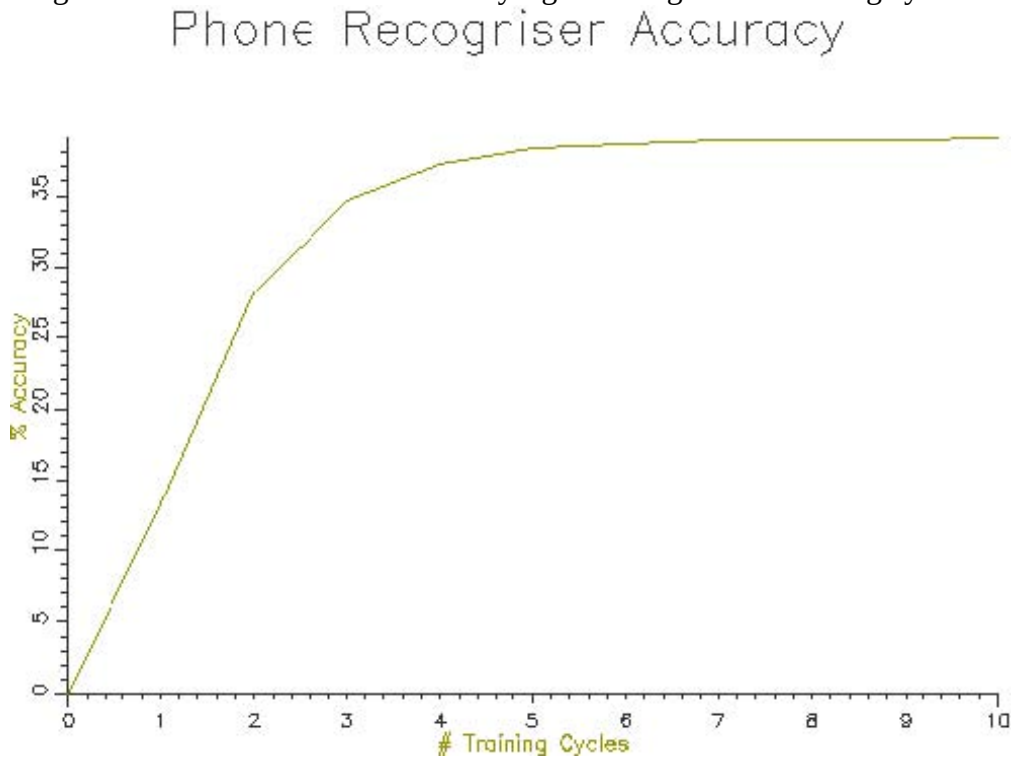


Figure E.3.2 - Phone recognition accuracy with number of training cycles

Our recogniser seems to reach a maximum performance of about 40% phone accuracy. This is not very good; the best phone recognisers on this data have an accuracy of 70%.

Parameter Analysis

One possible contributory factor to the relatively poor performance of our phone recogniser might be the use of single gaussian (normal) distributions for the modelling of the cepstral coefficient variation within a state. We can easily write an SML script to investigate whether the actual distributions are not modeled well by a gaussian distribution. The following script asks for a segment label and then scans the source SFS files to build histograms of the first 12 cepstral coefficients as they vary within instances of that segment. The distributions are plotted with an overlay of a normal distribution.

```
/* codist.sml - plot distributions of MFCC data values */

/* raw data */
var    rdata[12,100000];
var    rcount;

/* distributions */
stat    rst[12];

/* segment label to analyse */
string    label;

/* graphics output */
file gop;

/* normal distribution */
function var normal(st,x)
stat st;
{
   var x;
   x = x - st.mean;
   return(exp(-0.5*x*x/st.variance)/sqrt(2*3.14159*st.variance));
}

/* plot histogram overlaid with normal distribution */
function var plotdist(gno,st,tab,tcnt)
stat st;
var tab[];
{
   var gno;
   var tcnt;
   var i,j,nbins,bsize;
   var hist[0:100];
   var xdata[1:2];
   var ydata[0:10000];

   /* find maximum and minimum in table */
   xdata[1]=tab[gno,1];
   xdata[2]=tab[gno,1];
   for (i=2;i<=tcnt;i=i+1) {
      if (tab[gno,i] < xdata[1]) xdata[1]=tab[gno,i];
      if (tab[gno,i] > xdata[2]) xdata[2]=tab[gno,i];
   }

   /* set up x-axes */
   plotxdata(xdata,1)

   /* estimate bin size */
   nbins = sqrt(tcnt);
   if (nbins > 100) nbins=100;
   bsize = (xdata[2]-xdata[1])/nbins;

   /* calculate histogram */
   for (i=1;i<=tcnt;i=i+1) {
      j=trunc((tab[gno,i]-xdata[1])/bsize);
      hist[j]=hist[j]+1/tcnt;
   }

   /* plot histogram */
   plotparam("title=C"++istr(gno));
   plotparam("type=hist");
   plot(gop,gno,hist,nbins);
```

This script can be run as

$ sml -f codist.sml train

For segment : r

Two outputs of the script are shown in Figures E.3.3 and E.3.4.

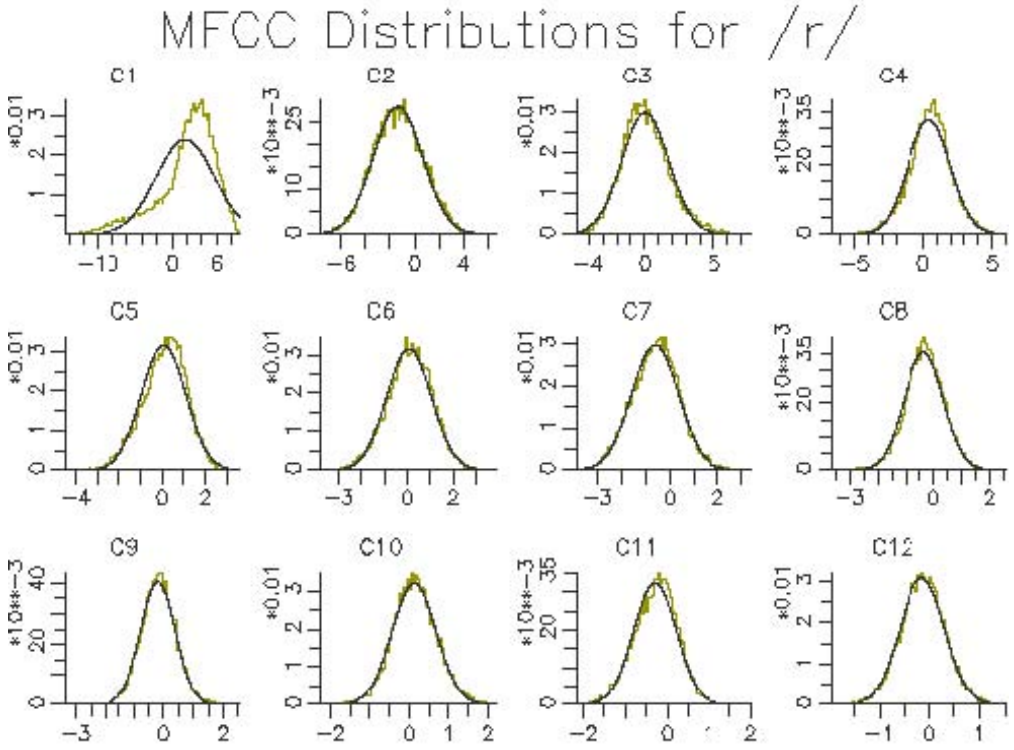


Figure E.3.3 - Modelled cepstral distributions for /r/

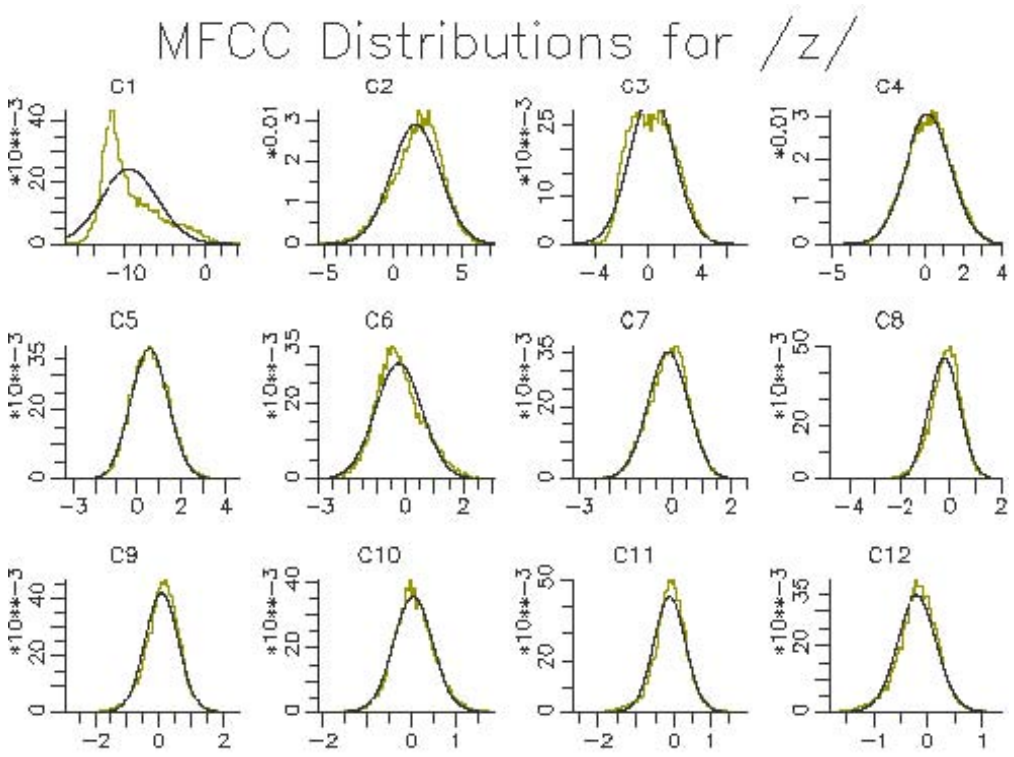


Figure E.3.4 - Modelled cepstral distributions for /z/

The figures show that a single gaussian distribution is quite good for most of the cepstral coefficients, but not for the first cepstral coefficient. This implies that we may get a small performance improvement by changing to more than one gaussian per state. To increase the number of gaussian mixtures on all phone models (for all cepstral coefficients) we can use the HTK program HHed. This program is a general purpose HMM editor and takes as input a control file of commands. In this case we just want to increase the number of mixtures on all states. Put this in a file called mix2.hed:

```
MU 2 {*.state[2-4].mix}
```

Then the HMMs can be edited with the command

```
$ HHed -H hmmdefs mix2.hed phone.lst
```

A few more cycles of training can now be applied to see the effect.

Viewing recognition results in SFS

The output of the phone recogniser above is an HTK master label file recout.mlf. If we want to view these results within SFS we need to load these as annotations. We can do this directly with the SFS program anload. First we take a copy of a test file, then load in the annotations corresponding to that file from the master label file:

```
$ cp test/c10/c10c020v.sfs .
$ anload -H recout.mlf test/c10/c10c020v.rec c10c020v.sfs
$ eswin -isp -aan c10c020v.sfs
```

Notice that the anload program takes the name of the MLF file and the name of the section in the file to load. The SFS program eswin displays the speech and annotations in the file, as shown in Figure E.3.5.

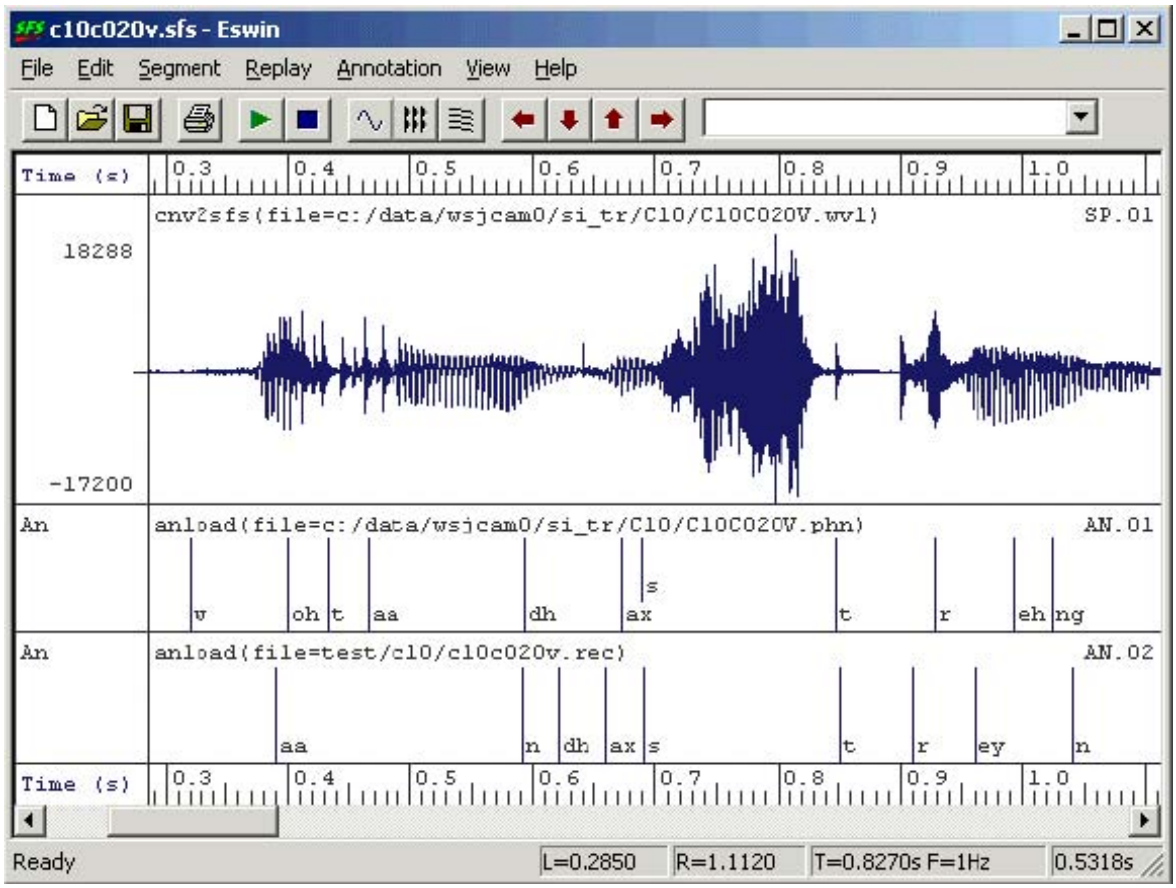


Figure.E.3.5 - Viewing recognition results

Although the HTK program HResults can analyse the performance of our recogniser on the test data and the confusions it makes, we can also perform a similar analysis in SFS for single or multiple files. The process is to load the recognised phone labels into SFS and then use the ancomp program in its "labeling" mode. We can do this on a single file with:

```
$ cp test/c10/c10c020v.sfs .
$ anload -H recout.mlf test/c10/c10c020v.rec c10c020v.sfs
$ ancomp -l c10c020v.sfs
Subst=21 Delete=6 Insert=5 Total=64 Accuracy=59.4%
```

To compare multiple files, we get ancomp to list its raw phone alignments to a file and then input the collected output to the confusion matrix program. Here is a script to do this:

```
# doancomp.sh
#
# collect mappings
(for f in `cat basetest.lst`
do
   cp test/$f.sfs temp.sfs
   anload -H recout.mlf test/$f.rec temp.sfs
   ancomp -l -m - temp.sfs
done) >ancomp.lst
rm temp.sfs
#
# build confusion matrix
conmat -esl ancomp.lst >conmat.lst
```

The file conmat.lst contains a large phoneme confusion matrix as well as an overall performance score. Since this is rather unwieldy, it is also interesting just to find the most common confusions. We can generate a list of confusions from ancomp.lst using some unix trickery:

```
$ gawk '{ if ( $1 != $2 ) print $0 }' ancomp.lst | sort | uniq -c | \
  sort -rn | head -20
   882 [] sil
   820 ax []
   724 t []
   616 ih []
   468 ih ax
   415 n m
   409 ih iy
   373 ih uw
   373 d []
   367 n []
   325 s z
   313 l []
   289 r []
   283 t s
   278 n ng
   268 dh []
   266 [] d
   264 l ao
   262 ax ih
   262 ax ah
```

The most common confusions are probably what we would expect: insertions of silence, deletions of /@/, /t/ and /I/, substitutions of /I/ with /@/, or /m/ for /n/, and so on. How this command works is left as an exercise for the reader!

Enhancements

We might improve the performance of the phone recogniser in a number of ways:

- **Add delta coefficients**: Adding the rate of change of each acoustic parameter to the feature set (deltas) has been shown to improve phone recognition performance as

has the addition of accelerations (delta-deltas). You can do this easily by changing the type of the HMM to MFCC_E_D_A.

- **Build phone in context models**: Building phone models which are different according to the context in which they occur can help a great deal. A typical approach to this is to build "triphone" models - where we build a model for each phone for every pair of possible left and right phones in the label files. Since this requires more data than we have typically, it is also necessary to smooth the probability estimates arising from training by "state-tying" - using the data over many triphone contexts to estimate observations for a state of one triphone model. This procedure is usually performed in a data-driven way using clustering methods. The HTK documentation gives details.

- **Add a phone language model**: Although our recogniser uses bigram probabilities to constrain recognition, there are also useful constraints on sequences longer than two phones that would help recognition. HTK does not have a simple way of doing this, but one might expect that 3-gram, 4-gram or even 5-gram phone grammars would help considerably.

4. Word recognition

Once we have built a phone recogniser, it is a simple matter to extend it to recognise words. Of course it is also possible to build a word recogniser in which each word is modelled separately with an HMM.

Dictionary

To build a word recogniser from a phone recogniser, we first need a dictionary that maps words to phone sequences. Here is an example for a simple application. Put this in digits.dic:

ZERO z ia r ow
ZERO ow
ONE w ah n
TWO t uw
THREE th r iy
FOUR f ao
FOUR f ao r
FIVE f ay v
SIX s ih k s
SEVEN s eh v n
EIGHT ey t
NINE n ay n
WHAT-IS w oh t ih z
PLUS p l ah s
MINUS m ay n ax s
TIMES t ay m z
DIVIDED-BY d ih v ay d ih d b ay
SIL [] sil

Grammar

Next we need a grammar file which constrains the allowed word order. The more constraints we can put here, the more accurate our recogniser. Here is a simple grammar file that allows us to recognise phrases such as "what is two plus five". Put this in digits.grm:

```
$digit = ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT | NINE | ZERO;
$operation = PLUS | MINUS | TIMES | DIVIDED-BY;
( SIL WHAT-IS <$digit> $operation <$digit> SIL )
```

To convert this file to a form that the recogniser can use, we need to run the HTK tool HParse, as follows:

```
$ HParse digits.grm digits.net
```

Word recogniser

The basic command for recognising HTK data file inp.dat using our digits task recogniser is then just:

```
$ HVite -T 1 -C config.txt -H hmmdefs -w digits.net digits.dic \
  phone.lst inp.dat
```

We can put together a simple script that uses SFS to acquire an audio signal, then performs an MFCC analysis, exports the coefficients to HTK and runs the recogniser:

```
# doreclive.sh
rm -f inp.sfs
hed -n inp.sfs >/dev/null
echo "To STOP this script, type CTRL/C"
#
remove -e inp.sfs >NUL
echo "***** Say Word *****"
while record -q -e -f 16000 inp.sfs
do
  replay inp.sfs
  mfcc -n12 -e -l100 -h6000 inp.sfs
  colist -H inp.sfs
  HVite -T 1 -C config.txt -H hmmdefs -w digits.net digits.dic \
    phone.lst inp.dat
  remove -e inp.sfs >NUL
  echo "***** Say Word *****"
done
```

You could also use the HTK live audio input facility for this demonstration. But then you should also use the HTK MFCC analysis in training as well, as the HTK MFCC analysis gives slightly different scaled coefficient values from the SFS program mfcc.

Enhancement

We might enhance our word recogniser in a number of ways:

- Use a better set of phone models, for example with triphones.
- For a small vocabulary and lots of training data, it is better to build word-level

models.

- For a large vocabulary it is better to use a statistical language model, such as a trigram model rather than trying to build a grammar.
- Model non-speech regions and silent gaps more intelligently, by having models for noises and inter-word silent gaps for example.

5. Phone alignment

Another application for our phone recogniser is phone alignment. In phone alignment we have an unlabeled audio signal and a transcription and the task is to align the transcription to the signal. This procedure is already implemented as part of SFS using the program analign, but we will show the basic operation of analign here.

Assume we have an audio recording of a single sentence, here it is the sentence "six plus three equals nine" stored in an sfs file called six.sfs. We first perform MFCC analysis on the audio signal:

```
$ mfcc -n12 -e -l100 -h6000 six.sfs
```

We next add an annotation containing the raw transcription in ARPABET format:

```
$ anload -t phone -T "sil s ih k s p l ah s th r iy iy k w ax l z n ay n sil" six.sfs
```

Then export both coefficients and annotations to HTK format:

```
$ colist -H six.sfs
$ anlist -h -O six.sfs
```

We can now run the HTK HVite program in alignment mode with:

```
$ HVite -C config.txt -a -o SM -H hmmdefs phone.dic phone.lst six.dat
```

And load the aligned annotations back into SFS:

```
$ anload -h six.rec six.sfs
```

The result is shown in figure E.5.1.

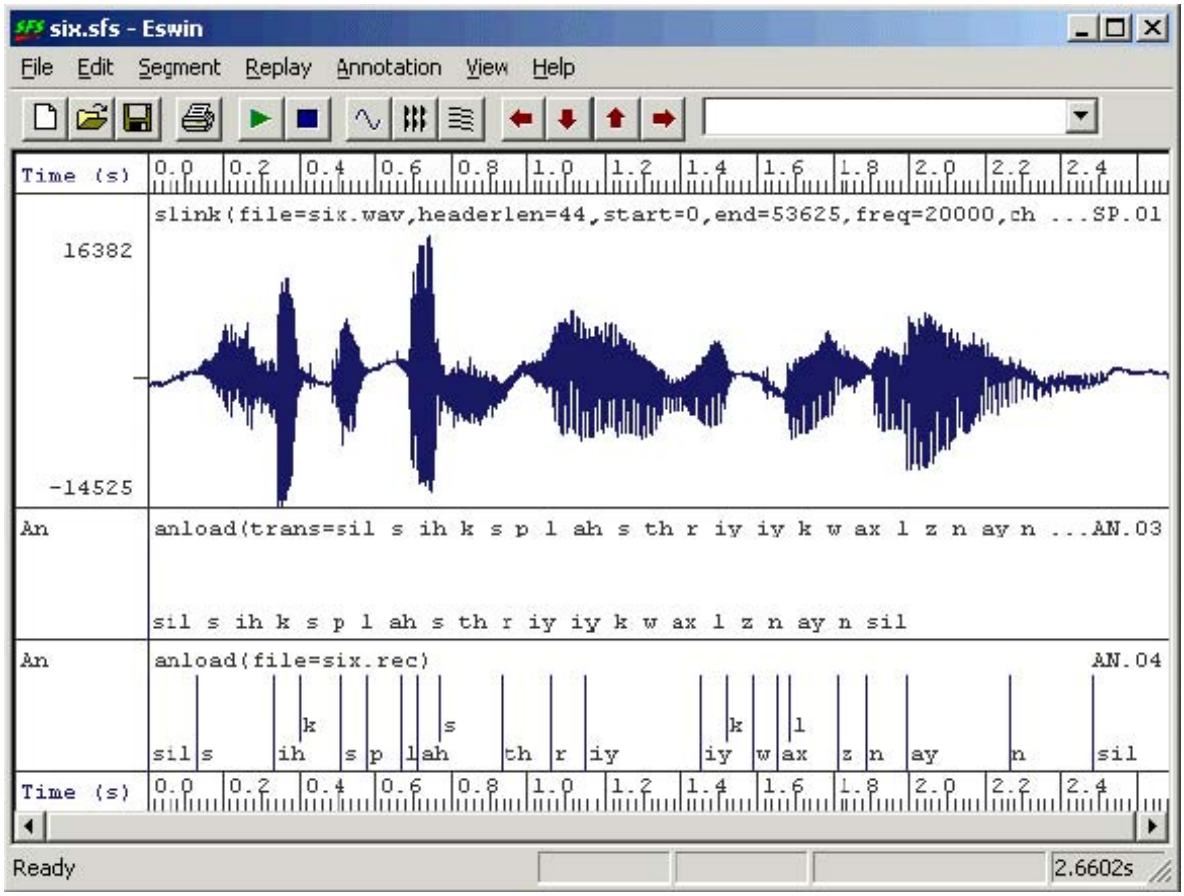


Figure E.5.1 - Aligned phone labels

The SFS program analign makes this process easier by handling all the export and import of data to HTK and also handles the chopping of large files into sentence sized pieces and the translation of symbol sets between SAMPA, ARPABET and JSRU.

6. Pronunciation variation analysis

A variation on phone alignment is to provide a shallow network of pronunciation alternatives to HVite and let it choose which pronunciation was actually used by a speaker. This kind of variation analysis could be useful in the study of accent variation or when attempting to recognized dysfluenceies from speakers with different accents.

In this example, we will take a sentence spoken by two speakers with different regional accents of the British Isles. The sentence is "after tea father fed the cat", and we are interested whether the vowel /{/ or /A:/ was used in the words "after" and "father". For reference, we always expect "cat" to be produced with /{/, so we will check on the analysis by also looking to see if the program rejects the pronunciation of "cat" as /kA:t/.

We first create a dictionary file for the sentence. Put this in accents.dic:

```
AFTER      aa f t ax
AFTER      ae f t ax
TEA        t iy
FATHER     f aa dh ax
FATHER     f ae dh ax
FED        f eh d
THE        dh ax
CAT        k aa t
CAT        k ae t
SIL    []   sil
```

Notice that the dictionary lists the alternative pronunciations in which we are interested. Next we create a grammar just for this sentence. Put this in accents.grm

```
( SIL AFTER TEA FATHER FED THE CAT SIL )
```

Next we convert the grammar file to a recognition network with HParse:

```
$ HParse accents.grm accents.net
```

Then we prepare HTK data files for the two audio signals:

```
$ mfcc -n12 -e -l100 -h6000 brm.sfs
$ mfcc -n12 -e -l100 -h6000 sse.sfs
$ colist -H brm.sfs
$ colist -H sse.sfs
```

Then we recognise the utterances using the grammar, but outputting the selected phone transcription.

```
$ HVite -C config.txt -H hmmdefs -w accents.net -m -o ST accents.dic \
  phone.lst brm.dat sse.dat
```

The "-m" switch to HVite causes the phone model names to be output to the recognised label files, while the "-o ST" switch cause the scores and the times to be suppressed. The output of the recogniser are in brm.rec and sse.rec as follows:

brm.rec	**sse.rec**
sil SIL	sil SIL
ae AFTER	aa AFTER
f	f
t	t
ax	ax
t TEA	t TEA
iy	iy
f FATHER	f FATHER
aa	aa
dh	dh
ax	ax
f FED	f FED
eh	eh
d	d
dh THE	dh THE
ax	ax
k CAT	k CAT
ae	ae
t	t
sil SIL	sil SIL

From this it is easy to see that the speaker from Birmingham used /{/ where the speaker from South-East England used /A:/ in "after".

7. Dysfluency recognition

Another application of phone recognition is the detection of dysfluencies. Although our phone recogniser is not particularly accurate we can still look for patterns in the recognised phone sequence which may be indicators of dysfluency. We will recognise a passage with the phone recogniser, load the phone labels into the SFS file then use an SML script to look for dysfluent patterns.

Here are the commands for performing MFCC analysis on the passage, saving the coefficients to HTK format, building a simple phone loop recogniser without bigram constraints, then running the recogniser and loading the phone annotations back into the file:

```
$ mfcc -n12 -e -l 100 -h 6000 dysfluent.sfs
$ colist -H dysfluent.sfs
$ HBuild phone.lst phone.net
$ HVite -C config.txt -H hmmdefs -w phone.net -o S phone.dic phone.lst dysfluent.dat
$ anload -h dysfluent.rec dysfluent.sfs
```

The following SML script takes the recognised phone sequence and looks for (i) single (non-silent) phones lasting for longer than 0.25s, (ii) repeated (non-silent) phone labels which together last for longer than 0.25s, (iii) patterns of five phone labels matching A-B-A-B-A where A is not silence. Output is another annotation item with the dysfluent regions marked with "(D)".

```
/* dysfind.sml – find dysfluencies from phone recogniser output */

/* input and output annotation sets */
item    ian;
item    oan;

/* table to hold dysfluent events */
var    times[1000,2];
var    tcount;

/* add times of dysfluencies to table */
function var addtime(posn,size)
{
   var posn,size;
   var i;

   /* put event in sorted position */
   i=tcount+1;
   if (i > 1) {
      while (posn < times[i-1,1]) {
         times[i,1] = times[i-1,1];
         times[i,2] = times[i-1,2];
         i = i - 1;
         if (i==1) break;
      }
   }
   times[i,1]=posn;
   times[i,2]=size;
   tcount=tcount+1;
}

/* process each input file */
main {
   var    i,numf,fdur,dmin;
   var ocnt,size;
   string lab1,lab2,lab3;

   /* get input & output */
   sfsgetitem(ian,$filename,str(selectitem(AN),4,2));
   numf=sfsgetparam(ian,"numframes");
   fdur=sfsgetparam(ian,"frameduration");
   sfsnewitem(oan,AN,fdur,sfsgetparam(ian,"offset"),1,numf);

   /* put minimum dysfluency length = 0.25s */
   dmin = 0.25/fdur;

   /* look for long non-silent annotations */
   tcount=0;
   for (i=1;i<=numf;i=i+1) {
      size = sfsgetfield(ian,i,1);
      if (size > dmin) {
         lab1 = sfsgetstring(ian,i);
         if (compare(lab1,"sil")!=0) {
            addtime(sfsgetfield(ian,i,0),size);
         }
      }
   }

   /* look for patterns like AA */
   for (i=2;i<=numf;i=i+1) {
      lab1 = sfsgetstring(ian,i-1);
      lab2 = sfsgetstring(ian,i);
      if (compare(lab1,lab2)==0) {
```

This script would be run with:

$ sml -ian^anload dysfind.sml dysfluent.sfs

Figure E.7.1 shows one particular pattern of "ih sil ih sil ih" which is a real dysfluency detected by the script.

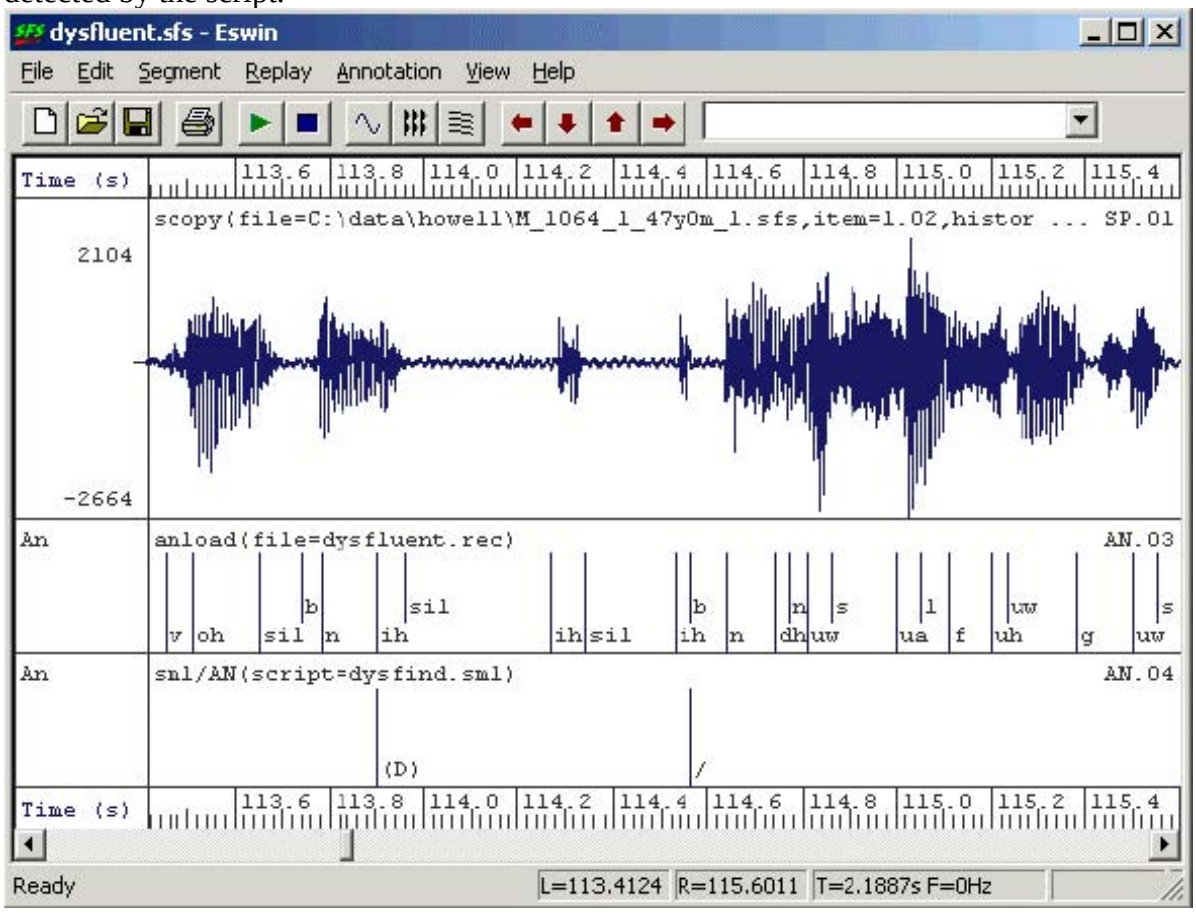


Figure E.7.1 - Dysfluency recognition

Bibliography

- BEEP British English pronunciation dictionary at ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/dictionaries/beep.tar.gz.
- Hidden Markov modelling toolkit at http://htk.eng.cam.ac.uk/.
- International Phonetic Alphabet at http://www.arts.gla.ac.uk/IPA/ipachart.html.
- SAMPA Phonetic Alphabet at http://www.phon.ucl.ac.uk/home/sampa/.
- Speech Filing System at http://www.phon.ucl.ac.uk/resource/sfs/.
- SCRIBE corpus at www.phon.ucl.ac.uk/resource/scribe.

Feedback

Please report errors in appendices C, D and E to SFS@phon.ucl.ac.uk. Questions about the use of SFS can be posted to the SFS speech-tools mailing list.