TR-I-0348

S/Splus for Speech Processing Research at ATR/ITL

Harald Singer

1993.3

ABSTRACT

This report is based upon our experience with the language S (Splus is a superset of S provided by StatSci) spanning about a year. It is intended to give some hints and tips about how to use S, whom to ask if something goes wrong, etc. By way of (actually used) examples we want to introduce you to the power of S, which is much more than a desktop calculator, a plotting program or a statistical package.

Contents

Chapter 1

Introduction

This report is based upon our experience with the language S [1] spanning about a year. It is intended to give some hints and tips about how to use S, whom to ask if something goes wrong, etc. By way of (actually used) examples we want to introduce you to the power of S, which is much more than a desktop calculator or a plotting program!

In the rest of this chapter we will show you where to get more information and how to work efficiently with S. This report supposes a thorough knowledge of UNIX programming (csh, awk, sed, grep) and shows with examples from speech research how one can use S. Installation of S on your workstation is described in Appendix A.

All examples and the source file for this text can be found online on host `atr-fs` in directory `/usr/common/src/singer/splus/S_ATR`. Any filenames in the text refer to this directory. To distinguish S commands and csh commands, we use > as S prompt and $ as csh prompt.

Please let us know, if you have any further suggestions, bug fixes and so on. We would also be grateful for contributions in the form of useful S routines. Have fun!

1.1 First Steps

1.1.1 Where to Get Information

Apart from asking your *Local Guide*, the following books and manuals may answer many of your questions.

- "The New S Language" was written by the developeres of S at AT&T. It is the bible for S users [Becker 88].
- There is also a predecessor from 1984, which was translated into Japanese [Becker 84]. Beware of changes in the language!
- "Reference Manual S-Plus Ver.2.3" is a printout of the manual, which can also be accessed online, using `help()`. Suppose you want to get help about the function `kmeans`. Type inside S

```
>help(kmeans,pager="cat")
```

- "User's Manual S-PLUS Ver.2.3" from Mathematical Systems Institute. The following chapters were most useful for us: "Appendix 1:Using S-Plus and the X11 Window System"

[1] Splus is a superset of S provided by StatSci

and "Appendix 4: S-News Group and Contributed Software", which describe a mailing list for users of S and an archive server `statlib@math.keio.ac.jp`.

If you want to get on this list your name should be registered on `atr-la` in the appropriate ATR mailing list (`/users/SETTING/MAIL-ALIASES/splus`). Watch out: traffic is high with an average of about 2 messages per day but there is also a lot of extremly useful information.

- W.Venables,"Notes on S: A Programming Environment for Data Analysis and Graphics". A concise introduction for beginners.
- Recently "Statistical Models in S"[Chambers 92] has been published. Unfortunately, S-PLUS Ver.3.0 is used in this book.

1.1.2 Try it out

Work through chapter 2 of the "Blue Book" [Becker 88] or any of the other introductions cited above.

1.2 Working Efficiently

There are many different ways of running S from your terminal. We recommend using S inside GNU Emacs, because you can easily copy, paste, change and save previously typed input.

We usually work under GNU Emacs with a DECterm window split into 2 subwindows (`C-x 2`). Subwindow 1 runs S (started with `M-x S`), subwindow 2 contains a S command file `foo.p`, which we want to edit. Additionally, another X window contains our graphic output. Graphic output can be opened from S with the function call

```
> X11()
```

In GNU Emacs you can switch between subwindows with `C-x o` or, if properly installed, with the mouse. Our usual development cycle is as follows:

1. suppose you define a function `Foo1()`; edit file `foo.p` in subwindow 2
2. `> source('foo.p')` in subwindow 1
3. on error go back to step 1 or first try out the relevant commands
4. run the function `> Foo1()`
5. on error or if you are not satisfied go back to step 1

Even so S programs don't need compilation, you should strive to remember the following points:

- Give meaningful names to your functions and variables.
- Keep your functions modular.
- Add comments after the function declaration.

This will be very useful, if you want to reuse a function or share it with somebody.

As a personal convention, we capitalize all variables and functions we define, to keep them separate from standard S variables and functions.

All input and additional information is automatically saved in file `.Data/.Audit`. To recover only the input in the file `Audit` use the following UNIX command

```
sed '/^#/d' .Data/.Audit > Audit
```

You can then edit file `Audit` to recover your desired input.

The assignment operator <- (greater sign followed by minus sign) can be abbreviated to _ (underscore) , which saves one key stroke.

Chapter 2

Examples

2.1 Interface to UNIX: Duration of Phonemes

Character manipulation is not a strong point of S. This is better left to standard UNIX tools like sed and awk.

Task: Analyze the durational distribution for the 5 Japanese vowels from labelled speech data. `MAU_DSB.5mS.cxt` contains the raw duration data with labels in a 4 field format: start frame, length in frames, label of phoneme and context. Phrases are separated by a blank line and a line with phrase information starting with #. Example:

duration/MAUDSB

```
     ...
3432 22 s d/e/s/u/g
3454 11 g s/u/g/a/-
3465 21 a u/g/a/-/-
3486 20 - g/a/-/-/-

# WAV/DSB/MAU_SB1_03.12K 240.0 9160.0
3506 20 - -/-/-/s/o
3526 17 s -/-/s/o/n
3543 11 o -/s/o/n/o
     ...
```

Method: The following csh shell script extracts all durations for the 5 vowels, separately for vowels at the end of an utterance (FINAL) and not at the end of an utterance (NOTFINAL) and writes them to files `a.FINAL`, `a.NOTFINAL`, `i.FINAL` etc.

duration/extract.csh

```
#!/bin/csh -f
# extract lengths of utterance final and utterance non-final vowels
set PHON = ( a i u e o )
set FILE = MAU_DSB.5mS.cxt
foreach PH ( $PHON )
        echo $PH
        # get final vowels
        nawk '$3=="'$PH'"&&$4~"/-/-$" { printf"%d\n", $2 }' < $FILE > $PH.FINAL
        # get non-final vowels
        nawk '$3=="'$PH'"&&$4!~"/-/-$" { printf"%d\n", $2 }'< $FILE >$PH.NOTFINAL
end
```

The files created by this shell script can then be easily read by S.

duration/duration.p

```
Plot.Phon.Dur.All _ function(){
# example: 1.a.FINAL contains durations of phoneme /a/ of XXX_1.5mS.cxt
#          which are in phrase final position
#          1.a.NOTFINAL contains durations of phoneme /a/ of XXX_1.5mS.cxt
#          which are not in phrase final position
        frame()                 # advance to next figure
        par(mfrow = c(2, 3))    # multiple figures, 2 rows, 3 columns
        for(P in c("a", "e", "i", "o", "u")) {
                # default separator for paste() would be a space!
                XF <- scan(paste(sep = "", P, ".FINAL"))
                XN <- scan(paste(sep = "", P, ".NOTFINAL"))
                Plot.Phon.Dur(XF, XN, P)
        }
}
Plot.Phon.Dur _ function(XF, XN, Title, Minbreak = 0, Maxbreak = 60){
# plots histograms of XF,XN in same plot
# to do this, we first have to get the maximum dimensions with no plotting
        X <- hist(XF, breaks = Minbreak:Maxbreak, plot = F)
        Y <- hist(XN, breaks = Minbreak:Maxbreak, plot = F)
        X <- range(X$counts, Y$counts)
        MF <- Round(mean(XF), 3)
        SF <- Round(sqrt(var(XF)), 3)
        MN <- Round(mean(XN), 3)
        SN <- Round(sqrt(var(XN)), 3)
        cat(Title, length(XF), length(XN), "\n")
        hist(XF, breaks = Minbreak:Maxbreak, ylim = X, lwd = 2, xlab = "")
        par(new = T) # on same figure
        hist(XN, breaks = Minbreak:Maxbreak, ylim = X, lwd = 0.5, axes = F,
                xlab = paste("nonfinal: ", MN, "+", SN, "  final: ", MF, "+", SF))
        title(Title)        # phoneme as title
}
Boxplot.Phon.Dur.All _ function(){
        frame()                 # advance to next figure
        par(mfrow = c(2, 3))    # multiple figures, 2 rows, 3 columns
        for(P in c("a", "e", "i", "o", "u")) {
                XF <- scan(paste(sep = "", P, ".FINAL"))
                XN <- scan(paste(sep = "", P, ".NOTFINAL"))
                Boxplot.Phon.Dur(XF, XN, P)
        }
}
Boxplot.Phon.Dur _ function(XF, XN, Title, Min = 0, Max = 60){
# plots boxplot of XN+XF, XN, XF
        boxplot(c(XF,XN), XN, XF, names=c("all","non-final","final"),
                ylim=c(0,60),notch=T,varwidth=T,main=Title)
}
Round _ function(Value, Precision = 3)    format(signif(Value, Precision))
```

We created Figure 2.1 and Figure 2.2 with the following commands

```
> source("duration.p")
> Plot.Phon.Dur.All()
> Boxplot.Phon.Dur.All()
```

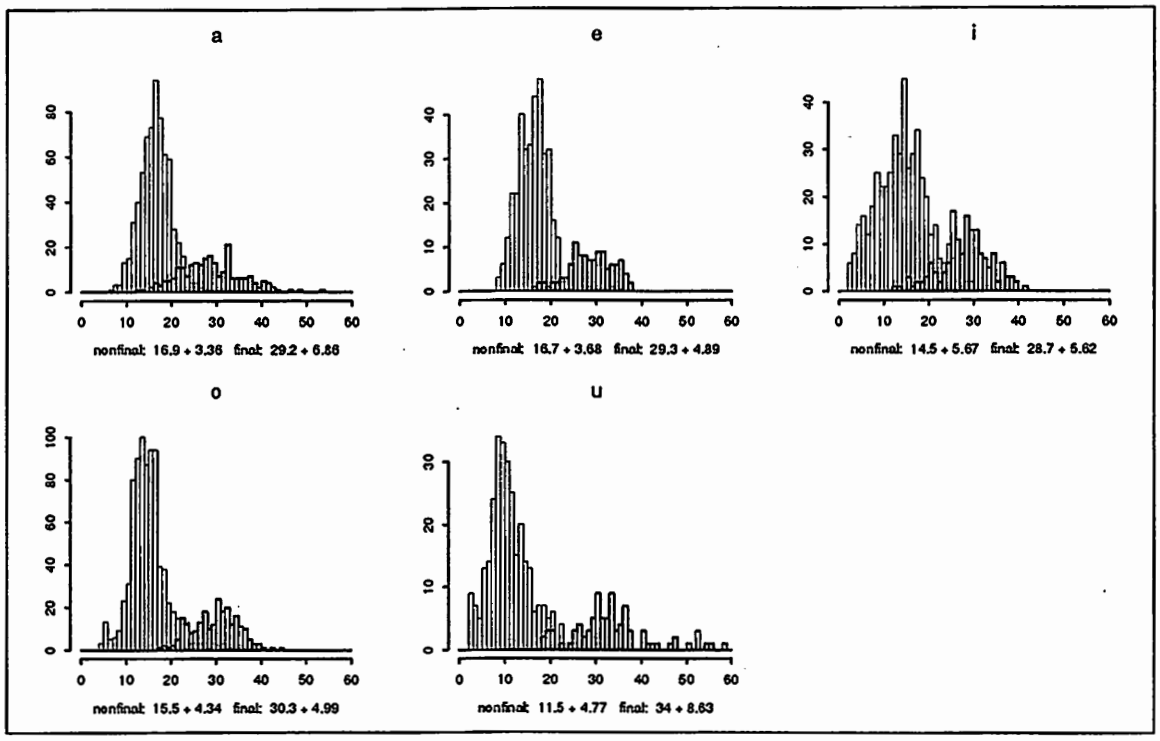

Figure 2.1: Histograms for vowel durations in non-final and final position

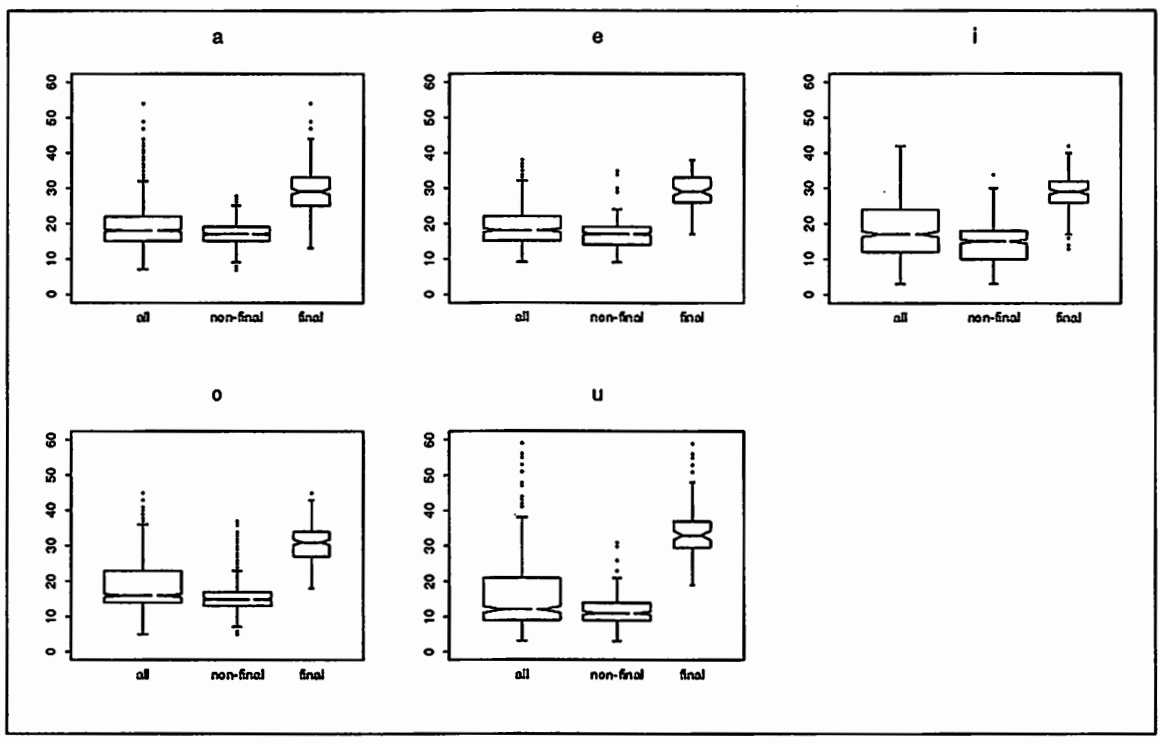

Figure 2.2: Boxplots for vowel durations in non-final and final position

Comments: We can see, that the durations vary considerably depending on the position of the phonemes.

The function `round()` is bugged, so we supply our own routine `Round()`.

Setting the breaks at integer values, here from 0 to 60, is important! Leaving the setting of breakpoints to the histogram function could produce breakpoints which are not spaced at integer intervals, so that the number of integer values differs from bin to bin.

2.2 Interface to C: Reading Speech Files

S can (more or less) easily be interfaced with C or FORTRAN routines in order to:

- speed up some slow S routine.
- reuse "tried and tested" C source code.
- do things impossible with S.

The following steps must be performed:

- write an S interface routine.
- write a C interface to an existing set of subroutines (which we will call *core routines*) like Waves+. As we only need the object file, we compile C code with `$ cc -c -G 0 foo.c`. The option `-G 0` is necessary: it makes sure, that no data will be accessed from the global pointer (see manual page for cc(1)).
- load `foo.o` into S with `> dyn.load("foo.o")`

Task: Read and plot a speech file, which is stored as a sequence of short integers.

Method:

binary/binary.p

```
ReadBinary _ function(Fname, Type, Start, Length){
# reads binary data from a file
        .C("SReadBinary",
                as.character(Fname),
                as.character(Type),
                as.integer(Start),
                as.integer(Length),
                result = as.double(double(Length)))$result
}
ReadShort _ function(Fname, Start, Length){
# reads file (sequence of short int)
        ReadBinary(Fname=Fname, Type="short", Start=Start, Length=Length)
}
ReadFloat _ function(Fname, Start, Length){
# reads file (sequence of float)
        ReadBinary(Fname=Fname, Type="float", Start=Start, Length=Length)
}
Length.File _ function(Fname){
# length of Fname in Bytes
        as.single(unix(paste("ls -lL '",Fname,"'|nawk '{print $4}'",sep="")))
}
Wave.Plot _ function(Fname){
# plots a speech file (sequence of short int)
        # gets number of samples in Fname
        Length <- Length.File(Fname = Fname)/2
        Data <- ReadShort(Fname, 0, Length)
        # to keep the plot symetric
        Range <- range(Data,  - Data)
        plot(Data, ylim = Range, type = "l", xlab = "nsample", ylab =
                "amplitude")
        # the 0 line in a different colour
        abline(h = 0, col = 2)
        title(Fname)
}
```

binary/binary.c

```
#include<stdio.h>
/* interface routine to S */
SReadBinary(p_fname,p_type,p_start,p_length,values)
char **p_fname;
char **p_type;
int  *p_start, *p_length;
double *values;
{
FILE *fp;
  int i,nread,size;
  short *sdata;
  float *fdata;
  char *fname = *p_fname, *type  = *p_type;
  if(ReadBinary(fname, *p_start, *p_length, type, values)==0){
    return(0);
  } else {
    return(1);
  }
}
/* "tried and tested" core routine */
static ReadBinary(fname,start,length,type,values)
char *fname;
int  start;
int  length;
char *type;
double *values;
{ FILE *fp;
  int i,nread,size;
  short *sdata;
  float *fdata;
  fp=fopen(fname,"r");
  if(fp==NULL){ fprintf(stdout,"(ReadBinary) Can't open %s for r\n",fname); return(1);}
  switch(type[0]){
  case 's':
    size = sizeof(short);
    sdata=(short *)S_alloc(length,size);
    fseek(fp,start*size,0);
    nread = fread(sdata,size,length,fp);
    for(i=0;i<nread;i++) values[i]=sdata[i];
    break;
  case 'f':
    size = sizeof(float);
    fdata=(float *)S_alloc(length,size);
    fseek(fp,start*size,0);
    nread = fread(fdata,size,length,fp);
    for(i=0;i<nread;i++) values[i]=fdata[i];
    break;
  default:
    fprintf(stderr,"(ReadBinary) wrong type %s\n",type);
    return(1);
    break;
  }
  fclose(fp);
  return(0);
}
```

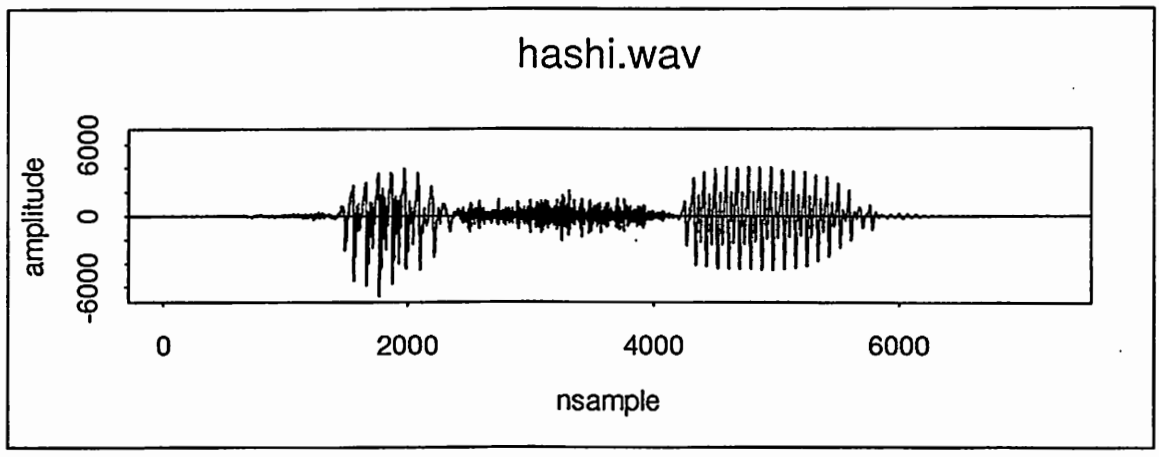

Figure 2.3: Speech Signal

After compiling with `$ cc -c -G 0 binary.c`, we created Figure 2.3 with

```
> dyn.load("binary.o")
> source("binary.p")
> Wave.Plot("hashi.wav")
```

Comments: The C function `S_alloc` is used instead of `calloc`. Avoid calling `exit()` inside your C subroutines.

Notice `Length.File()` as another example for interfacing S to UNIX.

2.3 Handling of Multi-Dimensional Arrays: Recognition Rates and Likelihoods

Task: Recognition experiments with 49 phonemes, varying the number of mixture densities. Desired outputs are likelihood and determinant for each phoneme, and accuracy and likelihood for each mixture. Results have to be analyzed and visually displayed.

Method: We will use only one 3-dimensional array containing all the results with suitably chosen dimension names. First, the results are formatted for easy inclusion into S. Each file `likelihood.data, determinant.data` has 49 rows. 5 columns contain results for 1, 5, 10, 15 and 20 mixtures.

array/array.p

```
Get.Dat _ function(){
# prepares a 3-dimensional data array
        # 49 phonemes * mixtures * ( likelihood, determinant, accuracy, correct, samples )
        Y <- array(0, c(49, 5, 5))
        dimnames(Y) <- list(phon=rep("",49),mix=rep("",5),,val=rep("",5))
        dimnames(Y)$phon <- scan("phon.text","")
        dimnames(Y)$mix  <- scan("mix.text","")
        dimnames(Y)$val  <- c("like","det","acc","corr","nsamp")

        Y[,,"like"] _ matrix(scan("likelihood.data"),ncol=5,byrow=T)
        Y[,,"det"] _ matrix(scan("determinant.data"),ncol=5,byrow=T)
        # accuracy and correct are same for all 49 phonemes!
        Y[,,"acc"] _ matrix(scan("accuracy.data"),nrow=49,ncol=5,byrow=T)
        Y[,,"corr"] _ matrix(scan("correct.data"),nrow=49,ncol=5,byrow=T)
        # number of samples is the same for each mixture!
        Y[,,"nsamp"] _ scan("nsamp.data")
        Y
}
Plot.Dat _ function(Y=Y){
        par(mfrow=c(1,2))
        # plot all data for 1 mixture (number of samples vs. likelihood for 1. mixture)
        plot(Y[,"mix1","nsamp"],Y[,"mix1","like"],type="n")
        text(Y[,"mix1","nsamp"],Y[,"mix1","like"],dimnames(Y)$phon)

        # plot detailed data for 1 mixture
        X _ (Y[,"mix1","nsamp"] < 600)&(Y[,"mix1","like"] < 500)
        plot(Y[X,"mix1","nsamp"],Y[X,"mix1","like"],type="n")
        text(Y[X,"mix1","nsamp"],Y[X,"mix1","like"],dimnames(Y[X,,])$phon)
}
Correlate.Dat _ function(Y=Y){
        sink(file="output")              # redirect output to file output
        for(I in c("like","det"))
        for(J in c("acc","corr")){
                C _ cor( apply(Y[,,I] * Y[,,"nsamp"],2,mean), Y[1,,J])
                cat("correlation between", I,"and",J,C,"\n")
                }
        C _ cor( Y[,,"like"],Y[,,"det"])
        cat("correlation between like and det\n")
        print(C)
        C _ cor( Y[1,,"acc"],Y[1,,"corr"])
        cat("correlation between acc and corr",C,"\n")
        sink()                           # redirect output to tty
}
```

```
> source("array.p")
> Y _ Get.Dat()
> Plot.Dat(Y)
```

```
> Correlate.Dat(Y)
```

The output looks like this:

array/output

```
correlation between like and acc 0.9577854
correlation between like and corr 0.9807262
correlation between det and acc -0.9664314
correlation between det and corr -0.9863711
correlation between like and det
           [,1]       [,2]       [,3]       [,4]        [,5]
[1,] -0.4560109 -0.3210325 -0.2395885 -0.2169071 -0.09069394
[2,] -0.4464950 -0.3230588 -0.2445051 -0.2235084 -0.09958541
[3,] -0.4433811 -0.3225494 -0.2489073 -0.2311191 -0.10898536
[4,] -0.4416984 -0.3227751 -0.2522230 -0.2368674 -0.11583763
[5,] -0.4404659 -0.3241981 -0.2579562 -0.2449849 -0.12546067
correlation between acc and corr 0.9949931
```

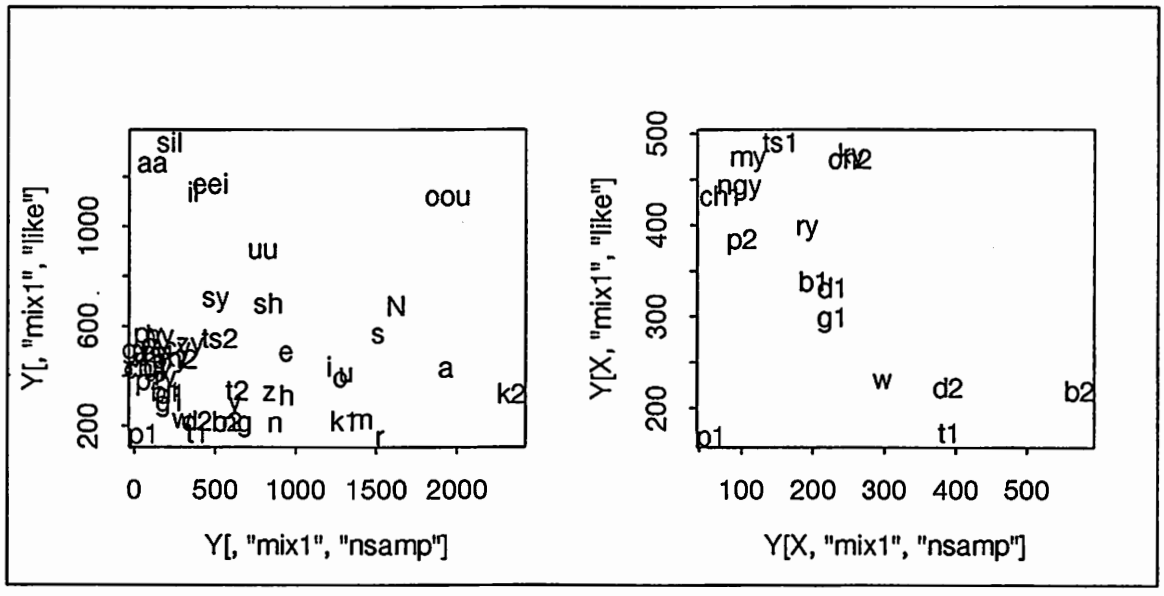

Figure 2.4: Likelihood vs. number of samples

Comments: Using `dimnames` makes it easier to avoid confusion.

2.4 Plotting Functions: HMM Mixture Gaussian Probabilities

Task: Show output probability distributions for multiple mixture Gaussian density HMM's.

Method: The following shell script extracts mixture weight, mean value and standard deviation for a specified cepstral dimension from an ATR-format HMM model.

gauss/gauss.awk

```
# extract mixture, mean and sigma from hmm-file
#       nawk -f gauss.awk mat=2 dim=1 hmm
/^mix/ {        if ( substr($1,5,1) == mat){
                        mix = $2
                        wanted = NR + dim }}
NR == wanted {  printf"%f %f %f\n",mix,$1,$2 }
```

gauss/gauss.p

```
Gauss.All _ function(X = seq(1, 20, length = 100)){
# plots gaussian mixture pdf's for 3 states for mixtures 1,3,10
# hmm1, hmm3 and hmm10 are hmm files in ATR format
        par(mfcol = c(3, 3))
        for(File in c("hmm1", "hmm3", "hmm10")) {
                for(M in 0:2) {
                        unix(paste("nawk -f gauss.awk mat=", M, " dim=1 ",
                                File, " > hmm.tmp", sep = ""))
                        Gauss.Plot(X, "hmm.tmp")
                        title(paste("File: ", File, "State: ", M))
                }
        }
}
Gauss.Plot _ function(X, File){
# plots a gaussian mixture pdf
        Y <- scan(File, list(mix = 0, mean = 0, sigma = 0))
        Dim <- length(Y$mix)
        plot(X, Gauss(X, Y$mix, Y$mean, Y$sigma), type="l", ylab="pdf", lwd=2)
        for(D in 1:Dim) {
                lines(X, Gauss(X, Y$mix[D], Y$mean[D], Y$sigma[D]))
        }
}
Gauss _ function(X, Mix = 1, Mean = 0, Sigma = 1){
# calculates values of a gaussian mixture pdf
        Dim <- length(Mix)
        R <- rep(0, length(X))
        for(D in 1:Dim) {
                R <- R + Mix[D]/sqrt(2 * pi * Sigma[D]) *
                    exp( - (X - Mean[D])^2/(2 * Sigma[D]))
        }
        R
}
```

```
> Gauss.All()
```

Comments: In reality, we should print all 34 dimensions, but this would give a page of at least 34 dim * 3 mat pictures, which is too dense. Here, we only printed the first dimension, i.e. the power term.

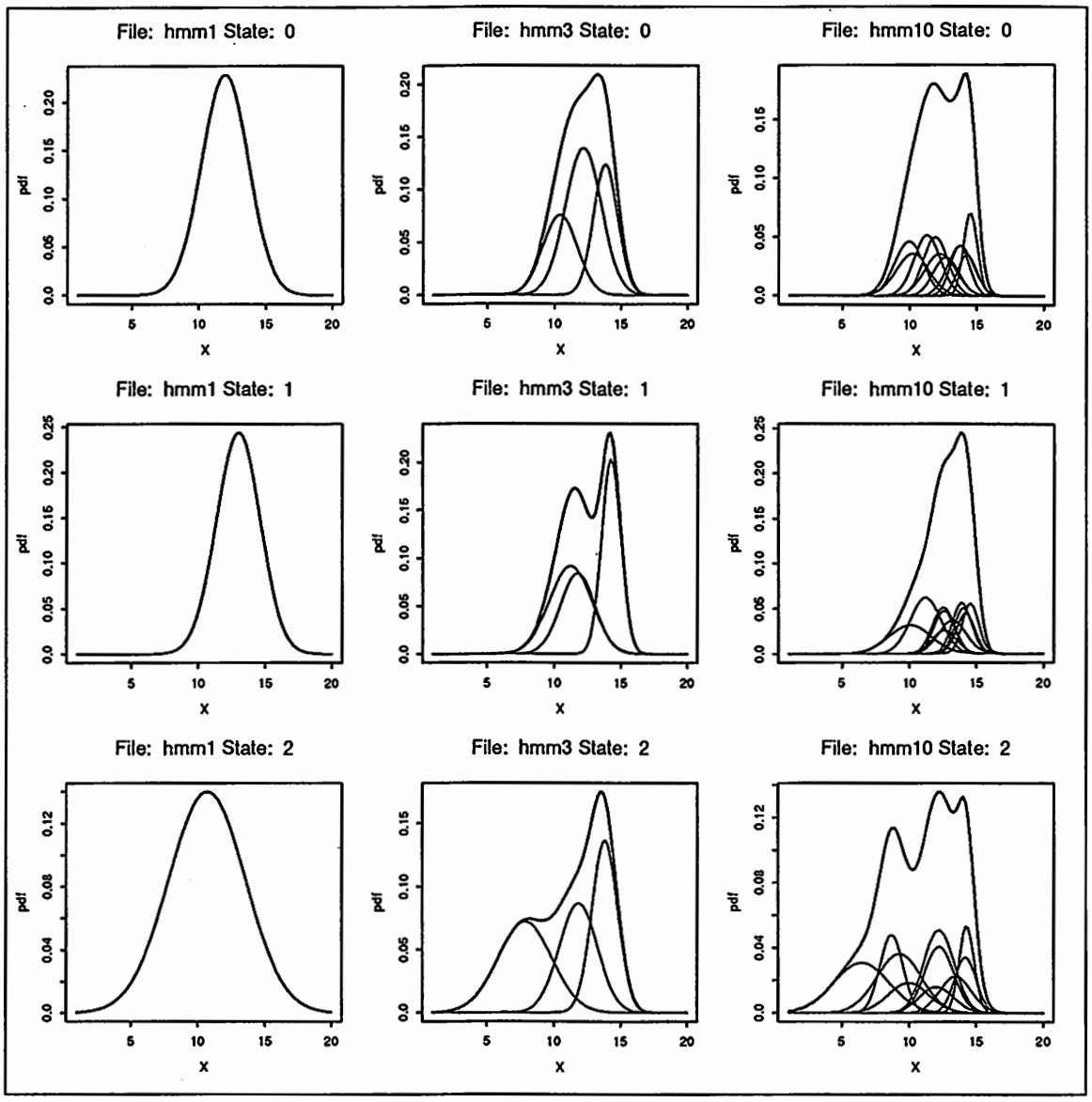

Figure 2.5: Mixture Gaussian pdf's for 3 states and 1,3,10 mixtures

2.5 Printing Output

2.5.1 Interface to TEX

S can create postscript format graphics files. There is no problem, when sending these files directly to the printer, but when using EPSF to include the .ps file in a TEXor LATEXdocument the picture is not positioned correctly. A simple UNIX filter solves this problem:

sps

```
#!/bin/csh -f
awk '/%%BoundingBox/{for (i=2;i<=5;i++)$i+=36}{print}'
```

Nevertheless, there can still be a problem with the proportions of the plot: in the EPSF command you can specify width and height. If the original porportions (as specified in `.Xdefaults` or `/usr/lib/X11/app-defaults/SPlus` with `splus*Canvas.width` and `splus*Canvas.height`) were different, characters will appear distorted along x and y-axis.

EPSF files are included in a LATEX document with e.g.

```
\begin{figure}[bht]
        \fbox{\epsfile{file=print/print.ps,width=150mm}}
        \caption{Speech File}
\end{figure}
```

The style file `epsf.sty` must be on the TEX search path and is included by

```
\documentstyle[epsf]{report}
```

at the beginning of the LATEXdocument.

The following function supports the creation of undistorted pictures for a LATEX document.

print/print.p

```
Print.PS.TEX _ function(Function, W = 8, H = 8, Outfile = "gomi.ps", StartX11 = T){
# prints for TEX input in specified width and height
# no shrinking of letters !
        pscript(width = W, height = H, print = F, horizo = F, paper = "a4")
        cat("writing", Outfile, "\n")
        Function()
        pscript(print = F)
        C <- unix(paste("sps < PostScript.out >", Outfile, "; rm PostScript.out"))
        if(StartX11 == T)  X11()
}
```

Figure 2.6 was created with the following commands:

```
> source("print.p")
> source("../binary/binary.p")
> dyn.load("../binary/binary.o")
> Tmp.func _ function() Wave.Plot("../binary/hashi.wav")
> Print.PS.TEX(Tmp.func,W=10,H=3,Outfile="print.ps")
```

Comments: Instead of plotting to the X11 window, the plot is written to a file `PostScript.out`, which is then renamed. We have to supply a function without any argument. Avoid the `height` option in the TEX `epsfile` command!. You would probably redistort the postscript picture.

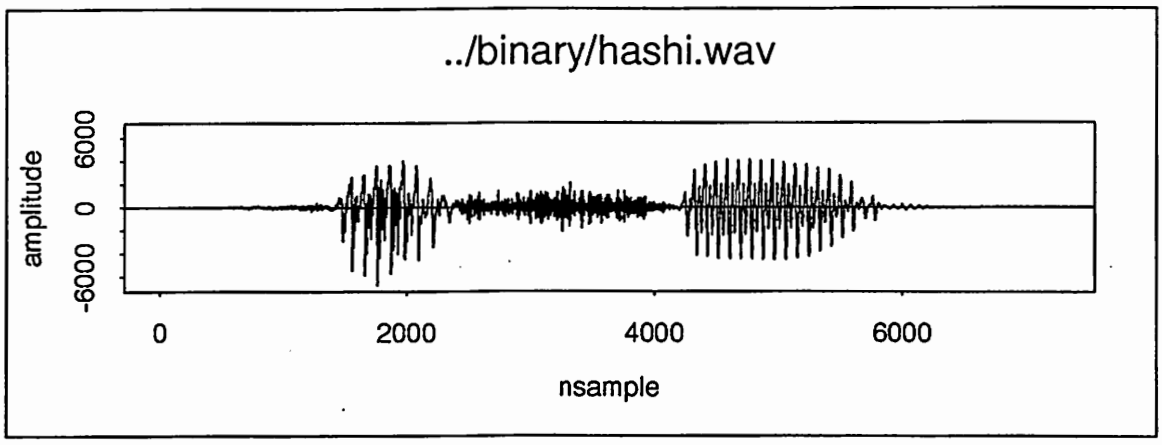

Figure 2.6: Speech File

2.5.2 The Art of Customizing Plots

Graphic parameters can either be specified by `par()` or as an argument to one of the plotting function. Some useful parameters are listed below:

`lwd`	linewidth
`axes=F`	no axes
`col=2`	specifying colour 2
`xpd=T`	text etc. may be plotted outside of plot region
`cex=1.5`	normal character size * 1.5
`xaxs`	style of x-axis label calculation
`xaxt='n'`	set-up of x axis but don't plot
`mfrow=c(2,3)`	multiple figures in 1 plot with 2 rows, 3 columns by row
`mfrow=c(2,3)`	multiple figures in 1 plot with 2 rows, 3 columns by column

2.6 3-Dimensional Graphics: Speech Spectrogram

The language S was not designed to be a signal processing package. Nevertheless it is fairly simple to plot data, even 3-dimensional data or spectrograms. As explained above, complicated subroutines can be written in C and dynamically loaded. Several functions permit visualization of 3-D graphics: `contour()`, `image()`, `perspective()`

Task: Read 256-dimensional DFT values and plot spectrogram .

Method:

spectrum/spectrum.p

```
Plot.Spectrum _ function(Fname="hashi.dft",DFTlength=256,Freq=6){
# Fname is assumed to contain DFTlength*Ncol float data
        Length _ Length.File(Fname)/4
        Ncol _ Length/DFTlength
        All _ list(x=1:Ncol,y=seq(0,Freq,length=DFTlength))
        All$z _ matrix(ReadFloat(Fname,0,Length),ncol=DFTlength,byrow=T)
        par(mar=c(3,2,0,0))          # make margins around plot smaller
        image(All)
}
```

```
> source("spectrum.p")
> source("../binary/binary.p")
> dyn.load("../binary/binary.o")
> Plot.Spectrum()
```

Comments: The spectrogram can also easily be displayed in colours on your workstation if the appropriate commands are included in `/.Xdefaults` (see Appendix A.3).

To calculate the DFT from a speech file or a Cepstrum file you have basically 3 options:

- Use a completely separate program, which you can call from inside S with `unix()`.
- Write S and C interfaces to existing C subroutines (see section 2.2).
- Rewrite FFT in S language code (see section 2.8).

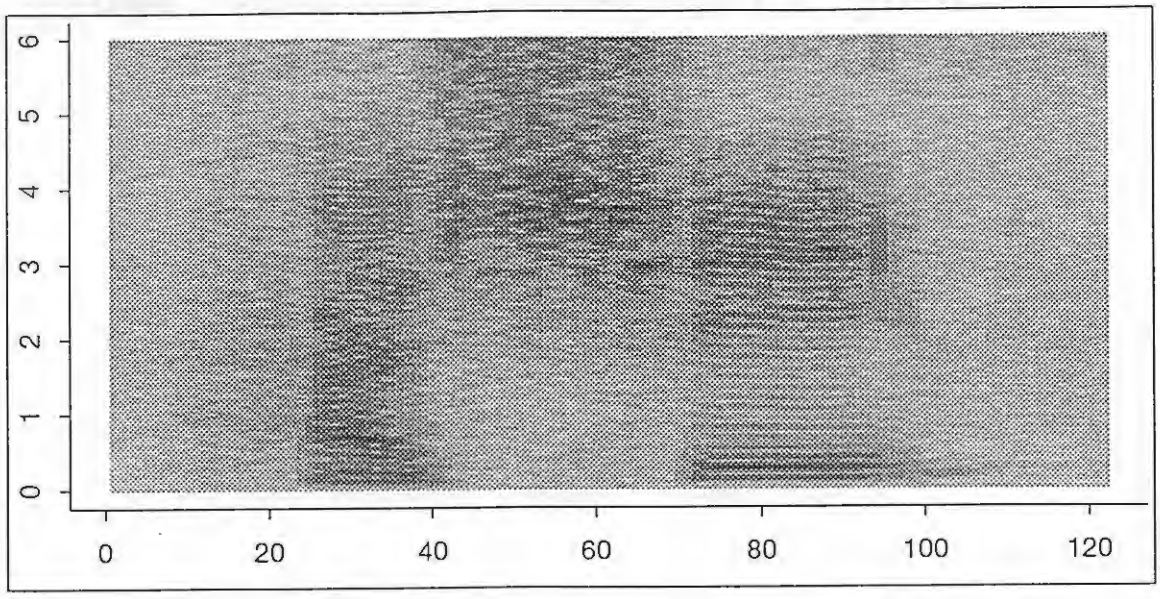

Figure 2.7: Spectrogram for "hashi"

2.7 Visualizing 2 Dimensions: Covariance Matrix for Pitch and Cepstrum

Another useful way of displaying 3-dimensional numeric information, often chosen for activation values of neural networks, is by size of rectangles.

Task: We previously calculated the correlation between pitch and a 17-dimensional feature vector containing 1 power value and 16 cepstral values [Singer 91] for two different data sets (different speaking style) and saved these values to a file. We want to display all of these values in 1 table with rows indexing the phonemes and columns indexing the features.

Method:

cov/cov.p

```
Matrix.Plot <- function(Mat, Normal = F){
# variables:    Mat     2-dimensional matrix, normalized between -1 and 1
#               Normal  logical value, if True y runs from bottom to top
#                                       if False y runs from top to bottom
#                       (use True for correlation matrices)
        col <- ncol(Mat)
        row <- nrow(Mat)
        print(paste("col", col, "row", row))
        x <- rep(1:col, rep(row, col))
        if(Normal)    y <- rep(1:row, col)
        else          y <- rep(row:1, col)
        v <- as.numeric(Mat)
        xs <- x[v >= 0]
        ys <- y[v >= 0]
        vs <- v[v >= 0]
        if(length(vs)) symbols( xs, ys, squares = vs,
                 inches=FALSE,ylim=c(1,row),xlim=c(1,col),axes=F,add=T)
        xc <- x[v < 0]
        yc <- y[v < 0]
        vc <- v[v < 0]
        if(length(vc)) symbols(xc, yc, circles =  - vc/2,
                                inches = FALSE, add = TRUE)
}
Cor.Matrix.Plot _ function(Type="DSA",Dim=16,Phon=24){
# plots correlation values for each phoneme
# file format: label followed by Dim+1 correlation values
#              for pow and Dim cep
        Name _ paste("MHT","_",Type,".",Dim,".cor",sep="")
        x _ scan(Name,"")                          # read everything as character
        Dim _ Dim+1
        # setup the plot
        plot(c(1,Dim),c(1,Phon),type="n",xlab="",ylab="",
                xpd=T,lab=c(Phon,Dim,1),xaxs="r",yaxt="n",xaxt="n",cex=1.5)
        # extract only character strings; these are the row labels
        names _ x[is.na(as.double(x))]
        text(-1.2,Phon:1,names,cex=1.3)            # labels for y-axis
        text(1:Dim,-0.5,c(1,1:(Dim-1)),cex=0.5)  # numbers and labels for x-axis
        text(1,-1.2,"pow",cex=1.5)
        text(Dim/2+0.5,-1.2,"cep",cex=1.5)
        x _ matrix(x[!is.na(as.double(x))],      # change character to matrix
                        ncol=Dim,nrow=Phon,byrow=T)
        Matrix.Plot(x,Normal=F)
        title(Name)
}
Cor.Matrix.All _ function(){
        par(mfrow=c(1,2))
        Dim _ 16
        for( Type in c("DSB","DSC"))    Cor.Matrix.Plot(Type=Type,Dim=Dim)
}
```

```
> source("covariance.p")
> Cor.Matrix.All()
```

Comments: Several tricks are used here. One problem was, that the data is a mixture of character strings (labels) and numeric data (correlation values). An unsolved problem is that no filled rectangles are provided in the function `symbols()`. We thus settled on circles to depict negative correlation values.

The large values for phoneme "p" of data set MHT_DSB stem from the fact, that there were only 2 voiced frames in the data leading to correlation values of plus and minus 1.

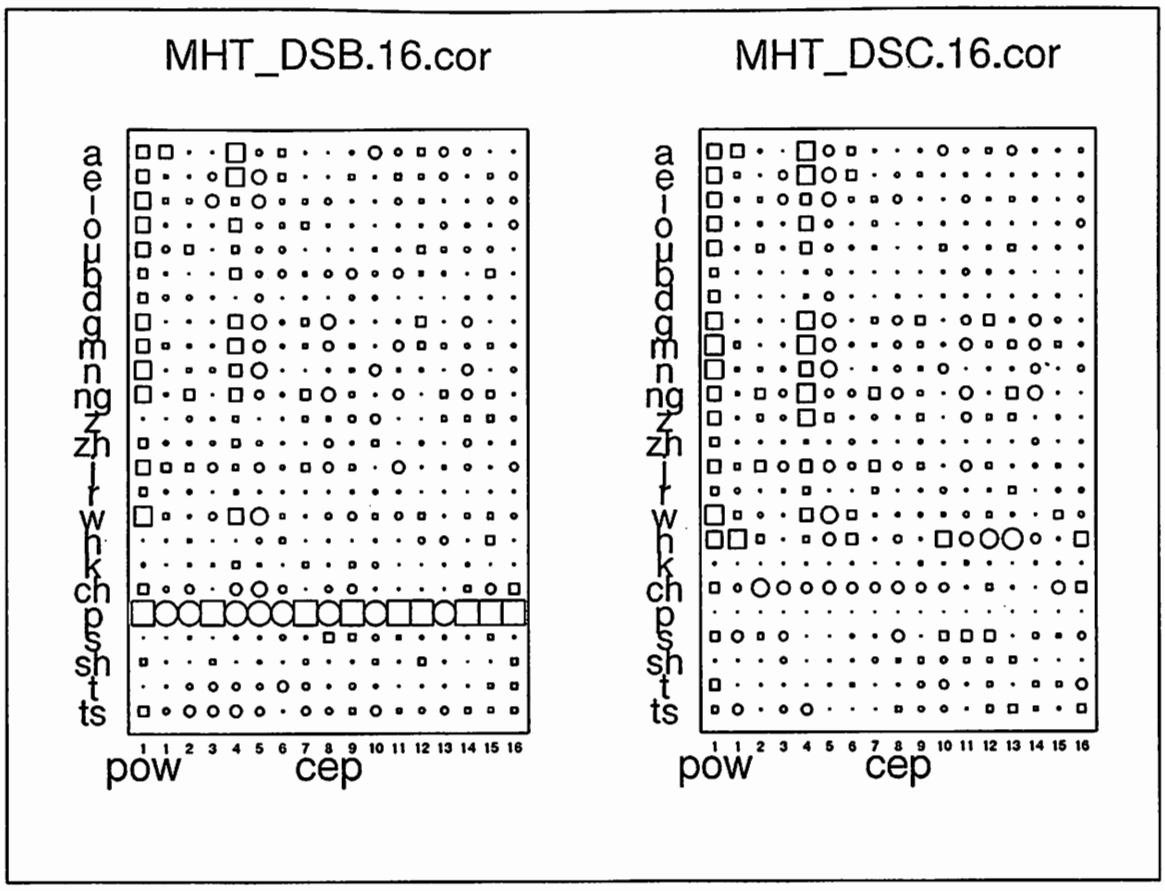

Figure 2.8: Correlation matrix between cepstrum and pitch

2.8 Avoiding Do-Loops: Delta Cepstrum as a Digital Filter

Because of the implementation of S, do-loops are very time consuming and should be aboided where possible. They also pose problems with memory management.

Task: In recent years, features for speech recognition also use differential time information, specifically the Cepstrum differentiated with respect to time. This so-called Delta Cepstrum can also be regarded as a linearly filtered Cepstrum [Furui 86].

We are interested in the transfer characteristics of this filter to optimize recognition results.

Method: Calculate and display the transfer characteristics of the "Delta Cepstrum Filter" (i.e. the Fourier transform of the Delta Cepstrum window coefficients.

delta/delta.p

```
Delta.Transfer.Plot _ function(Halflength = 10){
# Transfer Function for triangular window weighted Delta Cepstrum
        Window _ c(0:Halflength, (Halflength - 1):0)         # triangular window
        par(mfrow=c(2,1))
        Transfer.Plot(Window)
}
Transfer.Plot _ function(Window){
# Window is supposed to be symmetrical
        tmp <- as.integer( - length(Window)/2)
        # construct Fourier transformation matrix
        X <-  - tmp:tmp
        Y <- seq(0, pi, len = 256)
        # temporary function definition
        Myexp <- function(x, y)          exp(x * y * (1i))
        # outer "product"
        Z <- outer(X, Y, FUN = Myexp)
        # create input sequence to be transformed
        Weight <- Window * ( - tmp:tmp)
        # plot Weight values
        plot(-X,-Weight,type='n',main="Delta Window and Regression Coefficients");
        lines(-X,Window,col=4,lwd=2)
        abline(h=0); abline(v=0)
        for(x in -X){
                lines(c(x,x),c(0,-Weight[x-tmp+1]),col=2,lwd=2)
                points(x,-Weight[x-tmp+1],col=2,cex=2)
        }
        # Fourier transform
        Result <- Mod(as.vector(Weight %*% Z))
        # plot dB against normalized frequency
        plot(Y/pi, 20 * log10(Result/max(Result)),ylim = c(-40, 0), xlab= "frequency in pi",
                ylab = "gain (dB)", type = "l",   main = "Transfer Function")
}
```

```
> source("delta.p")
> Delta.Transfer.Plot()
```

Comments: We could have used a do-loop to calculate the power spectrum at every frequency point. We avoided this by using `outer()` to construct a Fourier transform matrix. The Fourier transformation was then performed as a simple matrix multiplication.

Other useful functions for avoiding do-loops are `apply`, `rbind`, `cbind` etc.

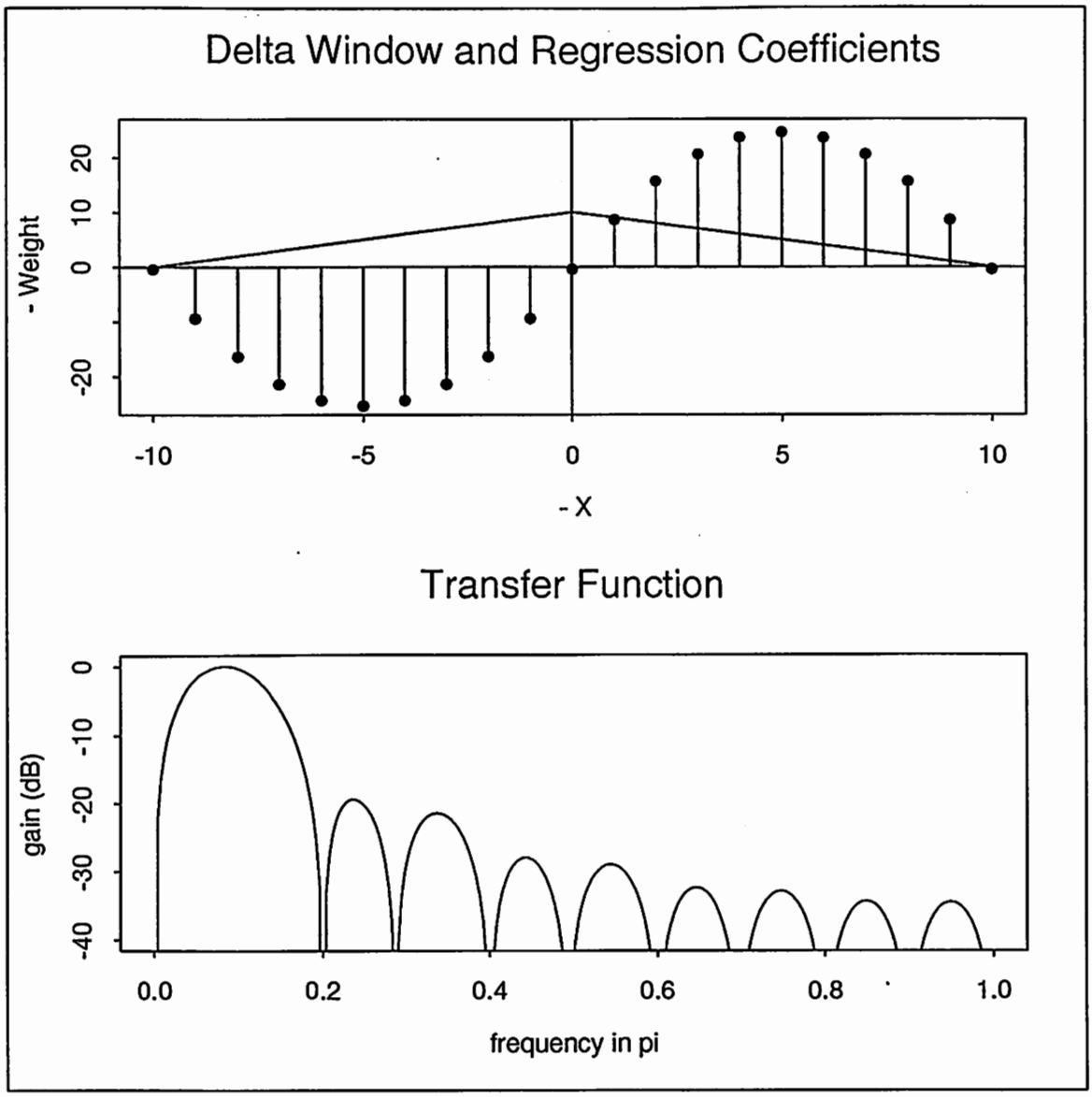

Figure 2.9: Transfer function for triagonal delta cepstrum

2.9 Running in Batch Mode

You can also run S as a batch program. Some of the reasons for running S in batch mode follow:

- Processing takes a long time
- To get around the memory freeing problem. Basically, you write the do loop inside a shell script which then runs S separately for each iteration.

One problem is that the normal S batch program runs S as a background process, i.e. the shell script would not wait until the S process finishes. We would thus get many S processes running in parallel. We therefore edited the standard batch program `SHOME/cmd/BATCH` by simply deleting the `&` at the end of the 4th row.

batch/BATCH

```
case $#
in
2) BATCH=yes; export BATCH
(echo "invisible(options(echo=TRUE))"; cat $1)| nohup Splus >$2 2>&1 &
;;
*) echo "Usage:
Splus BATCH inputfile outputfile"
;;
esac
```

An example of `BATCH.FG` (FG for foreground) is demonstrated in this (admittingly boring) shell script.

batch/batch.csh

```
#!/bin/csh -f
# batch.csh H.Singer 22MAY92
set PHON = ( a e i o u)
set LOGFILE = S.log                    # logfile for Splus
if(-e $LOGFILE) rm $LOGFILE
foreach PH ( $PHON )                   # loop over all phonemes
        # create a temporary Splus command file as "here" document
        # "$" and "\" have to be escaped with "\";
        # else they are interpreted as shell variables  !!!
        cat > tmp.p << EOF_HERE
                Phon _ "$PH"           # put your commands here
                memory.size()
EOF_HERE
        Splus BATCH.FG tmp.p tmp.log   # run Splus as "foreground" batch
        cat tmp.log >> $LOGFILE        # save log messages
        rm tmp.p tmp.log               # clean-up
end
```

2.10 Unsolved Problems and Bugs

- Do-loops are very slow and should be avoided.
- Serious memory management problems, even leading to system crash.
- It is not clear how to document user-created functions. The `help()` mechanism and `prompt` are quite awkward (see [Becker 88], pp. 551).
- Graphics can be displayed in color. Nevertheless, it is not possible to create color graphics postscript files in Splus version 2.3. Splus Version 3.0 supports color postscript.

Bibliography

[Becker 84] Becker, R. et al.: "S An Interactive Environment for Data Analysis and Graphics (I and II)", Kyouritsu Shuppan,1987.(in Japanese)

[Becker 88] Becker, R. et al.: "The New S Language", Wadsworth, Pacific Grove, California,1988.

[Chambers 92] Chambers, J. et al.: "Statistical Models in S", Wadsworth, Pacific Grove, California,1992.

[Furui 86] S. Furui, "Speaker-Independent Isolated Word Recognition Using Dynamic Features of Speech Spectrum," IEEE Trans. ASSP, Vol.ASSP-34, pp.52-59, 1986.

[Singer 91] H.Singer, S.Sagayama:"Use of Correlation between Pitch and Spectral Parameters for HMM Phoneme Recognition", IEICE Technical Report, SP91-57, 1991.

Appendix A

Installation of S at ATR Speech Processing

A.1 Minimal Installation

Login on your host and type the following commands

```
$ cd /usr/local/splus
$ su
$ Splus.install
$ exit

$ Splus < 'Splus SHOME'/adm/test/NEW/plot.X11
```

To get correct font size in your graphics window, you also should have a file `.X11Startup` containing at least the following lines:

model.X11Startup

```
xset +fp /usr/lib/X11/fonts/decwin/75dpi
xset fp+ /usr/lib/X11/fonts/MIT
xset fp+ /usr/lib/X11/fonts/compX10
xset fp  rehash
exit
```

A.2 Installation for GNU Emacs

Using Emacs is so much better than working with plain S, that we strongly recommend you to take the extra steps to use Emacs. Please enter the following commands:

```
$ mkdir ~/emacslib
$ cd ~/emacslib
$ cp /NFS/atr-fs/pub1/local/splus/library/emacslib/comint.el .
$ cp /NFS/atr-fs/pub1/local/splus/library/emacslib/S.el .
```

Include the following lines in your `.emacs` startup script:

rel.emacs

```
;;; for S
(load "~/emacslib/comint.el")
(autoload 'S "S" "" t)
```

You can now start S from inside Emacs with `M-x S`. Make sure, that your `.cshrc` file contains something like these lines:

rel.cshrc

```
setenv EMACSLOADPATH "$HOME/emacslib:$EMACSLOADPATH"
setenv ESHELL "/bin/csh"
set path=( $path /usr/local/bin )
```

A.3 Installing Colors

See Appendix 1 of the User's Manual or copy relevant lines from `model.Xdefaults`.

color.Xdefaults

```
/* The following settings are for color monitors with large colormaps   */
/* (64 or more entries). The image scheme is identical to that          */
/* specified by the S-PLUS data set "xcm.heatA".                        */

splus*colors : black yellow cyan magenta green MediumBlue red \
               black 11 blue 12 magenta 12 red 6 yellow 6 white
splus*firstImageColor : 7
splus*nHalftones      : 0
splus*halftonePolygon : No
splus*halftoneImage   : Yes

/* The backslash (\) must be the last character on the line, it         */
/* continues the colors list.                                           */

/* This colormap uses the first 7 colors for regular plotting, and rest */
/* of the color specification create a ramp of smoothly blending colors */
/* The splus*firstImageColor tells the image command to start with the  */
/* 7th color when creating images.  The first color listed is used as   */
/* the background color, and isn't counted in the first image color.    */

/* Set the colors used in the S-PLUS X11 control panel                  */

splus*background          : blue
splus*Text*background     : #a0f
splus*Text*foreground     : yellow
splus*Label*background    : cyan
splus*Label*foreground    : red
splus*Command*background : red
splus*Command*foreground : cyan
```

Appendix B

Useful Source Files

Additional programs for signal processing were implemented and installed: Spectrogram (from Cepstrum file), pitch, labelling information.

These routines are collected in libraries and can be loaded with

```
>library(signals)
>library(help="signals")
>dyn.load(paste(unix("echo $SHOME/library"),"/signals/signals.o",sep=""))
```

The following file `util.p` contains some useful utility functions, also accessing the `signals` library (specifically the functions `Write()`, `Extract()`, `Extracts()`, `Extractf()`).

```
# H.Singer    15DEC91
#             13JAN92 AD/DA routines
#             11MAY92 additional comments and renaming
#

.First _ function(){
# this function is automatically called when starting Splus
        # load library routines
        library("signals")
        # load C code for signals
        dyn.load(paste(unix("echo $SHOME/library"),"/signals/signals.o",sep=""))
        # start X11 graphics window
        X11()
}

Tempfile.Create _ function(Pattern = "tmp"){
# creates filename for temporary file
        unix(paste("echo ", Pattern, "$$", sep = ""))
}

Loc.X _ function(){
# print x position(s) at position(s) of left mouse button click
        text((tmp <- locator()), signif(tmp$x, 3))
}

Loc.Y _ function(){
# print y position(s) at position(s) of left mouse button click
        text((tmp <- locator()), signif(tmp$y, 3))
}

Loc.Text _ function(Text="gomi",cex=1){
# print text at position of left mouse button click
```

```
# cex designates size
        text(locator(),Text,cex=cex)
}

Scan.ASCII.Matrix. _ function(File){
# reads in "well-formed" ASCII numeric data
        n <- as.numeric(unix(paste("sed 1q", File, "| wc -w")))
        matrix(scan(File), ncol = n, byrow = T)
}

H _ function(Function){
# help on an xterm window
        if(!missing(Function) && !is.character(Function))
                Function <- substitute(Function)        # Function is not yet evaluated
        if(is.language(Function) && !is.name(Function))
                Function <- eval(Function)
        if(Function == "/" || Function == "%/%")
                Function <- "Arithmetic"
        Function <- paste("'", Function, "'", sep = "")
        help(Function, pager = "view", window = T)
}

Home _ function(File){
# creates absolute path filename
        File <- paste("$HOME/", File, sep = "")
        unix(paste("echo", File))
}

Norm.Mat _ function(Mat){
# normalizes matrix with row means by rows
        mat.mean <- apply(Mat, 1, mean)
        sweep(Mat, 1, mat.mean)
}

Length.File _ function(Fname){
# length of Fname in Bytes
# -L to resolve symbolic links
        as.single(unix(paste("ls -lL '",Fname,"' | nawk '{print $4}'",sep="")))
}

Cut.File _ function(Fname1, Fname2, Start, Length, Type="short"){
# cuts specified part of Fname1 into Fname2
# C functions Write and Extract have to be loaded (dyn.load)
        Write(  fname=Fname2,
                type=Type,
                length = Length,
                Extract(fname=Fname1,type=Type,start=Start,length=Length),
                permission = "w")
        cat("File", Fname2, "written with length",Length,"\n")
}


Plot.Short _ function(File){
# plots a complete file (short integer), e.g. wave file
        plot(Extracts(File,0,Length.File(File)/2),type="l",xlab="samples")
}

Plot.Float _ function(File){
# plots a complete file (float), e.g. pitch file
        plot(Extractf(File,0,Length.File(File)/4),type="l",xlab="samples")
}

Get.Short _ function(File){
# reads a complete file (short integer), e.g. wave file
        Extracts(File,0,Length.File(File)/2)
```

```
}

Get.Float _ function(File){
# reads a complete file (float), e.g. pitch file
        Extractf(File,0,Length.File(File)/4)
}

Print.PS _ function(File="gomi.ps",Printer="img",Out=FALSE,Horizontal=FALSE){
# prints graphics on the X11 device
# put this in at the end of a for-loop
# UNIX filter sps must be in your $PATH
        printgraph(     method="postscript",
                        command=paste("(sps > ",File,")<",sep=""),
                        horizontal=Horizontal,
                        onefile=FALSE,
                        paper="a4")
        if(Out){
                print(paste("Printing",File,"at",Printer))
                unix(paste("lpr -P",Printer," ",File,sep=""))
        }
}

Print.PS.TEX _ function(Function,W=8,H=8,Outfile="gomi.ps",StartX11=T,Mar=c(5.1,4.1,4.1,2.1)){
# prints for TEX input in specified width and height
# no shrinking of letters! useful in batch mode!
# Function cannot take any arguments right now -> define temporary function
# UNIX filter sps must be in your $PATH
        pscript(width=W,height=H,print=F,horizo=F,paper="a4")
par(mar=Mar)
        cat("writing",Outfile,"\n")
        Function()
        pscript(print=F)
        C _ unix(paste("sps < PostScript.out >",
                        Outfile,"; rm PostScript.out"))
        if(StartX11==T) X11()
}

Round _ function(Value, Precision = 3){
# round is bugged, so this is a replacement
# you can only specify the number of precision
        format(signif(Value, Precision))
}

Recover.S.Functions _ function(pos=1){
# gets all S Functions in pos of .Search.List
# useful if you lost your ASCII source codes
# problematic if you have to many files in this directory!
        X _ ls(pos=pos)
        for(i in X){
                # makes sure that you only have function objects!
                if ( is.function(get(i)) ){
                        cat("\n# ************************************************\n")
                        cat(i," <- ")
                        print(get(i))
                }
        }
        invisible(X)
}

Plot.Spectrum _ function(Fname="gomi",DFTlength=256,Freq=6){
# plot spectrum as spectrogram
# file Fname is assumed to contain spectral data
# as DFTlength*Ncol float data
        Length _ Length.File(Fname)/4
        Ncol _ Length/DFTlength
```

```
        All _ list(x=1:Ncol,y=seq(0,Freq,length=DFTlength))
        All$z _ matrix(Extractf(Fname,0,Length),ncol=DFTlength,byrow=T)
        image(All)
}

# ADDA functions assume that an AD/DA box is connected and
# that the following commands are in $PATH
#         DA: daout -f <freq> <file>
#         AD: adin -f <freq> -t <time in sec> <file>

Record.Cut.ADDA _ function(Fname="gomi",DA=T){
# record, check and cut signal to create filename
        TmpFile _ Tempfile.Create("tmp")
        par(mfrow=c(3,1))
        while(1){
                Record.ADDA(TmpFile)
                Plot.Short(TmpFile)
                if(DA)  Play.ADDA(TmpFile,MB=T)
                print("if OK click MB, if not OK click left MB then middle MB")
                if(is.null(locator()$x)) break
        }
        Cut.ADDA(TmpFile,Fname)
        unix(paste("rm",TmpFile))
        Plot.Short(Fname)
        if(DA)  Play.ADDA(Fname)
}

Record.ADDA _ function(Fname="gomi",Time=2,Frequency=12,MB=T){
# record signal in Fname; default time is 2 seconds; default
# sampling frequency 12 kHz
        if(MB){
                print("change amplifier switch to TAPE1")
                print("click middle MB to start AD")
                locator()
        }
        print("AD started")
        unix(paste("adin -f",Frequency,"-t",Time,Fname))
        print("AD finished")
}

Play.ADDA _ function(Fname="gomi",Frequency=12,MB=F){
# play back Fname
        if(MB){
                print("change amplifier switch to TUNER")
                print("click middle MB to start DA")
                locator()
        }
        unix(paste("daout -f",Frequency,Fname))
}

Cut.ADDA _ function(Fname1="in",Fname2="out"){
# plot and cut signal in Fname1
        Plot.Short(Fname1)
        print("click left MB on start and end, then click middle MB")
        X _ as.integer(locator()$x)
        print(X)
        L _ X[2] - X[1]
        Cut.File(Fname1=Fname1,Fname2=Fname2,Start=X[1],Length=L)
}
############################## end of file ###################################
```