TR－A－0136

UNIX上の音声研究用ツール
－ *SpeechTools* －

天白 成一

1992. 3. 9

ATR視聴覚機構研究所

〒619-02 京都府相楽郡精華町光台2-2 ☎07749-5-1411

ATR Auditory and Visual Perception Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Telephone: +81-7749-5-1411
Facsimile: +81-7749-5-1408

も　く　じ

付録．

第１章　はじめに

概要

音声・聴覚の研究においては、音響信号波形をA/D変換して計算機上に取り込み、その波形やスペクトルあるいは信号処理結果を表示したり、波形やスペクトルを加工／編集したり、加工／編集した信号をD/A変換して聴いたりすることが必要であり、このような一連の作業を効率良く行えるか否かは大きな問題である。そこで、このような作業を支援するために音声研究用ソフトウエアツール(SpeechTools)開発してきた。

SpeechToolsは、広くUNIXをＯＳとする計算機上で利用することができ、音声分析・合成に関した各種のアルゴリズムを単一のインタフェースを通して使用できるコマンドと種々のデータ形式の変換等を行うユーティリティとそれらを支えるライブラリによって構成されている。

SpeechToolsの対象分野としては音声の研究全般を考え、音声分析・合成等に関した確立された技術を提供することを主目的とした。そして、SpeechToolsは大きな一つのプログラムではなく、小さな機能ごとに個別に分けられたコマンドの集合体として設計されており、コマンドを連結することでより多くの目的に利用できるようになっている。また、データ変換などを行うユーティリティとしては、acat（アスキーデータからバイナリデータへの変換）、bcat（バイナリデータからアスキーデータへの変換）、bcalc（２項演算を２つバイナリデータファイルに実行）、cv（各種バイナリデータのフォーマット変換）、merge（アスキーデータの列の併合）、separate（アスキーデータの列の分割）、smath（アスキーデータの科学算術演算）の７個があり、ライブラリとしては、libcommand.a（ファイルの入出力、コマンドのインタフェース等を含む）、libmx.a（行列演算を含む）、libst.a（音響分析・合成、DFT、高次方程式の解法等を含む）、libtlist.a（簡易なリスト処理を含む）の４個がある。プログラムは、すべてＣ言語で記述されており、総行数は約２万行である。また、マニュアルは、すべてroff形式で記述されており、オンラインで読むことができる。

利用方法

SpeechToolsは、通常/usr/local/stにインストールされ、コマンドとユーティリティは/usr/local/st/bin にインストールされるので、各自のホームディレクトリのファイル'.login'や'.cshrc'のコマンドのサーチパスに、/usr/local/st/binを加えなければならない。また、ライブラリとヘッダファイルは、それぞれ/usr/local/st/libと、/usr/local/st/includeにインストールされており、SpeechToolsのライブラリを用いて新たにコマンドを開発する場合は、これらのディレクトリに注意しなければならない。同様に、SpeechToolsのマニュアルは/usr/local/st/manにインストールされている。

本稿の構成

本稿は、５章から構成されている。第２章では、SpeechToolsのコマンドで実現されている一貫した使用方法を基本編で説明し、応用編ではシェルスクリプトによる利用例について説明する。第３章では、SpeechToolsのユーティリティに関して、第４章では、SpeechToolsのライブラリに関して簡単にそれぞれ説明する。第５章では、まとめとしてSpeechToolsの問題点と今後の方向性等について述べる。そして、付録にはマニュアルの目次とマニュアルの一部を付けた。

第２章　コマンドの使い方

この章では、SpeechToolsの多くのコマンドで実現されている一貫した使用方法を、基本編と応用編に分けて説明する。基本編では、一貫した使用方法について、応用編では、利用例を通じて使い方を説明する。

２．１　使い方（基本編）

このSpeechToolsのコマンドの多くは、一貫した同じ方法で使うことができる。以下の説明では、下線部がユーザが入力している部分で、% はUNIXのプロンプトでありユーザの入力を促している。

①UNIXのプロンプトの出ている状態でコマンド名を入力する。

```
% fft_run
Calculate DFT running spectra.
usage :: fft_run filename
         fft_run -o arguments

Defaults are as follows.
SAMPLING FREQUENCY : 20.0 kHz
WINDOW LENGTH      : 30 msec
WINDOW TYPE        : HANNING
FFT LENGTH         : 1024
FRAME PERIOD       : 5 msec
PREEMPHASIS        : 0.98
INPUT FILE NAME    : TMP.DAT float
OUTPUT FILE NAME   : TMP.DFT float
```

コマンドは、簡単な説明と使い方とパラメータの既定値(default value)を画面に表示する。

```
Calculate DFT running spectra.
```

これが、コマンドの簡単な説明である。

```
usage :: fft_run [options] filename
         fft_run [options] -o arguments
```

これが、コマンドの使い方である。

各コマンドは、パラメータを持ち、各パラメータには既定値が設定されている。コマンドを使うときは、パラメータを指定して使わなければならない。

コマンドには、大きく分けて２通りの使い方がある。コマンドの引数にファイル名を与える方法と、－ｏを与え、以下、パラメータをコマンドラインの引数に与える方法である。これらの方法を通じてパラメータの値を変更し、コマンドを実行する。

以下に、パラメータの例を示す。

```
SAMPLING FREQUENCY : 20.0 kHz
WINDOW LENGTH      : 30 msec
WINDOW TYPE        : HANNING
FFT LENGTH         : 1024
FRAME PERIOD       : 5 msec
PREEMPHASIS        : 0.98
INPUT FILE NAME    : TMP.DAT float
OUTPUT FILE NAME   : TMP.DFT float
```

パラメータは、以下の形式で指定する。

パラメータ名　：　値　単位　＃　コメント

たとえば、

SAMPLING FREQUENCY : 20.0 kHz

では、SAMPLING FREQUENCY がパラメータ名で、20.0 が値で、kHz が単位である。コメントは、記号（＃）から改行までであり、コマンドを実行するときは、コメントは無視される。

②パラメータをファイルに書き込む。

```
% fft_run > envfile
Calculate DFT running spectra.
usage :: fft_run [options] filename
         fft_run [options] -o arguments

Defaults are as follows.
```

パラメータをファイルに一旦取り込むにはリダイレクション(redirection)を使う。リダイレクションによって、標準出力装置(stdout)に出力されていたものがファイルに書き込まれる。ここで注意すべきは、'Defaults are as follows.'以下の行だけが標準出力装置に出力されているということである。

確認のために、catを使ってファイルの内容を表示させると次の様になる。

```
% cat envfile
SAMPLING FREQUENCY : 20.0 kHz
WINDOW LENGTH      : 30 msec
WINDOW TYPE        : HANNING
FFT LENGTH         : 1024
FRAME PERIOD       : 5 msec
PREEMPHASIS        : 0.98
INPUT FILE NAME    : TMP.DAT float
OUTPUT FILE NAME   : TMP.DFT float
```

これを、エディッタを使って編集して望みの値を設定する。

```
% fft_run envfile
..........................................................................
...............................................
!120
```

多くのコマンドは、実行すると記号'.'が、標準出力装置に表示され、最後に記号'!'と数値を表示する。数値は、記号'.'の数に等しく大きなループの回数を表わしている。

③－ｏオプションを使う。

第１引数に－ｏを指定すると、コマンドに、ファイルを使用せずに既定値を変更することができる。この使い方は、特にシェルスクリプトを使用してバッチ処理を行なうときに有効である。

```
% fft_run -o "FRAME PERIOD : 10 msec" "FFT LENGTH : 4096"
```

注意事項

・指定されないパラメータは、その既定値が使われる。

・指定するパラメータの順序は問わないが、２回以上与えると後の方が有効となる。

・全ての文字列では、大文字と小文字の区別を行なってる。

・パラメータの既定値を変更するには、～／．ｓｔｒｃを利用する。

変数名について

各コマンドの変数名は、おおよそ同じなので１ｐｃ＿ｒｕｎを例にして変数名について説明する。１ｐｃ＿ｒｕｎの変数を表１にまとめた。

表１　１ｐｃ＿ｒｕｎの変数

変数名	意　味	既定値	単位	
SAMPLING FREQUENCY	標本化周波数	20.0	kHz	①
WINDOW LENGTH	分析窓長	30.0	msec	②
WINDOW TYPE	分析窓の種類	HANNING		③
FFT LENGTH	ＤＦＴの長さ	1024		④
FRAME PERIOD	分析フレームの周期	5.0	msec	⑤
PREEMPHASIS	高域強調のパラメータ	0.98		⑥
ORDER OF LPC	線形予測係数の次数	16		⑦
INPUT FILE NAME	入力のデータファイル名	TMP.DAT	float	⑧
OUTPUT LPC FILE NAME	出力のLPCデータのファイル名	TMP.LPC	float	⑨
OUTPUT ALPHA FILE NAME	出力の線形予測係数のファイル名	TMP.ALP	float	⑩

①には、音声データの標本化周波数を指定する。
②には、分析窓長を指定する。
③には、以下のものから選択できる。

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

④は、2^nの正の整数である。ただし、標本化周波数×分析窓長よりも長くなくてはならないが、上限はない。
⑤には、分析のフレーム周期を指定する。もし、分析窓長よりも長い場合は、分析フレームは、重なりがないことになる。
⑥は、高域強調に関する係数で、０～１の値を取る。０を与えると高域強調しない。
⑧⑨⑩には、ファイル名を与えるが、単位に示されたデータ型のバイナリィファイルである。通常、音声データに関してはshortを用いるが、コマンドの単位がfloatである場合、単位の部分に望みの単位を指定すると自動的に変換してファイルを読み書きする機能がある（詳しくは、各コマンドのマニュアルを参照すること）。また、ファイル名の指定に際しては、絶対パスで指定することができるが、特殊文字（’～’，’．’，’．．’）を用いて相対パスで指定することもできる。なお、ワイルドカードを表す特殊文字（’＊’，’？’）は、使えない。

2．2　使い方（応用編）

例えば、多くのデータを一度に同じ方法で分析したり、あるいは、ある一定の条件で異なったパラメータを用いて合成音を作ったりと、定型的な作業を連続して行なうときがよくある。この様な一連の定型的な処理のことをバッチ(batch)処理と言う。

UNIXでは対話的に処理を行なうことが基本となっているが、バッチ処理を行なう場合にはシェルスクリプト(shell script)を用いるのが簡単な方法である。シェルとはUNIXの対話的なユーザーインターフェースそのものであり、ユーザーからの要求を解釈して実行するものである。

他の多くのオペレーティングシステムとは異なり、UNIXでは、シェルは、一つのコマンドであり、そのため、UNIXには多くのシェルが存在する。代表的なものとして、ｓｈ（Bourneシェルまたは単にシェルという）とｃｓｈ（Ｃシェルという）がある。シェルスクリプトとは、シェルの文法に従って記述するプログラムのことである。

このマニュアルでは、シェルにはｃｓｈを用いる。以下、例題にしたがっていくらかのｃｓｈの説明を加えるが、詳しくは正著[Anderson & Anderson, 1986]にゆずる。例えば、ファイルを編集して、シェルを実行するのには、ファイルに実行許可の指定を行なわなければならないことなどは、ここでは説明しない。

ここで述べる例は極めて簡単なものであるが、これらのものは基本的なものであり、組み合せることによって多くの複雑なことができる。よって、ここで取り上げた例を修得することを勧める。

参考文献

Anderson, G. and Anderson, P. (1986) : "THE UNIX C SHELL FIELD GUIDE",
訳　落水浩一朗、大木敦雄、パーソナルメディア㈱

[例題　1：　ＤＦＴ分析]

```
[dft.csh]
#! /bin/csh -f
set     DATA = TEST
fft_run -o\
        "SAMPLING FREQUENCY : 20.0 kHz"\
        "WINDOW LENGTH      : 30 msec"\
        "WINDOW TYPE        : HANNING"\
        "FFT LENGTH         : 1024"\
        "FRAME PERIOD       : 5 msec"\
        "PREEMPHASIS        : 0.98"\
        "INPUT FILE NAME    : $DATA.DAT short"\
        "OUTPUT FILE NAME   : $DATA.SPC float"
```

<u>ｃｓｈの起動</u>

まず、第１行目では、ｃｓｈを使用することを宣言している。記号'#'は、通常、注釈（コメント）を表わすが、ファイルの一番最初の行に'#! /bin/csh -f'と記述すると以下このファイルを、ｃｓｈの文法に従って記述でき、かつ、実行されることを意味する。ｃｓｈを使うときは必ず第１行目にこれを書かなければならない。もし、この行がファイルの第１行目にないときはｃｓｈの記法は使用できないので、文法的なエラーが生じる。

'-f'は 'csh' のオプションで、素早くｃｓｈを起動することを意味する。ｃｓｈは、通常、ユーザのホームディレクトリィにある'.cshrc'ファイルを読んだ後、起動されるが、'-f'オプションを付けておくと'.cshrc'ファイルを読まないで起動される。このことによって、'.cshrc'ファイルに記述されたエイリアス(alias)等の定義は、有効にならない。しかし、環境変数（定義済シェル変数）は、プロセス生成時に継承されるので、例えば、環境変数の１つである'path'の指定は、継承される。

<u>シェル変数を定義するには</u>

'set'を使って変数を定義するには、

```
set  変数名 = 値
```

とする。値には、数値、文字列、リストがある。リストは、記号'('と記号')'によって作られる。ここでは、

```
set     DATA = MAU10018
```

'DATA'という変数名に'MAU10018'という文字列が代入されている。変数を使用するときは変数名の前に記号'$'を付ける。'$DATA.AD'と'$DATA.DFT'は、それぞれ、'MAU10018.AD'と'MAU10018.DFT'に対応している。

<u>一行に収めるには</u>

記号'＼'は、バックスラシュである。ｃｓｈでは、バックスラシュによって便宜的に行を続けることができる。'－ｏ'オプションを使うときは、１行に全ての既定値を書かなければならないので、バックスラシュによって一つの行としなければならない。

なお、コマンドに'－ｘ'オプションを与えると、シェルスクリプトを書くのが便利なようにバックスラシュ着きでパラメータが標準出力装置(stdout)に出力される。

［例題　２：　ＬＰＣ分析］

```
[lpc.csh]
#! /bin/csh -f
foreach FILE ( $* )
        set DATA = $FILE:r
        lpc_run -o\
                "SAMPLING FREQUENCY      : 20.0 kHz"\
                "WINDOW LENGTH           : 30 msec"\
                "WINDOW TYPE             : HANNING"\
                "FFT LENGTH              : 1024"\
                "FRAME PERIOD            : 5 msec"\
                "PREEMPHASIS             : 0.98"\
                "ORDER OF LPC            : 16"\
                "INPUT FILE NAME         : $DATA.DAT short"\
                "OUTPUT LPC FILE NAME    : $DATA.LPC float"\
                "OUTPUT ALPHA FILE NAME : $DATA.ALP float"
end
```

コマンド引数の渡し方

　コマンド行からファイル名等を渡したいときがよくある。'$*'は、コマンド行のリスト全てになる。'$*'は、ｃｓｈでは、コマンド行のリストである。具体的に述べると、コマンド行で、以下の様に入力されると

```
% lpc.csh FSU10018.AD FKN10018.AD MAU10018.AD MHT10018.AD
```

'$1'は、FSU10018.AD、'$2'は、FKN10018.AD、'$3'は、MAU10018.AD、'$4'は、MHT10018.ADになる。

　なお、'$1'と'$argv[1]'ならびに'$*'と'$argv[*]'は、表記が違うが同じ意味である。'argv'という変数は、ｃｓｈの方であらかじめ定義されている変数リストである。このように、ｃｓｈの方であらかじめ定義された変数が幾つかある。

　一般的に、リストのｎ番目を参照するときは、'$変数名[n]'とし、リストの長さは、'$#変数名'とすれば良い。

　例えば、以下のようなことが可能である。

```
% set list = ( a b c d )
% echo $list
a b c d
% echo $#list
4
% echo $list[2]
b
```

繰り返し実行の仕方

　繰り返し実行させるときには、'foreach'を使うのが１つの方法である。'foreach'は、'end'までを繰り返し実行する。

　'foreach FILE ($*)'では、'FILE'は仮変数であり、リストの先頭に順次、単一化されて行く。リストが空になった時点でこのループは終了する。

［例題　３：　ローパスフィルタリング］

```
[lpf.csh]
#! /bin/csh -f
set LIST = `cat TEMPLATE`
foreach DATA ( $LIST )
        lpf1 -o\
                "SAMPLING FREQUENCY : 20.0 kHz"\
                "CUT OFF FREQUENCY  : 100.0 Hz"\
                "INPUT FILE NAME    : $DATA short"\
                "OUTPUT FILE NAME   : $DATA.LPF float"
end
```

<u>外部ファイルからの入力</u>

外部ファイルからファイル名のリストを読み込んで実行するには、

```
set LIST = `cat TEMPLATE`
```

とする。ここで、記号'`'は、バッククォートであり、バッククォートで囲まれた文字列は、ｃｓｈによって評価される。例ではファイル 'TEMPLATE' を出力する。出力された内容が、変数 'LIST' に代入される。

また、外部コマンドを実行して変数に代入するには、以下の様にすれば良い。

```
set LIST = `ls *.AD`
```

［例題　４：　ピッチ抽出］

```
[pitch.csh]
#! /bin/csh -f
if ( $#argv != 3 ) then
        echo "usage::"
        echo "pitch.csh filname min-freq[Hz] max-freq[Hz]"
        exit -1
endif
#
set DATA = $1
set MIN  = $2
set MAX  = $3
#
pitcher -o\
        "SAMPLING FREQUENCY : 20.0 kHz"\
        "WINDOW LENGTH      : 30 msec"\
        "WINDOW TYPE        : HANNING"\
        "FRAME PERIOD       : 5 msec"\
        "MINIMUM FREQUENCY  : $MIN Hz"\
        "MAXIMUM FREQUENCY  : $MAX Hz"\
        "THRESHOLD POWER    : 40 dB"\
        "POLARIZATION       : ON"\
        "INPUT FILE NAME    : $DATA short"\
        "PITCH FILE NAME    : $DATA.PIT float"\
        "POWER FILE NAME    : $DATA.POW float"
```

目的に応じて

しばしば、コマンドラインから条件を変えて分析をしたいときがある。この例では、分析対象のファイル名とピッチ抽出の最大と最小の基本周波数を指定している。

忘れるということを忘れないように

ｃｓｈやＣ言語などを使ってプログラムを作るとき、少し時間がたち日もたつと、使い方を忘れてしまったり、どんなことをしてくれるプログラムなのかを忘れてしまったりするときがよくある。引数リストの順番は、必ずといっていいほど忘れてしまう。もし要求される引数の数が違ったら、実行しないようにするよう心掛けると良い。また、引数の数が間違っていたら使い方を簡単に説明するようにプログラムするのが賢明な方法である。

［例題　５：　データの切り出し］

```
[cut.csh]
#! /bin/csh -f
echo -n "Please input a file name : "
set DATA = $<
/bin/ls -l $DATA
if ( $status ) then
        exit -1
endif
echo -n "Offset [byte] : "
set OFFSET = $<
echo -n "Length [byte] : "
set LENGTH = $<
#
subset -o\
        "SOURCE FILE NAME : $DATA"\
        "RESULT FILE NAME : $DATA.CUT"\
        "SIZE OF ARRAY    : 1 byte"\
        "OFFSET           : $OFFSET"\
        "LENGTH           : $LENGTH"
```

<u>会話的に使用するには</u>
'echo -n'は、メッセージを出力した後に改行しないことを表わす。そして、'$<'によって入力を待つ。

```
set DATA = $<
```

入力された文字列は変数 'DATA' に代入される。

<u>失敗を発見する</u>
このような方法で、ｃｓｈでも会話的に処理を行なうことができるが、会話的に処理を行なうとき注意しなければならないのは入力ミス等である。ｃｓｈでは、入力ミス等を発見するための機能が、あまり用意されてはいない。
直前のコマンド等が正常に終了したかどうかを確かめるには、変数 '$status' を参照する。

```
/bin/ls -l $DATA
if ( $status ) then
        exit -1
endif
```

'bin/ls -l $DATA'によってファイル名をリストし、かつファイルの長さを利用者に提示している。もしファイルが存在しなかったときは、'$status' には０以外の数字が入っている。UNIXの多くのコマンドは、正常終了したときは０を、異常終了したときは０以外の数字を返すようになっている。
本マニュアルで説明するコマンドも多くは、この慣例に従って、正常終了したときは０を、異常終了したときは－１を返すようになっている。

［例題　５：　バイナリィファイルの作成］

```
[atobi.csh]
#! /bin/csh -f

cat << EOF > TMP.ASC
   0.0  120.0
1000.0  100.0
EOF
#
atobi -o\
        "INTERPOLATION METHOD : LINEAR"\
        "TOTAL LENGTH         : 1000.0 msec"\
        "FRAME PERIOD         : 5.0 msec"\
        "TIME COLUMN          : 1"\
        "DATA COLUMN          : 2"\
        "INPUT FILE NAME      : TMP.ASC ascii"\
        "OUTPUT FILE NAME     : TMP.OUT float"
#
/bin/rm -f TMP.ASC
```

<u>ヒアドキュメントの利用</u>

ヒアドキュメント（埋め込み文書）とは、ｃｓｈ中に文書を埋め込む方法である。例では、

```
cat << EOF > TMP.ASC
  0.0   120.0
100.0   100.0
EOF
```

この部分が、ヒアドキュメントである。記号'<<'はインラインでの入力の切り替えを指定する。'cat << EOF'は、文字列 'EOF'がくるまでを読み込んで、それを出力する。この例では、'cat << EOF > TMP.ASC'となっているので、出力はリダイレクションによってファイル'TMP.ASC'に出される。

コマンド'atobi'は、ファイル'TMP.ASC'を読み込んで、線形補間によってバイナリィデータを作成する。ファイル'TMP.ASC'の第１列目は時刻［単位：ｍｓｅｃ］であり、第２列目は、なんらかの値（例えば基本周波数の値）である。

第３章　ユーティリティ

この章では、SpeechToolsのユーティリティを紹介する。ここで紹介するユーティリティの使い方は、第２章で説明した方法と異なるので、詳しくは、各マニュアルを参照されたい。

３．１　ａｃａｔ／ｂｃａｔ

ａｃａｔ　　　　アスキー形式からバイナリィ形式への変換
ｂｃａｔ　　　　バイナリィ形式からアスキー形式への変換

ファイルのデータ形式として大きく分けると２種類ある。

・アスキー(ascii)形式

アスキー文字で構成されテキストとして扱うことができ、比較的扱いやすい。例えば、emacsやviなどのエディッタを使って編集することができる。

・バイナリィ(binary)形式

非アスキー文字で構成されテキストとして扱うことができないので、扱いにくいが、コンパクトにデータを保存でき、また、低レベルでデータの読み書きができる。バイナリィ形式のデータには、char, int, short, long, float, doubleなどがある。

SpeechToolsでは、アスキー形式のデータとバイナリィ形式のデータ共にあり、アスキー／バイナリィ間でデータ形式の変換をａｃａｔとｂｃａｔで行なう。

３．２　ｃｖ

ｃｖ　　　　バイナリィデータ形式間の変換

short型のバイナリィデータは音声信号の入出力に用いられ、float型のバイナリィデータは音響分析等によって作成されるので、float型からshort型に変換する必要がしばしば生じる。また、ｃｖは変換を行うときにデータの最大値を指定することができる。

３．３　ｍｅｒｇｅ／ｓｅｐａｒａｔｅ

ｍｅｒｇｅ　　　　列の併合
ｓｅｐａｒａｔｅ　　　　列による分割

こでは、スプレッド・シートなどの表データ（アスキー形式）を扱うコマンドについて説明する。表データとは、以下のようなもので列と行によって特徴付けできる。

［表データの例］

a	1	200.0
b	2	300.0
c	3	100.0

この［表データの例］では、３列４行のデータで、１列目にはアルファベットが、２列目には整数が、３列目には実数が記されている。この様な表データがファイルに保存されているとき１列目を取り出したり、１列目と３列目を取り出したり、また、４列目に新しくデータを付け足したりすることがよくある。少ない行数のデータであればエディッタを用いればいいが、行数が長かったり、操作が定型的である場合には何かコマンドで実行する方が賢明である。幸いにしてUNIXにはこの種の仕事を簡単に行なうことのできるａｗｋというコマンドが用意されているが、ａｗｋは機能が多くその分使い方も繁雑である。そこで、ｍｅｒｇｅ、ｓｅｐａｒａｔｅは、列に対してのみの分割と併合をそれぞれ行なう。

3．4　ｂｃａｌｃ／ｓｍａｔｈ

ｂｃａｌｃ　　　　　　　　　　　　バイナリィデータ間の四則演算
ｓｍａｔｈ　　　　　　　　　　　　アスキーデータに対する算術演算

ｂｃａｌｃは、２つのバイナリィファイルを読み込んで四則演算（＋，－，×，÷）をほどこした後、結果をバイナリィデータとしてファイルに書込むか、または、アスキーデータとして標準出力装置(stdout)に出力する。
一方、ｓｍａｔｈでは、指数・対数演算、三角関数等の算術演算を行うことができ、標準入力装置(stdin)からアスキーデータを読み込んで、算術演算を入力したデータにほどこした後、演算結果を標準出力装置(stdout)に出力する。

第４章　ライブラリ

SpeechToolsのライブラリには、以下の４種類があり、Ｃ言語で記述されたプログラムから利用することができる。

libcommand.a	ファイルの入出力、コマンドのインタフェース等
libst.a	音響分析・合成、DFT、高次方程式の解法等
libmx.a	行列演算、連立方程式の解法等
libtlist.a	簡易なリスト処理等

通常、SpeechToolsのライブラリとヘッダファイルは、それぞれ/usr/local/st/libと、/usr/local/st/includeにインストールされており、SpeechToolsのライブラリを用いて新たにコマンドを開発する場合は、これらのディレクトリに注意しなければならない。このようなときには、ｓｔｍｋｍｆ(SpeechTools Make Makefile)を利用するとよい。これは、シェルスクリプトであり、コンパイルとリンクに必要なMakefileとmain.cを作成するものである。ｓｔｍｋｍｆによって作成されるMakefileにはライブラリやヘッダファイルのディレクトリ情報が記述されており、コンパイルやリンク等に必要なフラグが設定されている。また、main.cにはSpeechToolsのコマンドで実現されている統一化されたインターフェースを使ってコマンドを作成するために必要な変数の受け渡し等が記述されている。利用者は、これらのファイルを目的にしたがって変更することによって、独自のコマンドを作成することができる。

なお、SpeechToolsのライブラリは、UNIXの数学ライブラリであるlibmx.aを利用しているので、新たにコマンドを開発するときは、このライブラリとリンクする必要がある。

第５章　まとめ

各種の計算機上で利用できる汎用的な音声研究用ツール(SpeechTools)の利用方法について詳しく述べてきたが、SpeechToolsは、以下に示すような特徴を持っている。

・単一化されたインターフェース
従来のUNIX上の音声研究用ツールでは、分析条件等のパラメータをコマンドの引数としてUNIXのコマンドラインから与える方式を採っているために各コマンドごとにどのようなオプションがあるのか、また、どのような意味を持つのかをマニュアル等で参照しなければならなかった。さらに、各コマンドごとに使い方が微妙に異なっている場合が多く、使い勝手が悪かった。これに対して、SpeechToolsでは、ユーティリティコマンド以外の約５０個の音声分析などを行うコマンドの使い方が全て同じであり、コマンド自身が簡単な説明を行うということである。

・機能ごとに分れたコマンド
SpeechToolsは大きな一つのプログラムではなく、小さな機能ごとに個別に分けられたコマンドの集合体として設計されており、コマンドを連結することでより多くの目的に利用できるようになっている。また、データ変換などを行うユーティリティを備え、既存のUNIXコマンドとの連結を支援している。

・ファイルを介したデータの受け渡し
SpeechToolsでは、コマンドの連結に際してファイルを利用することである。UNIXにおいては、コマンドを連結する場合にパイプ機能を利用することができるが、パイプによるコマンドの連結はテキストデータを扱う場合には良いが、音声研究で利用するデータを１バイトのデータ列とするにはあまりにも複雑すぎ、また、ある種の分析手法では１つの入力データから２つ以上の分析結果を出力する場合があるのでパイプを使ってコマンドを連結するのは非常に困難である。そこで、SpeechToolsでは一部のユーティリティコマンドを除いてデータを受け渡しする方法としてファイルを用いている。

・煩雑なデータの変換の支援
音声研究においては、各種データ型のファイルを扱う必要があり、データ変換の作業は、必要不可欠である。そこで、SpeechToolsには、７個のユーティリティコマンドがあり、煩雑なデータ変換の作業を行えるようになっている。また、コマンドには、ファイルを読み込む際にデータフォーマットを自動的に変換する機能を持っているものもある。

・多数のライブラリ
SpeechToolsのライブラリには既存の音声分析／合成等に必要なアルゴリズムやファイルの入出力、データ変換等のプログラムの部品となる関数が数多く集められている。

・各種計算機上での利用が可能
現在、SpeechToolsは、以下の計算機上で動作することが確認されている。

FX/80	Alliant Computer Systems Co.
HP9000	Hewlett-Packard Company
MC6000	Concurrent Computer Co.
R6000/R3000	MIPS Computer Systems Inc.
SUN3/SUN4	Sun Microsystems Inc.
NEWS	Sony Co.
NeXT	NeXT Computer, Inc.
VAX8600	Digital Equipment Co.

今後の課題

一方、各機種上で利用するためにSpeechToolsには各機種ごとの特別なチューニングが行われていないので、各機種固有の機能であるA/D･D/Aやグラフィック等は含まれていないのと、対話的なインタフェースを持っていないのが欠点である。
現在、A/D･D/Aに関しては、MC6000(Concurrent Computer Co.)上でA/D･D/AコンバータMD8000(パベック電子開発)を、SUN3/SUN4(Sun Microsystems Inc.)上でDSPボードDSP56K(M.I.システムズ)をそれぞれ利用することができ、SpeechToolsとリンクしたコマンドを使用している。また、SpeechToolsのライブラリを一部取り入れて、SUN4上で波形表示／編集などの機能を持ったツールをDSP56KとX-Windowを利用して作成中である。これらのものは、SpeechToolsと密接には関係しておらず、それぞれ独立に動作するツールである。今後は、このようにSpeechToolsの周辺ツールとして、各種の装置や多様なウインドウシステム上でA/D･D/Aや波形等をグラフィック表示するツールを個別に増やしていく必要があるが、SpeechToolsは、どのような状況下にでも利用可能なコマンドやライブラリを多数備えているものと確信している。
筆者としては、特にNeXT上で動作するA/D･D/A、波形表示／編集、各種音響パラメータの表示／加工などの機能を持ったSpeechToolsのインタフェースを開発していきたいと考えている。

【謝辞】

SpeechTools作成に関して多くの方より御助言と御助力を頂きました。聴覚研究室に滞在していたAlain de Cheveigne氏や京都工芸繊維大学小林豊氏からは有益な御助言を頂戴しました。ここに深く感謝いたします。また、本稿を執筆する際して、草稿段階より多くの御助言を頂きました聴覚研究室の東倉洋一氏、平原達也氏（現ＮＴＴ）、山田玲子氏、小林範子氏、小原和昭氏ならびに各位に感謝いたします。

付　録.

Utility	
acat(1)	acat – convert ascii data (from 'stdin') to binary format file.
bcat(1)	bcat – convert binary format files to ascii data ('stdout').
cv(1)	cv – convert between binary data formats.
merge(1)	merge – merge ascii streams
separate(1)	separate – split an ascii stream
bcalc(1)	bcalc – calculate a basic arithmetic expression (+ , - , * /) on binary data.
smath(1)	smath – calculate arithmetic functions using the standard input/output stream.
Analysis	
peak_pick(1)	peak_pick – Peak-Picking for formants.
pitcher(1)	pitcher – Pitch extraction by auto-correlation method.
power(1)	power – Calculation of power with window normalization.
zerocrs(1)	zerocrs – Counting the zero cross points.
Axis	
dft_bark(1)	dft_bark – Convert the frequency axis from [Hz] scale to [Bark] scale.
dft_mel(1)	dft_mel – Convert the frequency axis from [Hz] scale to [Mel] scale.
DFT	
dft_cep(1)	dft_cep – Calculate DFT cepstrum smoothed envelope.
dft_forward(1)	dft_forward – Transfer time-domain data to complex data.
dft_inverse(1)	dft_inverse – Transfer complex data to time-domain data
dft_one(1)	dft_one – DFT one frame data
fft_run(1)	fft_run – Calculate the DFT running spectra.
Distortion	
cep_dist(1)	cep_dist – LPC Cepstrum Distance.
euclid_dist(1)	euclid_dist – Calculate Euclid distance.
spc_dist(1)	spc_dist – Calculate Spectrum Distortion.
Edit	
autoseg(1)	autoseg – Automatic segmentation.
subset(1)	subset – Cutting a subset of the file.
taper(1)	taper – Tapering the data.
transpose(1)	transpose – Transpose array (X <=> Y).
Experiment	
fileshuffle(1)	fileshuffle – Shuffle file lists.
make_pair(1)	make_pair – Making Pairs.
Filter	
bpfx(1)	bpfx – Band pass filter to eliminate noise components
emphasis(1)	emphasis – Self differential filtering.
iir_response(1)	iir_response – Calculate frequency response of IIR filter.
hpf1(1)	hpf1 – IIR high-pass filter.
lpf1(1)	lpf1 – IIR low-pass filter.

	syn_pole(1)	syn_pole – Single formant filter.
	syn_zero(1)	syn_zero – Single anti-formant filter.
Find		
	find_zerocrs(1)	find_zerocrs – Find the zero crossing points.
Interpolation		
	atobi(1)	atobi – Convert the ASCII table to binary data.
LPC		
	alp_fmt(1)	alp_fmt – Formant data from the alpha parameters of LPC method.
	alp_spc(1)	alp_spc – Calculate spectrum envelope from LPC parameters.
	dcep(1)	dcep – Calculation of spectrogram movement.
	ftrack(1)	ftrack – Formant Tracking by LPC method.
	lpc_cepst(1)	lpc_cepst – Calculate LPC CEPSTRUM.
	lpc_rev(1)	lpc_rev – Calculate moving-average parameters.
	lpc_run(1)	lpc_run – Calculate LPC running spectra.
	parcor(1)	parcor – Calculate PARCOR parameters.
Misc		
	autocorr(1)	autocorr – Calculate auto-correlation.
	differential(1)	differential – Differential calculation.
	integral(1)	integral – Integral calculation.
Smoothing		
	fmt_smooth(1)	fmt_smooth – Median smoothing for formant data.
	lstsq_smooth(1)	lstsq_smooth – Smoothing by least square method.
	median_smooth(1)	median_smooth – Median Smoothing.
Synthesis		
	npulse(1)	npulse – Generating a pulse/noise source.
	pulse(1)	pulse – Pulse generation
	residual(1)	residual – Calculate residual error.
	syn_alpha(1)	syn_alpha – Synthesizing with alpha parameters
	syn_cascade(1)	syn_cascade – Cascade formant synthesizer.
	syn_parcor(1)	syn_parcor – PARCOR Synthesizer.
Waveform		
	noise_wave(1)	noise_wave – Generating a band noise.
	sig_wave(1)	sig_wave – Generating a simple signal waveform.
	winfun(1)	winfun – Generating time-window functions.

NAME

SpeechTools – a powerful workbench system for the speech researcher.

DESCRIPTION

SpeechTools offers a collection of speech processing tools, both as stand-alone commands and as a library of routines. *SpeechTools* has three categories: **commands, utilities,** and **libraries. Commands** are stand-alone commands, which share the same usage (described later). **Utilities** are also stand-alone commands, which are concerned with conversion data format. **Libraries** are library function routines, which support a C-language interface.

[Commands]

All *SpeechTools* **commands** share the same usage and syntax. Basically, **commands** accept three usages:

% **command**
or
% **command** *parameter_file*
or
% **command** -o *parameters...*

The first form prints a list of the parameters that the **command** needs to run, together with their default values. This list can be redirected to a parameter file:

% **command** > *parameter_file*

After that, the *parameter_file* can be edited, and fed back to the **command** using the second form:

% **command** *parameter_file*

The **command** then runs using these parameters.

In the third form:

% **command** -o *parameters...*

parameters are specified on the command line and the **command** runs using these parameters. In any case, parameter entries obey the following format:

PARAMETER NAME : *VALUE UNIT # COMMENT*

Each parameter entry must be on a separate line. Spaces are allowed in the *PARAMETER NAME*, however, spaces are not allowed in the *VALUE* field. The *UNIT* field may be required for some parameters, in which case they are checked for validity when the **command** runs. Mismatching *UNIT* field causes a warning message. Because some **commands** support an automatic converting when the command reads/writes data from/to files. The *COMMENT* field is introduced by a '#' sign, and finishes at the end of the line. The *COMMENT* field is an optional note and is ignored when the **command** runs. Maximum line length for a parameter entry is fixed at 128 characters.

Example:

```
SAMPLING FREQUENCY     : 20 kHz
INPUT FILE NAME        : TMP.SRC short # short integers binary file
OUTPUT FILE NAME       : TMP.OUT float # floating point binary file
```

Using the third form, the same parameters specified on the command line or in a shell script:

```
command -o \
"SAMPLING FREQUENCY     : 20 kHz"\
"INPUT FILE NAME        : TMP.SRC short"\
"OUTPUT FILE NAME       : TMP.OUT float"
```

Parameters may be specified in any order. Some or all parameter entries may be missing. Missing parameters are given default values. Each command has its own default values and also, each user can set the values using a startup file in the user's home directory (**~/.strc**), which she or he must own. File formats of **~/.strc** is equal to those of *parameter_file*. If a parameter is repeated several times in the **~/.strc** file, in the parameter file or on the command line, the last specification overrides.

In addition, **commands** allow three options:

–d	enable debug mode
–s	disable warning message
–w	enable warning message

and special usages:

–i	parameters specified from *stdin* in order to fork process
–x	printing a command name and a list of the parameters in order to help writing shell scripts
–h	printing help messages

File path names relative to current or home directory are allowed ('~', '.', '..'). But wildcard specifications ('*', '?', etc.) are not supported.

Note that the input and output of *SpeechTools* commands are always to or from a file. Standard input (*stdin*) and output (*stdout*) and pipes are not allowed. The reasons for this restriction are explained below. Exceptions to this rule are the following seven utilities.

[Utilities]

acat	convert ascii data (from *stdin*) to binary format file.
bcat	convert binary format files to ascii data (*stdout*).
cv	convert between binary data formats.
bcalc	calculate a basic arithmetic expression (+, - , *, /) on binary data.
merge	merge ascii streams.
separate	split an ascii stream.
smath	arithmetic functions that operate on an ascii stream.

For a full explanation of the utilities, see the manual page for each specific utility and see also the manual page of **Utility(1)**.

[Libraries]

The *SpeechTools* routines are available in four libraries:

libcommand.a	routines for feeding parameters to programs
libst.a	routines for speech processing
libmx.a	routines for matrix handling
libtlist.a	routines for list handling

SpeechTools libraries support a C-language interface. For a full explanation of the libraries, see the manual page for each specific library function and see also the manual page of **Library(3)**.

PHILOSOPHY

The parameter mechanism of the *SpeechTools* commands is self-documenting, and is designed to give all commands a uniform usage style. The overhead may seem slightly heavy for simple commands, but it pays off quickly for commands that require more complex parameter specifications.

SpeechTools follows the convention from/into files for input and output of numerical data, to the exclusion of *stdin* and *stdout*. The reason for this restrictive convention is consistency. Many commands need to process several input channels, or produce multiple channels of data. Others require random access to all the data at once. Allowance of the standard input and output in some cases but not in others would seem arbitrary to the user, and would be difficult to document. Redirection of input and output and pipes are, therefore, not allowed: chains of commands must use temporary files. This should not cause too great a performance penalty, as modern file systems are well buffered.

INSTALLATION

The *SpeechTools* commands and utilities are normally installed in the /usr/local/st/bin directory. This directory should be in the path environment variable specified in **.login** or **.cshrc**. There are routine libraries and include files in the /usr/local/st/lib and /usr/local/st/include, respectively. When you want to make a new application using *SpeechTools* libraries, pay attention to the location of these directories. In this case, **stmkmf** (= SpeechTools Make Makefile) can help you. The **stmkmf** is a C-shell script, which makes a new directory and creates a *Makefile* and a *main.c* under the new directory. The *Makefile* involves information of routine libraries and include files for compiling and linking. The *main.c* is a template file for making a new command with *SpeechTools* libraries. Manual pages are also installed in the /usr/local/st/man directory. *SpeechTools* manual pages are written in *roff* format (with *eqn*). In order to read the manual pages on terminals (such as *tty* terminals), use **stman**, which is a C-shell script, instead of **man** command.

SpeechTools can be used on wide range of UNIX-based workstations. *SpeechTools* currently runs on the following systems:

FX/80	Alliant Computer Systems Co.
HP9000	Hewlett-Packard Company
MC6000	Concurrent Computer Co.
R6000/R3000	MIPS Computer Systems Inc.
SUN3/SUN4	Sun Microsystems Inc.
NEWS	Sony Co.
NeXT	NeXT Computer, Inc.
VAX8600	Digital Equipment Co.

It should be relatively easy to build the software on a variety of other UNIX systems.

FILES

~/.strc   user's startup file

AUTHOR

ATR Auditory and Visual Perception Research Laboratories,
2-2 Hikaridai Seika-cho Soraku-gun Kyoto 619-02 Japan
developed by Seiichi TENPAKU

NAME

acat, bcat, cv, bcalc, merge, separate, smath – utility commands of SpeechTools

DESCRIPTION

SpeechTools has seven utilities;

acat convert ascii data (from *stdin*) to binary format file.

bcat convert binary format files to ascii data (*stdout*).

cv convert between binary data formats.

bcalc calculate a basic arithmetic expression (+, - , *, /) on binary data.

merge merge ascii streams.

separate split an ascii stream.

smath arithmetic functions that operate on an ascii stream.

They are very convenient for importing into *SpeechTools* from another tools or for exporting from *SpeechTools* into another tools, because most *SpeechTools* commands read or write binary files. For instance, if you have sound data as ascii format, you can convert the data into short integer binary data by using **acat** or by using **smath** and **acat.** In *SpeechTools* sound data format must be 16 bit bipolar, when your sound data is different from that, **smath** helps to convert the data. Then the data can be fed into *SpeechTools* commands.

merge and **separate** do it easy to make ascii tables of of data from *SpeechTools* outputs. For example, **pitcher** and **ftrack** produce F0 (fundamental frequency) and formant data in single precision floating point binary data. To produce an ascii table, you first apply **bcat,** then **merge** and **separate** as needed. In this case, **bcat** helps to get ascii data from any binary data in *SpeechTools.* If you just need F1, F2 and F3, **separate** can pick up F1, F2 and F3 from all formant frequency data by using following style;

```
example% bcat -p 8 -t "%4.1f" TMP.FRQ
961.6  1070.5  2076.4  3400.5  4695.9  5978.2  7253.1  8530.4
970.5  1043.2  2044.5  3383.2  4684.0  5969.4  7246.8  8526.7
974.8  1032.1  2008.9  3361.9  4667.8  5956.2  7236.5  8520.6
978.4  1026.5  1990.7  3351.4  4659.9  5949.9  7231.6  8517.7
example% bcat -p 8 -t "%4.1f" TMP.FRQ | separate -s '1 2 3'
961.6  1070.5  2076.4
970.5  1043.2  2044.5
974.8  1032.1  2008.9
978.4  1026.5  1990.7
```

Then you can make an ascii table by using **merge.** Of course, **awk** or other UNIX commands can do these operations, but they have many functions and they are more complicated to use than **merge** and **separate.**

smath supports many arithmetic functions [e.g. sin, cos, log, ...]. **bcalc** has four basic arithmetic functions: they are add, subtract, multiply and divide. **smath** calculates the four basic arithmetic functions, too. But the most difference point between **bcalc** and **smath** is input and output. Since **smath** reads input data from *stdin* and prints out the result to *stdout,* **smath** can be used in a *pipe.* When you need to calculate dB, use following sequence:

```
example% cat data.ascii
1
2
3
example% cat data.ascii | smath log10 | smath mul 10
0.000000
3.010300
4.771210
```

On the other hand, **bcalc** reads input data from two binary files and writes the result to a file or *stdout,* so that **bcalc** can modify binary data files. If you need to square data, in a data file named 'foo', which is a single floating binary file:

```
example% bcat foo
1.000000
2.000000
3.000000
example% bcalc +m foo foo
1.000000
4.000000
9.000000
```

Anyway, there are many cases to use these seven utilities. Please see each manual page carefully. They can help solve intricate *conversion* problems.

SEE ALSO

acat(1), bcat(1), cv(1), bcalc(1), merge(1), separate(1), smath(1)

AUTHOR

Seiichi TENPAKU

NAME

libcommand.a, libst.a, libmx.a, libtlist – library routines of SpeechTools

DESCRIPTION

The *SpeechTools* routines are available in four libraries:

libcommand.a routines for feeding parameters to programs. See following manual pages: column(3) command(3) convert(3) count(3) file(3) getabspath(3) gethome(3) getpwd(3) interface(3) message(3) meter(3) signal(3) string(3)

libst.a routines for speech processing. See following manual pages: alp_cep(3) alp_par(3) average(3) axis(3) cepstrum(3) complex(3) cor_alp(3) correlation(3) critical(3) differential(3) dka(3) dot(3) fft(3) filter(3) find(3) gain(3) interpol(3) lagwin(3) lip(3) lsp(3) lstsq(3) median(3) par_alp(3) pole_alpha(3) pole_zero(3) random(3) sig_cor(3) smooth(3) stat(3) swap(3) synthe(3) taper(3) winfun(3) zplane(3)

libmx.a routines for matrix handling. See following manual pages: mxalloc(3) mxcholesky(3) mxhouse(3) mxmul(3) mxsolve(3) mxtrans(3)

libtlist.a routines for list handling. See following manual pages: tlist(3)

SpeechTools libraries support a C-language interface. When you want to make a new application using *SpeechTools* libraries, pay attention to the location of routine libraries and include files. In this case, **stmkmf** (= SpeechTools Make Makefile) can help you. The **stmkmf** is a C-shell script, which makes a new directory and creates a *Makefile* and a *main.c* under the new directory. See the *Makefile* and the *main.c*. Since *SpeechTools* libraries needs the **libmx.a**, which is a mathematical library in each UNIX system, the new application must be linked to the **libmx.a**.

LIST OF LIBRARY FUNCTIONS

Function Name	Manual Page	Library Name
BarkToFreq	axis(3)	*libst.a*
CommandInit	command(3)	*libcommand.a*
FreqToBark	axis(3)	*libst.a*
FreqToMel	axis(3)	*libst.a*
IntrfcFileRead	interface(3)	*libcommand.a*
IntrfcFileWrite	interface(3)	*libcommand.a*
IntrfcFindUnit	interface(3)	*libcommand.a*
IntrfcLineGet	interface(3)	*libcommand.a*
IntrfcPktAset	interface(3)	*libcommand.a*
IntrfcPktFind	interface(3)	*libcommand.a*
IntrfcPktGet	interface(3)	*libcommand.a*
IntrfcPktRead	interface(3)	*libcommand.a*
IntrfcPktSet	interface(3)	*libcommand.a*
IntrfcPktStr	interface(3)	*libcommand.a*
IntrfcPktWrite	interface(3)	*libcommand.a*
IntrfcStrPkt	interface(3)	*libcommand.a*
MelToFreq	axis(3)	*libst.a*
PointToBark	axis(3)	*libst.a*
PointToFreq	axis(3)	*libst.a*

ZpToFB	zplane(3)	*libst.a*
aberth_init	dka(3)	*libst.a*
alp_cep_double	alp_cep(3)	*libst.a*
alp_cep_float	alp_cep(3)	*libst.a*
alp_par_double	alp_par(3)	*libst.a*
alp_par_float	alp_par(3)	*libst.a*
average_double	average(3)	*libst.a*
average_float	average(3)	*libst.a*
average_int	average(3)	*libst.a*
average_short	average(3)	*libst.a*
binRead	file(3)	*libcommand.a*
binStore	file(3)	*libcommand.a*
cepstrum	cepstrum(3)	*libst.a*
cepstrum_envelope	cepstrum(3)	*libst.a*
columnGet	column(3)	*libcommand.a*
convertArray	convert(3)	*libcommand.a*
convertRead	convert(3)	*libcommand.a*
convertWrite	convert(3)	*libcommand.a*
cor_alp_double	cor_alp(3)	*libst.a*
cor_alp_float	cor_alp(3)	*libst.a*
correlation_double	correlation(3)	*libst.a*
correlation_float	correlation(3)	*libst.a*
countChar	count(3)	*libcommand.a*
countLine	count(3)	*libcommand.a*
countWord	count(3)	*libcommand.a*
critical_bandwidth	axis(3)	*libst.a*
critical_damp	critical(3)	*libst.a*
critical_frequency	axis(3)	*libst.a*
cx_abs	complex(3)	*libst.a*
cx_add	complex(3)	*libst.a*
cx_arg	complex(3)	*libst.a*
cx_conj	complex(3)	*libst.a*
cx_div	complex(3)	*libst.a*
cx_exp	complex(3)	*libst.a*
cx_mul	complex(3)	*libst.a*
cx_pow	complex(3)	*libst.a*
cx_print	complex(3)	*libst.a*
cx_root	complex(3)	*libst.a*

cx_sub	complex(3)	*libst.a*
cx_tocomplex	complex(3)	*libst.a*
cx_zero	complex(3)	*libst.a*
debugMsg	message(3)	*libcommand.a*
debugMsgSet	message(3)	*libcommand.a*
differential_double	differential(3)	*libst.a*
differential_float	differential(3)	*libst.a*
differential_int	differential(3)	*libst.a*
differential_short	differential(3)	*libst.a*
dka_method	dka(3)	*libst.a*
dka_solve	dka(3)	*libst.a*
dot_double	dot(3)	*libst.a*
dot_float	dot(3)	*libst.a*
dot_int	dot(3)	*libst.a*
dot_short	dot(3)	*libst.a*
errorMsg	message(3)	*libcommand.a*
errorMsgSet	message(3)	*libcommand.a*
fft	fft(3)	*libst.a*
filter_dif1	filter(3)	*libst.a*
filter_hpf1	filter(3)	*libst.a*
filter_lpf1	filter(3)	*libst.a*
find_double_absmax	find(3)	*libst.a*
find_double_absmin	find(3)	*libst.a*
find_double_max	find(3)	*libst.a*
find_double_min	find(3)	*libst.a*
find_double_peak	find(3)	*libst.a*
find_double_zero	find(3)	*libst.a*
find_float_absmax	find(3)	*libst.a*
find_float_absmin	find(3)	*libst.a*
find_float_max	find(3)	*libst.a*
find_float_min	find(3)	*libst.a*
find_float_peak	find(3)	*libst.a*
find_float_zero	find(3)	*libst.a*
find_short_absmax	find(3)	*libst.a*
find_short_absmin	find(3)	*libst.a*
find_short_max	find(3)	*libst.a*
find_short_min	find(3)	*libst.a*
find_short_peak	find(3)	*libst.a*

find_short_zero	find(3)	*libst.a*
fopenp	file(3)	*libcommand.a*
gain_control_double	gain(3)	*libst.a*
gain_control_float	gain(3)	*libst.a*
gain_control_short	gain(3)	*libst.a*
get_file_length	file(3)	*libcommand.a*
getabspath	getabspath(3)	*libcommand.a*
gethome	gethome(3)	*libcommand.a*
getpwd	getpwd(3)	*libcommand.a*
ifft	fft(3)	*libst.a*
interpol_blend	interpol(3)	*libst.a*
interpol_blend_init	interpol(3)	*libst.a*
interpol_gettype	interpol(3)	*libst.a*
interpol_lagrange	interpol(3)	*libst.a*
interpol_linear	interpol(3)	*libst.a*
interpol_spline	interpol(3)	*libst.a*
interpol_spline_init	interpol(3)	*libst.a*
lagwin_double	lagwin(3)	*libst.a*
lagwin_float	lagwin(3)	*libst.a*
lip_inverse_double	lip(3)	*libst.a*
lip_inverse_float	lip(3)	*libst.a*
lsp_calc	lsp(3)	*libst.a*
lstsq_calc_double	lstsq(3)	*libst.a*
lstsq_calc_float	lstsq(3)	*libst.a*
lstsq_double	lstsq(3)	*libst.a*
lstsq_float	lstsq(3)	*libst.a*
median_float	median(3)	*libst.a*
meterEnd	meter(3)	*libcommand.a*
meterInc	meter(3)	*libcommand.a*
meterSet	meter(3)	*libcommand.a*
meterSwitch	meter(3)	*libcommand.a*
mx1alloc	mxalloc(3)	*libmx.a*
mx1free	mxalloc(3)	*libmx.a*
mx2alloc	mxalloc(3)	*libmx.a*
mx2free	mxalloc(3)	*libmx.a*
mx3alloc	mxalloc(3)	*libmx.a*
mx3free	mxalloc(3)	*libmx.a*
mx_cholesky	mxcholesky(3)	*libmx.a*

mx_householder_solve	mxhouse(3)	*libmx.a*
mx_ldua	mxsolve(3)	*libmx.a*
mx_mx_mul	mxmul(3)	*libmx.a*
mx_solve	mxsolve(3)	*libmx.a*
mx_trans_mul	mxmul(3)	*libmx.a*
mx_transpose	mxtrans(3)	*libmx.a*
mx_vc_mul	mxmul(3)	*libmx.a*
openRead	file(3)	*libcommand.a*
openWrite	file(3)	*libcommand.a*
par_alp_double	par_alp(3)	*libst.a*
par_alp_float	par_alp(3)	*libst.a*
pole_alpha	pole_alpha(3)	*libst.a*
pole_zero	pole_zero(3)	*libst.a*
pole_zero_fast	pole_zero(3)	*libst.a*
random_num	random(3)	*libst.a*
random_seed	random(3)	*libst.a*
random_var	random(3)	*libst.a*
set_signal_handler	signal(3)	*libcommand.a*
sig_cor_double	sig_cor(3)	*libst.a*
sig_cor_float	sig_cor(3)	*libst.a*
smooth_moving_ave	smooth(3)	*libst.a*
stat_arithmetic_mean	stat(3)	*libst.a*
stat_correlation	stat(3)	*libst.a*
stat_variance	stat(3)	*libst.a*
string_compare	string(3)	*libcommand.a*
string_partition	string(3)	*libcommand.a*
strrev	string(3)	*libcommand.a*
strsave	string(3)	*libcommand.a*
swap_char	swap(3)	*libst.a*
swap_double	swap(3)	*libst.a*
swap_float	swap(3)	*libst.a*
swap_int	swap(3)	*libst.a*
swap_long	swap(3)	*libst.a*
swap_short	swap(3)	*libst.a*
syn_direct	synthe(3)	*libst.a*
syn_pole	synthe(3)	*libst.a*
syn_twomul	synthe(3)	*libst.a*
syn_zero	synthe(3)	*libst.a*

taper_gettype	taper(3)	*libst.a*
taper_linear	taper(3)	*libst.a*
taper_second	taper(3)	*libst.a*
taper_sinc	taper(3)	*libst.a*
taper_sine	taper(3)	*libst.a*
tlist_getword	tlist(3)	*libtlist.a*
tlist_next	tlist(3)	*libtlist.a*
tlist_parent	tlist(3)	*libtlist.a*
tlist_print	tlist(3)	*libtlist.a*
tlist_read	tlist(3)	*libtlist.a*
tlist_search	tlist(3)	*libtlist.a*
warningMsg	message(3)	*libcommand.a*
warningMsgSet	message(3)	*libcommand.a*
winfun_generate_double	winfun(3)	*libst.a*
winfun_generate_float	winfun(3)	*libst.a*
winfun_gettype	winfun(3)	*libst.a*
winfun_set_float_double	winfun(3)	*libst.a*
winfun_set_float_float	winfun(3)	*libst.a*
winfun_set_func	winfun(3)	*libst.a*
winfun_set_short_double	winfun(3)	*libst.a*
winfun_set_short_float	winfun(3)	*libst.a*