TR－A－0137

SpeechTools Manual Pages

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

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

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

SYNOPSIS

acat [-s | -i | -d | -f] *filename*
acat -help

DESCRIPTION

acat converts ascii data from the standard input to binary format file. Data format can be one among :

-s	short
-i	integer
-d	double
-f	float (default)

If no option is present, the '-f' option is assumed. Out-of-range data will cause an error.
A comment line starts at the '#' sign and ends at the next *NEWLINE*. The comment line is ignored when the **acat** runs.

SEE ALSO

Utility(1), bcat(1)

EXAMPLE

```
acat a.float < a.ascii
acat -s a.short < a.ascii
```

AUTHOR

Seiichi TENPAKU

NAME

alp_fmt – Formant data from the alpha parameters of LPC method.

SYNOPSIS

alp_fmt *filename*

alp_fmt –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **alp_fmt**.
The *parameters* of **alp_fmt** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
ORDER OF LPC	: 16
OUTPUT FORMANT NUMBER	: 8
INPUT ALPHA FILE NAME	: TMP.ALP float
FREQUENCY FILE NAME	: TMP.FRQ float
BANDWIDTH FILE NAME	: TMP.BND float

When **alp_fmt** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
: Set the sampling frequency of input data.

ORDER OF LPC
: Set the order of LPC (Linear Prediction Coefficients).

OUTPUT FORMANT NUMBER
: Set the number for output of formant data. The value must be less than a half of "ORDER OF LPC".

INPUT ALPHA FILE NAME
: Set the file name of input data. The format of input data must be single-floating binary. There are no restrictions as to the file length.

FREQUENCY FILE NAME
: Set the file name for the output of formant frequency data. The size of one frame is equal to "OUTPUT FORMANT NUMBER". The format of output data is single-floating binary.

BANDWIDTH FILE NAME
: Set the file name for the output of formant bandwidth data. The size of one frame is equal to "OUTPUT FORMANT NUMBER". The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), lpc_cepst(1), lpc_run(1), parcor(1)

AUTHOR

Seiichi TENPAKU

NAME

alp_spc – Calculate spectrum envelope from LPC parameters.

SYNOPSIS

alp_spc *filename*

alp_spc -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **alp_spc**.

The *parameters* of **alp_spc** are listed below;

ORDER OF LPC	: 16
FFT LENGTH	: 1024
INPUT FILE NAME	: TMP.ALP float
OUTPUT FILE NAME	: TMP.SPC float

When **alp_spc** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

ORDER OF LPC

Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

FFT LENGTH

Set the number of DFT points. The value must be 2^n, There are no restrictions as to the value.

INPUT FILE NAME

Set the file name of input data. Input data format is single precision floating binary. There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output of LPC running spectra, which is log power spectra data in linear frequency scale. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. Output data format is single precision floating binary.

SEE ALSO

SpeechTools(1), lpc_run(1)

AUTHOR

Seiichi TENPAKU

NAME

atobi – Convert the ASCII table to binary data.

SYNOPSIS

atobi *filename*
atobi –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **atobi.**
The *parameters* of **atobi** are listed below;

INTERPOLATION METHOD	: LINEAR
TOTAL LENGTH	: 1000.0 msec
FRAME PERIOD	: 5.0 msec
TIME COLUMN	: 1
DATA COLUMN	: 2
INPUT FILE NAME	: TMP.TBL ascii
OUTPUT FILE NAME	: TMP.OUT float

When **atobi** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

INTERPOLATION METHOD

Set the type of interpolation function. The type of interpolation function can be chosen from the following table:

LINEAR
LAGRANGE
BLEND
SPLINE

TOTAL LENGTH

Set the total duration.

FRAME PERIOD

Set the duration of frame period.

TIME COLUMN

Set column number of the index time. The column number starts at 1.

DATA COLUMN

Set column number of the data on each time. The column number starts at 1.

INPUT FILE NAME

Set the file name of the input ascii table. One example of the ascii table is:

100	10
200	20
300	30
400	40
500	50

The left column means the index times. The right column mean the values on each time.

OUTPUT FILE NAME

Set the file name for the output of results. The format of output data is single-floating binary.

NOTE

A comment line starts at the '#' sign and ends at the next *NEWLINE*. The comment line is ignored when the **atobi** runs.

EXAMPLE

example% atobi
Convert the ASCII table to binary data.

```
usage :: atobi filename
       atobi -o arguments

Defaults are as follows.
INTERPOLATION METHOD     : LINEAR
TOTAL LENGTH             : 1000.0 msec
FRAME PERIOD             : 5.0 msec
TIME COLUMN              : 1
DATA COLUMN              : 2
INPUT FILE NAME          : TMP.TBL ascii
OUTPUT FILE NAME         : TMP.OUT float
example% cat TMP.TBL
100    10
200    20
300    30
400    40
500    50
example% atobi -o "TOTAL LENGTH : 500.0 msec" "FRAME PERIOD : 50 msec"
!10
example% bcat TMP.OUT
10.000000
10.000000
10.000000
15.000000
20.000000
25.000000
30.000000
35.000000
40.000000
45.000000
```

SEE ALSO

SpeechTools(1), bcat(1)

AUTHOR

Seiichi TENPAKU

NAME

autocorr – Calculate auto-correlation.

SYNOPSIS

autocorr *filename*
autocorr -o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **autocorr**.
The *parameters* of **autocorr** are listed below;

INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When **autocorr** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

autoseg – Automatic segmentation.

SYNOPSIS

autoseg *filename*

autoseg –o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **autoseg**.
The *parameters* of **autoseg** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
BLOCK LENGTH	: 15 msec
DATA LEVEL	: 40.0 dB
SKIP LEVEL	: 36.0 dB
MINIMUM DATA BLOCK	: 6
MAXIMUM SKIP BLOCK	: 5
MARGIN DURATION	: 100 msec
INPUT FILE NAME	: TMP.DAT float
BASE NAME OF OUTPUT FILE	: TMP.OUT float
OUTPUT LOG FILE	: TMP.LOG ascii

When **autoseg** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

BLOCK LENGTH

Set the duration, which is one block of segment.

DATA LEVEL

Set the level, which is a threshold of data segment.

SKIP LEVEL

Set the level, which is a threshold of skip segment.

MINIMUM DATA BLOCK

Set the number, which is a minimum block of data segment. When the power is over the "DATA LEVEL" at least the "MINIMUM DATA BLOCK", utterance assume to be starting.

MAXIMUM SKIP BLOCK

Set the number, which is a maximum block of skip segment. When the power is under the "SKIP LEVEL" at least the "MAXIMUM SKIP BLOCK", utterance assume to be ending.

MARGIN DURATION

Set the duration, which is a margin of utterance.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

BASE NAME OF OUTPUT FILE

Set the base name for output files. The names of output files are generated with the base name and a suffix number which starts at 000005 and increments by 5. Output data format can be one (only) among :

short	short integer binary

int	integer binary
double	double precision floating binary
float	single precision floating binary

OUTPUT LOG FILE

Set the file name for traced of results. The format of this file is following:
"FILE NAME" "START" "END" "LENGTH" "START SEGMENT" "END SEGMENT"
The value of "START" and "END" is point number based on the input data. The value of "LENGTH" is length of the output file named by "FILE NAME". The value of "START SEGMENT" and "END SEGMENT" is point number based on the input data. The value of "START" is calculated by "START SEGMENT" - "MARGIN DURATION". The value of "END" is calculated by "END SEGMENT" + "MARGIN DURATION".

NOTE

autoseg divides long-utterance data into short-utterance data by using power. This method is very simple. Hopefully, **autoseg** works well. But the results must be verified. When you need to cut the data by manual, **subset** might be an useful command.

EXAMPLE

```
example% autoseg -o "INPUT FILE NAME : FKN short" "BASE NAME OF OUTPUT FILE : FKN short"
WARNING:Unit of 'INPUT FILE NAME' might be 'float'.
WARNING:Unit of 'BASE NAME OF OUTPUT FILE' might be 'float'.
WARNING:FKN is converted to read [short -> float], length = 1000000
WARNING:FKN.000005 is converted to write [float -> short], length = 26160
WARNING:FKN.000010 is converted to write [float -> short], length = 49200
WARNING:FKN.000015 is converted to write [float -> short], length = 89520
WARNING:FKN.000020 is converted to write [float -> short], length = 43440
WARNING:FKN.000025 is converted to write [float -> short], length = 32640
WARNING:FKN.000030 is converted to write [float -> short], length = 43440
WARNING:FKN.000035 is converted to write [float -> short], length = 35520
WARNING:FKN.000040 is converted to write [float -> short], length = 51360
WARNING:FKN.000045 is converted to write [float -> short], length = 61440
example% cat TMP.LOG
FKN.000005     75840   102000   26160   80640   97200
FKN.000010    152880   202080   49200  157680  197280
FKN.000015    243600   333120   89520  248400  328320
FKN.000020    332160   375600   43440  336960  370800
FKN.000025    417840   450480   32640  422640  445680
FKN.000030    502080   545520   43440  506880  540720
FKN.000035    561120   596640   35520  565920  591840
FKN.000040    597120   648480   51360  601920  643680
FKN.000045    675600   737040   61440  680400  732240
```

SEE ALSO

SpeechTools(1), subset(1)

AUTHOR

Seiichi TENPAKU

NAME

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

SYNOPSIS

bcalc [*operator*] [*type*] *input_file1 input_file2* [*output_file*]
bcalc -help

DESCRIPTION

bcalc calculates a basic arithmetic expression. *operator* can be one (only) among :

+a add
+s subtract
+m multiply
+d divide

If *operator* is not specified, add is assumed. *type* is a data type of input and output file. *type* can be one (only) among :

-s short
-i integer
-f float
-d double

If *type* is not specified, float is assumed. If *output_file* is not specified, result of calculation is printed out to **stdout** as ascii format data. **bcalc** doesn't care about the range of data, so that out-of-range data causes an error.

SEE ALSO

Utility(1)

EXAMPLE

```
bcalc +a data1 data2
bcalc +s data1 data2 output
bcalc +m -d data1 data2 > output
```

AUTHOR

Seiichi TENPAKU

NAME

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

SYNOPSIS

bcat [-p *n*] [-t *format*] [-s | -i | -d | -f] *files* ...
bcat -help

DESCRIPTION

bcat converts binary format files to ascii data on the standard output. Data format can be one (only) among :

- -s short
- -i integer
- -d double
- -f float (default)

If no option is present, the '-f' option is assumed. With the '-t' option, you can use any *format* supported by the C library. The '-t' option does not replace or override the data format option. There is no error checking. The '-p n' option splits data into multiple columns.

SEE ALSO

Utility(1), acat(1)

EXAMPLE

```
bcat a.float b.float c.float > data.ascii
bcat -d *.double
bcat -t '%4.1f' -f *.float
bcat -p 3 -t '%4.1f' -f *.float
```

AUTHOR

Seiichi TENPAKU

NAME

bpfx – Band pass filter to eliminate noise components

SYNOPSIS

bpfx *filename*
bpfx -o *parameter* ...

USAGE

filename contains *parameter*s, which are concerned with **bpfx.**
The *parameters* of **bpfx** are listed below;

INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When **bpfx** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

cep_dist – LPC Cepstrum distance.

SYNOPSIS

cep_dist *filename*
cep_dist –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **cep_dist**.
The *parameters* of **cep_dist** are listed below;

ORDER OF CEPSTRUM	: 16
ORIGINAL FILE NAME	: TMP.ORG float
REFERENCE FILE NAME	: TMP.REF float
DIFFERENCE FILE NAME	: TMP.DIS float
INFORMATION FILE NAME	: TMP.IFO ascii

When **cep_dist** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

ORDER OF CEPSTRUM
Set the order of cepstrum coefficients.

ORIGINAL FILE NAME
Set the file name of input data. The format of input data must be single-floating binary. There are no restrictions as to the file length.

REFERENCE FILE NAME
Set the file name of input data. The format of input data must be single-floating binary. The file length of "REFERENCE FILE NAME" must be equal to the file length of "ORIGINAL FILE NAME".

DIFFERENCE FILE NAME
Set the file name for the output of differences in each frame. The format of output data is single-floating binary.

INFORMATION FILE NAME
Set the file name for the output of results.

SEE ALSO

SpeechTools(1), lpc_cepst(1), dtw_dist(1), euclid_dist(1), spc_dist(1)

AUTHOR

Seiichi TENPAKU

NAME

cv – convert between binary data formats.

SYNOPSIS

cv [-n *max*] [*input_type*] *input_file* [*output_type*] *output_file*
cv -help

DESCRIPTION

cv converts between binary data formats. if **-n** option is present, input data is normalized at max value before converting. *input_type* is a data type of input file and *output_type* is a data type of output file. *input_type* and *output_type* can be one (only) among :

-s	short
-i	integer
-f	float
-d	double

If *input_type* is not specified, float is assumed. And if *output_type* is not specified, short is assumed. cv doesn't care about the range of data, so that out-of-range data causes an error.

SEE ALSO

Utility(1)

EXAMPLE

```
cv data.float data.short
cv -d data.double data.short
cv data.float -f data.float
cv -n 32767 data.float data.short
```

AUTHOR

Seiichi TENPAKU

NAME

dcep – Calculation of spectrogram movement.

SYNOPSIS

dcep *filename*

dcep –o *parameter* ...

DESCRIPTION

dcep calculates the delta cepstrum.

USAGE

filename contains *parameter*s, which are concerned with **dcep**.

The *parameter*s of **dcep** are listed below;

AVERAGE PERIOD	: 5
SIZE OF ARRAY	: 17
INPUT FILE NAME	: TMP.ALP float
OUTPUT FILE NAME	: TMP.DCP float

When **dcep** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

AVERAGE PERIOD

Set the point number for averaging.

SIZE OF ARRAY

Set the array size of one frame data.

[1] If the input data is made using LPC analysis methods, the value of "SIZE OF ARRAY" must be set to "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is used 16, the value of "SIZE OF ARRAY" must be set to 17.

[2] If the input data is a kind of spectra data, the value of "SIZE OF ARRAY" must be set to a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is used 1024, the value of "SIZE OF ARRAY" must be set to 512.

INPUT FILE NAME

Set the file name of input data. The format of input data must be single-floating binary. There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output of results. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dft_cep(1), dft_forward(1), lpc_cepst(1), fft_run(1), lpc_run(1)

AUTHOR

Seiichi TENPAKU

NAME

dft_bark – Convert the frequency axis from [Hz] scale to [Bark] scale.

SYNOPSIS

dft_bark *filename*

dft_bark -o *parameter ...*

DESCRIPTION

dft_bark converts the frequency axis from [Hz] scale to [Bark] scale.

dft_mel converts the frequency axis from [Hz] scale to [Mel] scale.

USAGE

filename contains *parameters*, which are concerned with these commands.

The *parameters* are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
INPUT ARRAY SIZE	: 512
OUTPUT ARRAY SIZE	: 64
INPUT FILE NAME	: TMP.DFT float
OUTPUT FILE NAME	: TMP.OUT float

When the command name is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency.

INPUT ARRAY SIZE

Set the array size of one frame of input data.

OUTPUT ARRAY SIZE

Set the array size of one frame of output data.

INPUT FILE NAME

Set the file name of input data. The input file contains log power spectra data in linear frequency scale. For example, these files are made by using **fft_run** , **lpc_run** and so on. The format of input data must be single-floating binary. There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name of output data. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dft_mel(1), dft_cep(1), dft_forward(1), fft_run(1), lpc_run(1)

AUTHOR

Seiichi TENPAKU

NAME

dft_cep – Calculate DFT cepstrum smoothed envelope.

SYNOPSIS

dft_cep *filename*

dft_cep -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **dft_cep**.

The *parameters* of **dft_cep** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FFT LENGTH	: 1024
FRAME PERIOD	: 5 msec
QUEFRENCY LENGTH	: 2.0 msec
INPUT FILE NAME	: TMP.DAT float
OUTPUT CEPSTRUM FILE NAME	: TMP.CEP float
OUTPUT SPECTRUM FILE NAME	: TMP.SPE float

When **dft_cep** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

WINDOW LENGTH

Set the duration of windowing function.

WINDOW TYPE

Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FFT LENGTH

Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

FRAME PERIOD

Set the duration of frame period.

QUEFRENCY LENGTH

Set the quefrency duration.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT CEPSTRUM FILE NAME

Set the file name for the output of cepstrum data. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

OUTPUT SPECTRUM FILE NAME

Set the file name for the output of spectrum data, which is log power spectra data in linear frequency scale. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), lpc_cepst(1), lpc_run(1)

AUTHOR

Seiichi TENPAKU

NAME

dft_forward – Transfer time-domain data to complex data.

SYNOPSIS

dft_forward *filename*
dft_forward -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **dft_forward**.
The *parameters* of **dft_forward** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FFT LENGTH	: 1024
FRAME PERIOD	: 5 msec
INPUT FILE NAME	: TMP.DAT float
SPECTROGRAM FILE NAME	: TMP.SPC float
REAL PART FILE NAME	: TMP.REAL float
IMAGINARY PART FILE NAME	: TMP.IMAG float
PHASE FILE NAME	: TMP.PHSE float

When **dft_forward** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

WINDOW LENGTH

Set the duration of windowing function.

WINDOW TYPE

Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FFT LENGTH

Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

FRAME PERIOD

Set the duration of frame period.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

SPECTROGRAM FILE NAME
: Set the file name for the output of DFT running spectra.

REAL PART FILE NAME
: Set the file name for the output of real-part data.

IMAGINARY PART FILE NAME
: Set the file name for the output of imaginary-part data.

PHASE FILE NAME
: Set the file name for the output of phase data.
The size of one frame for the output data is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dft_inverse(1), fft_run(1)

AUTHOR

Seiichi TENPAKU

NAME

dft_inverse – Transfer complex data to time-domain data

SYNOPSIS

dft_inverse *filename*
dft_inverse -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **dft_inverse**.
The *parameters* of **dft_inverse** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FFT LENGTH	: 1024
FRAME PERIOD	: 5 msec
REAL PART FILE NAME	: TMP.REAL float
IMAGINARY PART FILE NAME	: TMP.IMAG float
OUTPUT FILE NAME	: TMP.OUT float

When **dft_inverse** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FFT LENGTH
Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

FRAME PERIOD
Set the duration of frame period.

REAL PART FILE NAME
Set the file name for the input of real-part data.

IMAGINARY PART FILE NAME
Set the file name for the input of imaginary-part data.

OUTPUT FILE NAME
Set the file name for the output of results. The format of output data might be single-floating binary.

SEE ALSO

SpeechTools(1), dft_forward(1)

AUTHOR

Seiichi TENPAKU

NAME

dft_mel – Convert the frequency axis from [Hz] scale to [Mel] scale.

SYNOPSIS

dft_mel *filename*
dft_mel -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with these commands.
The *parameters* are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
INPUT ARRAY SIZE	: 512
OUTPUT ARRAY SIZE	: 64
INPUT FILE NAME	: TMP.DFT float
OUTPUT FILE NAME	: TMP.OUT float

When the command name is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency.

INPUT ARRAY SIZE
Set the array size of one frame of input data.

OUTPUT ARRAY SIZE
Set the array size of one frame of output data.

INPUT FILE NAME
Set the file name of input data. The input file contains log power spectra data in linear frequency scale. For example, these files are made by using **fft_run** , **lpc_run** and so on. The format of input data must be single-floating binary. There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name of output data. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dft_bark(1), dft_cep(1), dft_forward(1), fft_run(1), lpc_run(1)

AUTHOR

Seiichi TENPAKU

NAME

dft_one – DFT one frame data

SYNOPSIS

dft_one *filename*

dft_one -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **dft_one.**

The *parameters* of **dft_one** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW TYPE	: HANNING
FFT LENGTH	: 1024
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.DFT float
PHASE FILE NAME	: TMP.PHA float

When **dft_one** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

WINDOW TYPE

Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FFT LENGTH

Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output data, which is log power spectra data in linear frequency scale. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

PHASE FILE NAME

Set the file name for the phase output. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1) dft_forward(1), fft_run(1)

AUTHOR

Seiichi TENPAKU

NAME

differential – Differential calculation.

SYNOPSIS

differential *filename*
differential -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **differential**.
The *parameters* of **differential** are listed below;

ORDER	: 1
MULTIPLIER	: 0.98
INPUT FILE NAME	: TMP.ORG float
OUTPUT FILE NAME	: TMP.OUT float

When **differential** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

ORDER

Set the order of differential.

MULTIPLIER

Set the multiplier factor. The range of this value is 0.0 to 1.0.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), integral(1)

AUTHOR

Seiichi TENPAKU

NAME

emphasis – Self differential filtering.

SYNOPSIS

emphasis *filename*
emphasis –o *parameter* ...

USAGE

filename contains *parameter*s, which are concerned with **emphasis**.
The *parameter*s of **emphasis** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
PREEMPHASIS	: 0.98
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When **emphasis** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

PREEMPHASIS
Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), hpf1(1), lpf1(1), syn_pole(1), syn_zero(1)

AUTHOR

Seiichi TENPAKU

NAME

euclid_dist – Calculate Euclid distance.

SYNOPSIS

euclid_dist *filename*
euclid_dist –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **euclid_dist**.
The *parameters* of **euclid_dist** are listed below;

SIZE OF ARRAY	: 64
ORIGINAL FILE NAME	: TMP.ORG float
REFERENCE FILE NAME	: TMP.REF float
DIFFERENCE FILE NAME	: TMP.DIS float
INFORMATION FILE NAME	: TMP.IFO ascii

When **euclid_dist** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SIZE OF ARRAY
Set the array size of one frame data.

ORIGINAL FILE NAME
Set the file name of input original data. The format of input data must be single-floating binary.

REFERENCE FILE NAME
Set the file name of input reference data. The format of input data must be single-floating binary.

DIFFERENCE FILE NAME
Set the file name of output differences in each frame. The format of input data is single-floating binary.

INFORMATION FILE NAME
Set the file name for output of results.

SEE ALSO

SpeechTools(1), cep_dist(1), spc_dist(1)

AUTHOR

Seiichi TENPAKU

NAME

fft_run – Calculate the DFT running spectra.

SYNOPSIS

fft_run *filename*
fft_run -o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **fft_run**.
The *parameters* of **fft_run** are listed below;

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

When **fft_run** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FFT LENGTH
Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

FRAME PERIOD
Set the duration of frame period.

PREEMPHASIS
Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output of DFT running spectra, which is log power spectra data in linear frequency scale. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dft_forward(1), dft_one(1)

AUTHOR

Seiichi TENPAKU

NAME

fileshuffle – Shuffle file lists.

SYNOPSIS

fileshuffle *filename*
fileshuffle –o *parameter* ...

DESCRIPTION

fileshuffle randomly arranges lists.

USAGE

filename contains *parameters*, which are concerned with **fileshuffle**.
The *parameters* of **fileshuffle** are listed below;

```
SEED OF RANDOMIZATION    : 17
NUMBER OF REPEAT         : 1
INPUT FILE NAME          : TMP.DAT ascii
OUTPUT FILE NAME         : TMP.OUT ascii
TEMPLATE FILE NAME       : TMP.LST ascii
```

When **fileshuffle** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SEED OF RANDOMIZATION
Set a random seed.

NUMBER OF REPEAT
Set a repeat number.

INPUT FILE NAME
Set the file name of input lists. There are no restrictions as to input lists.

OUTPUT FILE NAME
Set the file name for the output of results.

TEMPLATE FILE NAME
Set the file name for the output of template.

EXAMPLE

```
example% fileshuffle
Shuffle file lists.
usage :: fileshuffle filename
        fileshuffle -o arguments

Defaults are as follows.
SEED OF RANDOMIZATION    : 17
NUMBER OF REPEAT         : 1
INPUT FILE NAME          : TMP.DAT ascii
OUTPUT FILE NAME         : TMP.OUT ascii
TEMPLATE FILE NAME       : TMP.LST ascii
example% cat TMP.DAT
a
b
c
d
e
f
example% fileshuffle -o "SEED OF RANDOMIZATION   : 100"
example% cat TMP.OUT
b
```

a
c
f
d
e
example% cat TMP.LST
2
1
3
6
4
5

SEE ALSO

SpeechTools(1), make_pair(1)

AUTHOR

Seiichi TENPAKU

NAME

find_zerocrs – Find the zero crossing points.

SYNOPSIS

find_zerocrs *filename*
find_zerocrs –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **find_zerocrs**.
The *parameters* of **find_zerocrs** are listed below;

INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT short

When **find_zerocrs** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output of results. The format of output data is short integer binary.

SEE ALSO

SpeechTools(1), zerocrs(1)

AUTHOR

Seiichi TENPAKU

NAME

fmt_smooth – Median smoothing for formant data.

SYNOPSIS

fmt_smooth *filename*
fmt_smooth -o *parameter ...*

USAGE

filename contains *parameter*s, which are concerned with **fmt_smooth**.
The *parameter*s of **fmt_smooth** are listed below;

MEDIAN BLOCK NUMBER : 5
NUMBER OF FORMANT : 5
INPUT FREQUENCY FILE NAME : INP.FRQ float
INPUT BANDWIDTH FILE NAME : INP.BND float
OUTPUT FREQUENCY FILE NAME: TMP.FRQ float
OUTPUT BANDWIDTH FILE NAME: TMP.BND float

When **fmt_smooth** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

MEDIAN BLOCK NUMBER
Set the block number.

NUMBER OF FORMANT
Set the number of formant.

INPUT FREQUENCY FILE NAME
Set the file name for input formant frequency data. The format data is single-floating binary.

INPUT BANDWIDTH FILE NAME
Set the file name for input formant bandwidth data. The format data is single-floating binary.

OUTPUT FREQUENCY FILE NAME
Set the file name for output formant frequency data. The format data is single-floating binary.

OUTPUT BANDWIDTH FILE NAME
Set the file name for output formant bandwidth data. The format data is single-floating binary.

SEE ALSO

SpeechTools(1), ftrack(1), peak_pick(1)

AUTHOR

Seiichi TENPAKU

NAME

ftrack – Formant Tracking by LPC method.

SYNOPSIS

ftrack *filename*
ftrack –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **ftrack**.
The *parameters* of **ftrack** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FRAME PERIOD	: 5 msec
PREEMPHASIS	: 0.98
ORDER OF LPC	: 16
INPUT FILE NAME	: TMP.DAT float
FREQUENCY FILE NAME	: TMP.FRQ float
BANDWIDTH FILE NAME	: TMP.BND float
ALPHA FILE NAME	: TMP.ALP float

When **ftrack** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FRAME PERIOD
Set the duration of frame period.

PREEMPHASIS
Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

ORDER OF LPC
Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

FREQUENCY FILE NAME

Set the file name for the output of formant frequency data. The size of one frame is a half of "ORDER OF LPC". For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 8. The format of output data is single-floating binary.

BANDWIDTH FILE NAME

Set the file name for the output of formant bandwidth data. The size of one frame is a half of "ORDER OF LPC". For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 8. The format of output data is single-floating binary.

ALPHA FILE NAME

Set the file name for the output of alpha-parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 17. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), peak_pick(1), fmt_smooth(1)

AUTHOR

Seiichi TENPAKU

NAME

hpf1 – IIR high-pass filter.

SYNOPSIS

hpf1 *filename*
hpf1 –o *parameter* ...

USAGE

filename contains *parameter*s, which are concerned with **hpf1.**
The *parameter*s are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
CUT OFF FREQUENCY	: 100.0 Hz
INPUT FILE NAME	: TMP.DAT short
OUTPUT FILE NAME	: TMP.OUT short

When the command name is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

CUT OFF FREQUENCY

Set the cut off frequency. The upper limit of the cut off frequency is a quarter of the sampling frequency. For example, when the sampling frequency 20 [kHz], the upper limit of the cut off frequency is 5000 [Hz].

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), lpf1(1), differ(1), syn_pole(1), syn_zero(1)

AUTHOR

Seiichi TENPAKU

NAME

iir_response – Calculate frequency response of IIR filter.

SYNOPSIS

iir_response *filename*

iir_response -o *parameter ...*

USAGE

filename contains *parameter*s, which are concerned with **iir_response**.

The *parameter*s of **iir_response** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
FFT LENGTH	: 1024
AR FILE NAME	: TMP.AR float
MA FILE NAME	: TMP.MA float
OUTPUT FILE NAME	: TMP.OUT float

When **iir_response** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

FFT LENGTH

Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

AR FILE NAME

Set the file name of AR part filter coefficients. Input data format is single precision floating binary.

MA FILE NAME

Set the file name of MA part filter coefficients. Input data format is single precision floating binary.

OUTPUT FILE NAME

Set the file name for the output of frequency response, which is log power spectra data in linear frequency scale. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

integral – Integral calculation.

SYNOPSIS

integral *filename*
integral -o *parameter ...*

USAGE

filename contains *parameter*s, which are concerned with **integral**.
The *parameter*s of **integral** are listed below;

ORDER	: 1
MULTIPLIER	: 0.98
INPUT FILE NAME	: TMP.ORG float
OUTPUT FILE NAME	: TMP.OUT float

When **integral** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

ORDER

Set the order of integral.

MULTIPLIER

Set the multiplier factor. The range of this value is 0.0 to 1.0.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), differential(1)

AUTHOR

Seiichi TENPAKU

NAME

lpc_cepst – Calculate LPC CEPSTRUM.

SYNOPSIS

lpc_cepst *filename*
lpc_cepst –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **lpc_cepst**.
The *parameters* of **lpc_cepst** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FRAME PERIOD	: 5 msec
PREEMPHASIS	: 0.98
ORDER OF LPC	: 16
ORDER OF CEPSTRUM	: 16
INPUT FILE NAME	: TMP.DAT float
OUTPUT CEPSTRUM FILE NAME	: TMP.CEP float
OUTPUT ALPHA FILE NAME	: TMP.ALP float

When **lpc_cepst** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

WINDOW LENGTH

Set the duration of windowing function.

WINDOW TYPE

Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FRAME PERIOD

Set the duration of frame period.

PREEMPHASIS

Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

ORDER OF LPC

Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

ORDER OF CEPSTRUM

Set the order of cepstrum coefficients. There are no restrictions as to the value.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT CEPSTRUM FILE NAME
Set the file name for the output of cepstrum coefficients. The size of one frame is "ORDER OF CEPSTRUM" + 1. For example, when the value of "ORDER OF CEPSTRUM" is set to 16, the size of one frame array is 17. The format of output data is single-floating binary.

OUTPUT ALPHA FILE NAME
Set the file name for the output of alpha-parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 17. The format of output data is single-floating binary.

SEE ALSO
SpeechTools(1), cep_dist(1)

AUTHOR
Seiichi TENPAKU

NAME

lpc_rev – Calculate moving-average parameters.

SYNOPSIS

lpc_rev *filename*
lpc_rev -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **lpc_rev .**
The *parameters* of **lpc_rev** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FFT LENGTH	: 1024
FRAME PERIOD	: 5 msec
PREEMPHASIS	: 0.0
ORDER OF LPC	: 16
INPUT FILE NAME	: TMP.DAT float
MA PARAMETER FILE NAME	: TMP.BET float

When **lpc_rev** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FFT LENGTH
Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

FRAME PERIOD
Set the duration of frame period.

PREEMPHASIS
Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

ORDER OF LPC
Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

MA PARAMETER FILE NAME

Set the file name for the output of moving-average parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 17. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), lpc_run(1)

AUTHOR

Seiichi TENPAKU

NAME

lpc_run – Calculate LPC running spectra.

SYNOPSIS

lpc_run *filename*
lpc_run –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **lpc_run**.
The *parameters* of **lpc_run** are listed below;

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	: TMP.DAT float
OUTPUT LPC FILE NAME	: TMP.LPC float
OUTPUT ALPHA FILE NAME	: TMP.ALP float

When **lpc_run** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

WINDOW LENGTH

Set the duration of windowing function.

WINDOW TYPE

Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FFT LENGTH

Set the number of DFT points. The value must be 2^n, and it must be longer than "SAMPLING FREQUENCY" × "WINDOW LENGTH". There are no restrictions as to the value.

FRAME PERIOD

Set the duration of frame period.

PREEMPHASIS

Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

ORDER OF LPC

Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary

float single precision floating binary

There are no restrictions as to the file length.

OUTPUT LPC FILE NAME

Set the file name for the output of LPC running spectra, which is log power spectra data in linear frequency scale. The size of one frame is a half of "FFT LENGTH". For example, when the value of "FFT LENGTH" is set to 1024, the size of one frame array is 512. The format of output data is single-floating binary.

OUTPUT ALPHA FILE NAME

Set the file name for the output of alpha-parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 17. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dcep(1), ftrack(1), lpc_cepst(1), parcor(1), peak_pick(1)

AUTHOR

Seiichi TENPAKU

NAME

lpf1 – IIR low-pass filter.

SYNOPSIS

lpf1 *filename*

lpf1 –o *parameter* ...

USAGE

filename contains *parameter*s, which are concerned with **lpf1.**

The *parameter*s are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
CUT OFF FREQUENCY	: 100.0 Hz
INPUT FILE NAME	: TMP.DAT short
OUTPUT FILE NAME	: TMP.OUT short

When the command name is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

CUT OFF FREQUENCY

Set the cut off frequency. The upper limit of the cut off frequency is a quarter of the sampling frequency. For example, when the sampling frequency 20 [kHz], the upper limit of the cut off frequency is 5000 [Hz].

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME

Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), lpf1(1), differ(1), syn_pole(1), syn_zero(1)

AUTHOR

Seiichi TENPAKU

NAME

lstsq_smooth – Smoothing by least square method.

SYNOPSIS

lstsq_smooth *filename*
lstsq_smooth **–o** *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **lstsq_smooth**.
The *parameters* of **lstsq_smooth** are listed below;

SMOOTHING POINTS	: 200
POLYNOMIAL ORDER (<= 20)	: 10
INPUT FILE NAME	: TMP.ORG float
OUTPUT FILE NAME	: TMP.OUT float

When **lstsq_smooth** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SMOOTHING POINTS
Set the smoothing points.

POLYNOMIAL ORDER (<= 20)
Set the polynomial order. The value must be less than 20.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

make_pair – Making Pairs.

SYNOPSIS

make_pair *filename*
make_pair –o *parameter* ...

DESCRIPTION

make_pair makes paired-lists.

USAGE

filename contains *parameters*, which are concerned with **make_pair**.
The *parameters* of **make_pair** are listed below;

```
INPUT LIST FILE NAME      : TMP.LST ascii
OUTPUT LIST FILE NAME     : TMP.TBL ascii
```

When **make_pair** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

INPUT LIST FILE NAME
Set the file name of input lists.

OUTPUT LIST FILE NAME
Set the file name of output lists.

EXAMPLE

```
example% make_pair
Making Pairs.
usage :: make_pair filename
         make_pair -o arguments

Defaults are as follows.
INPUT LIST FILE NAME      : TMP.LST ascii
OUTPUT LIST FILE NAME     : TMP.TBL ascii
example% make_pair > temp
Making Pairs.
usage :: make_pair filename
         make_pair -o arguments

Defaults are as follows.
example% cat TMP.LST
a
b
c
example% make_pair temp
example% cat TMP.TBL
a	b
a	c
b	a
b	c
c	a
c	b
```

SEE ALSO

SpeechTools(1), fileshuffle(1)

AUTHOR

Seiichi TENPAKU

NAME

median_smooth – Median Smoothing.

SYNOPSIS

median_smooth *filename*
median_smooth –o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **median_smooth**.
The *parameters* of **median_smooth** are listed below;

MEDIAN BLOCK NUMBER	: 5
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When **median_smooth** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

MEDIAN BLOCK NUMBER
Set the block number.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

EXAMPLE

```
example% median_smooth
Median Smoothing.
usage :: median_smooth filename
         median_smooth -o arguments

Defaults are as follows.
MEDIAN BLOCK NUMBER       : 5
INPUT FILE NAME           : TMP.DAT float
OUTPUT FILE NAME          : TMP.OUT float
example% bcat TMP.DAT
0.000000
1.000000
2.000000
0.000000
1.000000
2.000000
0.000000
1.000000
2.000000
0.000000
example% median_smooth -o "MEDIAN BLOCK NUMBER : 3"
```

```
!10
example% bcat TMP.OUT
0.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
0.000000
```

SEE ALSO

SpeechTools(1), bcat(1)

AUTHOR

Seiichi TENPAKU

NAME
merge – merge ascii streams

SYNOPSIS
merge *files* ...

DESCRIPTION
merge merges ascii data columns side-by-side and displays the results on the standard output.

SEE ALSO
Utility(1), separate(1)

EXAMPLE
```
example% cat A
a       1
b       2
c       3
example% cat B
2
4
6
example% merge A B
a       1       2
b       2       4
c       3       6
```

AUTHOR
Seiichi TENPAKU

NAME

noise_wave – Generating a band noise.

SYNOPSIS

noise_wave *filename*
noise_wave –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **noise_wave.**
The *parameters* of **noise_wave** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
START FREQUENCY	: 1000 Hz
END FREQUENCY	: 1000 Hz
STEP FREQUENCY	: 10 Hz
AMPLITUDE	: 20000
DC BIAS	: 0
DATA LENGTH	: 200 msec
SIGNAL LENGTH	: 100 msec
OFFSET TIME	: 50 msec
RISE ENVELOPE	: 10 msec
FALL ENVELOPE	: 10 msec
NOISE SEED	: 257
OUTPUT FILE NAME	: TMP.SIG float

When **noise_wave** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency.

START FREQUENCY
Set the start frequency.

END FREQUENCY
Set the end frequency.

STEP FREQUENCY
Set the step frequency.

When the value of "START FREQUENCY" is equal to the value of "END FREQUENCY", the generating frequency is constant. On the other hand, when the value of "START FREQUENCY" is not equal to the value of "END FREQUENCY", the generating frequency is stepped from the value of "START FREQUENCY" to the value of "END FREQUENCY" at the value of "STEP FREQUENCY".

AMPLITUDE
Set the amplitude.

DC BIAS
Set the DC bias. If the value of "DC BIAS" is 0, there are no effects on DC.

DATA LENGTH
Set the duration of whole data.

SIGNAL LENGTH
Set the duration of just generating signal. The value of "SIGNAL LENGTH" must be less than the value of "DATA LENGTH".

OFFSET TIME
Set the offset time of generating signal. The value of "OFFSET TIME" must be less than the value of subtract "DATA LENGTH" from "SIGNAL LENGTH".

RISE ENVELOPE

Set the duration of rising envelope. The value of "RISE ENVELOPE" must be less than the value of "SIGNAL LENGTH".

FALL ENVELOPE

Set the duration of falling envelope. The value of "FALL ENVELOPE" must be less than the value of "SIGNAL LENGTH".

NOISE SEED

Set the ranodom seed. If the value of "NOISE SEED" is equal to 0, In each frequency, the phases of waveform are started at 0.

OUTPUT FILE NAME

Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), sig_wave(1)

AUTHOR

Seiichi TENPAKU

NAME

npulse – Generating a pulse/noise source.

SYNOPSIS

npulse *filename*
npulse -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **npulse.**
The *parameters* of **npulse** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
FRAME PERIOD	: 5 msec
F0 FILE NAME	: TMP.F0 float
AV FILE NAME	: TMP.AV float
AF FILE NAME	: TMP.AF float
OUTPUT FILE NAME	: TMP.OUT float

When **npulse** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

FRAME PERIOD
Set the duration of frame period.

F0 FILE NAME
Set the file name for fundamental frequency control data.

AV FILE NAME
Set the file name for amplitude of pulse control data.

AF FILE NAME
Set the file name for amplitude of noise control data.

OUTPUT FILE NAME
Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), pulse(1)

AUTHOR

Seiichi TENPAKU

NAME

parcor – Calculate PARCOR parameters.

SYNOPSIS

parcor *filename*

parcor –o *parameter* ...

USAGE

filename contains *parameter*s, which are concerned with **parcor** .

The *parameter*s of **parcor** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FRAME PERIOD	: 5 msec
PREEMPHASIS	: 0.98
ORDER OF LPC	: 16
INPUT FILE NAME	: TMP.DAT float
ALPHA FILE NAME	: TMP.ALP float
PARCOR FILE NAME	: TMP.PAR float

When **parcor** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

WINDOW LENGTH

Set the duration of windowing function.

WINDOW TYPE

Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FRAME PERIOD

Set the duration of frame period.

PREEMPHASIS

Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

ORDER OF LPC

Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT ALPHA FILE NAME

Set the file name for the output of alpha-parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 17. The format of output data is single-floating binary.

OUTPUT PARCOR FILE NAME

Set the file name for the output of alpha-parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 17. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dcep(1), ftrack(1), lpc_cepst(1), lpc_run(1), peak_pick(1)

AUTHOR

Seiichi TENPAKU

NAME

peak_pick – Peak-Picking for formants.

SYNOPSIS

peak_pick *filename*
peak_pick -o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **peak_pick**.
The *parameters* of **peak_pick** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
ARRAY SIZE	: 512
NUMBER OF PEAK	: 5
INVERSION OPTION	: OFF
AUTO REGRESSIVE FUNCTION	: ON
INPUT FILE NAME	: TMP.SPC float
FREQUENCY FILE NAME	: TMP.FRQ float
BANDWIDTH FILE NAME	: TMP.BND float
AMPLITUDE FILE NAME	: TMP.AMP float

When **peak_pick** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency.

ARRAY SIZE
Set the one frame size of input data.

NUMBER OF PEAK
Set the peak-picking number.

INVERSION OPTION
Set the inversion option. When the value is "ON", the input data will be inversed.

AUTO REGRESSIVE FUNCTION
Set the auto-regressive function. When the value is "ON", the input data will be applied the auto-regressive function.

INPUT FILE NAME
Set the file name of input data. The input file contains log power spectra data in linear frequency scale. For example, these files are made by using **lpc_run, dft_cep** and so on. The format of input data must be single-floating binary. There are no restrictions as to the file length.

FREQUENCY FILE NAME
Set the file name for the output of formant frequency data. The size of one frame is equal to the value of "NUMBER OF PEAK". The format of output data is single-floating binary.

BANDWIDTH FILE NAME
Set the file name for the output of formant bandwidth data. The size of one frame is equal to the value of "NUMBER OF PEAK". The format of output data is single-floating binary.

AMPLITUDE FILE NAME
Set the file name for the output of formant peak amplitude data. The size of one frame is equal to the value of "NUMBER OF PEAK". The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1), dft_cep(1), ftrack(1), lpc_run(1)

AUTHOR

Seiichi TENPAKU

NAME

pitcher – Pitch extraction by auto-correlation method.

SYNOPSIS

pitcher *filename*
pitcher -o *parameter ...*

USAGE

filename contains *parameter*s, which are concerned with **pitcher**.
The *parameter*s of **pitcher** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FRAME PERIOD	: 5 msec
MINIMUM FREQUENCY	: 70 Hz
MAXIMUM FREQUENCY	: 300 Hz
THRESHOLD POWER	: 40 dB
POLARIZATION	: ON
INPUT FILE NAME	: TMP.DAT float
PITCH FILE NAME	: TMP.PIT float
POWER FILE NAME	: TMP.POW float

When **pitcher** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FRAME PERIOD
Set the duration of frame period.

MINIMUM FREQUENCY
Set the minimum fundamental frequency to search the pitch.

MAXIMUM FREQUENCY
Set the maximum fundamental frequency to search the pitch.

THRESHOLD POWER
Set the threshold power value. If the power value of one frame data is less than the value of "THRESHOLD POWER", the pitch extraction can not apply the frame.

POLARIZATION
Set the polarization function. If the value is ON, the input data is polarized before pitch extraction. That meens the value x[i] of input data is changed to 1 if x[i] >= 0, or to 0 if x[i] < 0, then **pitcher** calculates autocorrelation and picks the peak.

INPUT FILE NAME

Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

PITCH FILE NAME

Set the file name for the output of the extracted pitch. The format of output data is single-floating binary.

POWER FILE NAME

Set the file name for the output of the calculated power. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

power – Calculation of power with window normalization.

SYNOPSIS

power *filename*
power -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **power**.
The *parameters* of **power** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FRAME PERIOD	: 5 msec
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When **power** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FRAME PERIOD
Set the duration of frame period.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output of results. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

pulse – Pulse generation

SYNOPSIS

pulse *filename*
pulse -o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **pulse**.
The *parameters* of **pulse** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
FRAME PERIOD	: 5 msec
F0 FILE NAME	: TMP.F0 float
AV FILE NAME	: TMP.AV float
OUTPUT FILE NAME	: TMP.PL float

When **pulse** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

FRAME PERIOD
Set the duration of frame period.

F0 FILE NAME
Set the file name for fundamental frequency control data.

AV FILE NAME
Set the file name for amplitude of pulse control data.

OUTPUT FILE NAME
Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), npulse(1)

AUTHOR

Seiichi TENPAKU

NAME

separate – split an ascii stream

SYNOPSIS

separate *n* [*file*]
separate -r *m n* [*file*]
separate -p *m n* [*file*]
separate -s '*l m n ...*' [*file*]

DESCRIPTION

1) **separate** *n* [*file*]
Get the *n*th column.
2) **separate** -r *m n* [*file*]
Get columns *m* through *n*.
3) **separate** -p *m n* [*file*]
Get columns *m* and *n*.
4) **separate** -s '*l m n ...*' [*file*]
Get columns *l*, *m*, *n*, ...

In each case, **separate** displays the results on the standard output. The column number starts at 1. If *file* is not present, **separate** reads data from the standard input.

SEE ALSO

Utility(1), merge(1)

EXAMPLE

```
example% cat data.ascii
a       1       2       X       100
b       2       4       Y       200
c       3       6       Z       300
example% separate 1 data.ascii
a
b
c
example% separate 2 data.ascii
1
2
3
example% separate -r 2 4 data.ascii
1       2       X
2       4       Y
3       6       Z
example% separate -r 4 2 data.ascii
1       2       X
2       4       Y
3       6       Z
example% separate -p 1 3 data.ascii
a       2
b       4
c       6
example% separate -p 5 2 data.ascii
100     1
200     2
300     3
example% separate -s '5 3 1' data.ascii
100     2       a
```

```
200    4    b
300    6    c
example% cat data.ascii | separate 3
2
4
6
```

AUTHOR

Seiichi TENPAKU

NAME

residual – Calculate residual error.

SYNOPSIS

residual *filename*
residual -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **residual**.
The *parameters* of **residual** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FRAME PERIOD	: 5 msec
PREEMPHASIS	: 0.98
ORDER OF LPC	: 16
INPUT FILE NAME	: TMP.DAT float
ALPHA FILE NAME	: TMP.ALP float
RESIDUAL FILE NAME	: TMP.RES float

When **residual** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FRAME PERIOD
Set the duration of frame period.

PREEMPHASIS
Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

ORDER OF LPC
Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

ALPHA FILE NAME

Set the file name for the input of alpha-parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array is 17. The format of alpha-parameters data is single-floating binary.

RESIDUAL FILE NAME

Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), syn_alpha(1),

AUTHOR

Seiichi TENPAKU

NAME

sig_wave – Generating a simple signal waveform.

SYNOPSIS

sig_wave *filename*
sig_wave -o *parameter ...*

USAGE

filename contains *parameters*, which are concerned with **sig_wave**.
The *parameters* of **sig_wave** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
START FREQ.	: 1000 Hz
END FREQ.	: 1000 Hz
AMPLITUDE	: 20000
DC BIAS	: 0
DATA LENGTH	: 200 msec
SIGNAL LENGTH	: 100 msec
OFFSET TIME	: 50 msec
RISE ENVELOPE	: 10 msec
FALL ENVELOPE	: 10 msec
SIGNAL FUNCTION	: SINE
OUTPUT FILE NAME	: TMP.SIG float

When **sig_wave** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETER

SAMPLING FREQUENCY
    Set the sampling frequency.

START FREQ.
    Set the start frequency.

END FREQ.
    Set the end frequency.

When the value of "START FREQ." is equal to the value of "END FREQ.", the generating frequency is constant. On the other hand, when the value of "START FREQ." is not equal to the value of "END FREQ.", the generating frequency is moved from the value of "START FREQ." to the value of "END FREQ.".

AMPLITUDE
    Set the amplitude.

DC BIAS
    Set the DC bias. If the value of "DC BIAS" is 0, there are no effects on DC.

DATA LENGTH
    Set the duration of whole data.

SIGNAL LENGTH
    Set the duration of just generating signal. The value of "SIGNAL LENGTH" must be less than the value of "DATA LENGTH".

OFFSET TIME
    Set the offset time of generating signal. The value of "OFFSET TIME" must be less than the value of subtract "DATA LENGTH" from "SIGNAL LENGTH".

RISE ENVELOPE
    Set the duration of rising envelope. The value of "RISE ENVELOPE" must be less than the value of "SIGNAL LENGTH".

FALL ENVELOPE

Set the duration of falling envelope. The value of "FALL ENVELOPE" must be less than the value of "SIGNAL LENGTH".

SIGNAL FUNCTION

Set the type of simple signal function. The type of simple signal function can be chosen from the following table:

SINE
RECTANGULAR
TRIANGULAR
SAW

OUTPUT FILE NAME

Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), noise_wave(1)

AUTHOR

Seiichi TENPAKU

NAME

smath – calculate arithmetic functions using the standard input/output stream.

SYNOPSIS

smath *function* [*argument*]

smath help

DESCRIPTION

smath reads data from the standard input and calculates using the arithmetic function. Then, **smath** prints out the results to the standard output. In following explanations, The known arithmetic functions are as follows:

erf(x) returns the error function of x.
erfc(x) returns 1.0–erf(x).
exp(x) returns the exponential function e^x.
log(x) returns the natural logarithm of x.
log10(x) return the logarithm to base 10.
pow(x,y) returns x^y. pow(x,0.0) is 1 for all x.
pow10(x) returns 10^x.
sqrt(x) returns the square root of x.
cbrt(x) returns the cube root of x.
sinh(x) returns the hyperbolic sine of x.
cosh(x) returns the hyperbolic cosine of x.
tanh(x) returns the hyperbolic tangent of x.
asinh(x) returns the inverse hyperbolic sine of x.
acosh(x) returns the inverse hyperbolic cosine of x.
atanh(x) returns the inverse hyperbolic tangent of x.
sin(x) returns the sine function of x. x are radian arguments.
cos(x) returns the cosine function of x. x are radian arguments.
tan(x) returns the tangent function of x. x are radian arguments.
asin(x) returns the arc sine in the range $-\pi/2$ to $\pi/2$.
acos(x) returns the arc cosine in the range 0 to π.
atan(x) returns the arc tangent of x in the range $-\pi/2$ to $\pi/2$.
fabs(x) returns the absolute value of x.
floor(x) returns the greatest integral value less than or equal to x.
ceil(x) returns the least integral value greater than or equal to x.
add(x,y) returns the value of $(x + y)$.
sub(x,y) returns the value of $(x - y)$.
mul(x,y) returns the value of $(x \times y)$.
div(x,y) returns the value of $(x \div y)$.
inv(x) returns the value of $(1 \div x)$.

NOTE

A comment line starts at the '#' sign and ends at the next *NEWLINE*. The comment line is ignored when the **smath** runs.

SEE ALSO

Utility(1)

EXAMPLE

```
example% cat data.ascii
1
2
3
example% cat data.ascii | smath mul 10
10.000000
20.000000
```

```
30.000000
example% cat data.ascii | smath log10
0.000000
0.301030
0.477121
example% cat data.ascii | smath log10 | smath mul 10
0.000000
3.010300
4.771210
```

AUTHOR

Seiichi TENPAKU

NAME

spc_dist – Calculate Spectrum Distortion.

SYNOPSIS

spc_dist *filename*
spc_dist –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **spc_dist**.
The *parameters* of **spc_dist** are listed below;

SIZE OF ARRAY	: 512
ORIGINAL FILE NAME	: TMP.ORG float
REFERENCE FILE NAME	: TMP.REF float
DIFFERENCE FILE NAME	: TMP.DIS float
INFORMATION FILE NAME	: TMP.IFO ascii

When **spc_dist** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SIZE OF ARRAY
: Set the array size of one frame data.

ORIGINAL FILE NAME
: Set the file name of input original data. The format of input data must be single-floating binary.

REFERENCE FILE NAME
: Set the file name of input reference data. The format of input data must be single-floating binary.

DIFFERENCE FILE NAME
: Set the file name of output differences in each frame. The format of input data is single-floating binary.

INFORMATION FILE NAME
: Set the file name for output of results.

SEE ALSO

SpeechTools(1), cep_dist(1), euclid_dist(1),

AUTHOR

Seiichi TENPAKU

NAME

subset – Cutting a subset of the file.

SYNOPSIS

subset *filename*
subset –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **subset**.
The *parameters* of **subset** are listed below;

SOURCE FILE NAME	: TMP.SRC
RESULT FILE NAME	: TMP.DST
SIZE OF ARRAY	: 2 byte
OFFSET	: 0
LENGTH	: 100

When **subset** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SOURCE FILE NAME
Set the file name of input data. The format of input data must be binary. There are no restrictions as to the file length.

RESULT FILE NAME
Set the file name for the output of results. The format of output data is binary.

SIZE OF ARRAY
Set the size of data in byte-order.

OFFSET
Set the offset of the input file.

LENGTH
Set the length of the input file. The value of "LENGTH" must be less than the value of subtract the total length of the input data from the value of "OFFSET". For example, when the format of input data is short integer binary, the value of "SIZE OF ARRAY" is set to 2. If the size of the input data is 200 bytes, the total length is 100. And if the value of "OFFSET" is set to 20, the value of "LENGTH" must be less than 80 (= 100 - 20).

EXAMPLE

```
example% subset
Cutting a subset of the file.
usage :: subset filename
        subset -o arguments

Defaults are as follows.
SOURCE FILE NAME      : TMP.SRC
RESULT FILE NAME      : TMP.DST
SIZE OF ARRAY         : 2 byte
OFFSET                : 0
LENGTH                : 100
example% bcat -s TMP.SRC
1
2
3
4
5
6
```

```
7
8
9
10
example% subset -o "OFFSET : 3" "LENGTH : 5"
!10
example% bcat -s TMP.DST
4
5
6
7
8
```

SEE ALSO

SpeechTools(1), bcat(1)

AUTHOR

Seiichi TENPAKU

NAME

syn_alpha – Synthesizing with alpha parameters

SYNOPSIS

syn_alpha *filename*
syn_alpha -o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **syn_alpha.**
The *parameters* of **syn_alpha** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
FRAME PERIOD	: 5 msec
ORDER OF LPC	: 16
INPUT FILE NAME	: TMP.DAT float
ALPHA FILE NAME	: TMP.ALP float
OUTPUT FILE NAME	: TMP.OUT float

When **syn_alpha** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing function.

WINDOW TYPE
Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

FRAME PERIOD
Set the duration of frame period.

PREEMPHASIS
Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

ORDER OF LPC
Set the order of LPC (Linear Prediction Coefficients). There are no restrictions as to the value.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

ALPHA FILE NAME
Set the file name for the input of alpha-parameters. The size of one frame is "ORDER OF LPC" + 1. For example, when the value of "ORDER OF LPC" is set to 16, the size of one frame array

is 17. The format of alpha-parameters data is single-floating binary.

OUTPUT FILE NAME

Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), residual(1),

AUTHOR

Seiichi TENPAKU

NAME

syn_cascade – Cascade formant synthesizer.

SYNOPSIS

syn_cascade *filename*

syn_cascade –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **syn_cascade**.
The *parameters* of **syn_cascade** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
FRAME PERIOD	: 5 msec
PREEMPHASIS	: 0.98
GAIN CONTROL	: 80 dB
NUMBER OF FORMANTS	: 5
INVERSE FILTERING MODE	: OFF
SOURCE VOICE FILE NAME	: TMP.SRC float
FORMANT FREQUENCY FILE NAME	: TMP.FRQ float
FORMANT BANDWIDTH FILE NAME	: TMP.BND float
OUTPUT FILE NAME	: TMP.SYN float

When **syn_cascade** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
: Set the sampling frequency.

FRAME PERIOD
: Set the duration of frame period.

PREEMPHASIS
: Set the pre-emphasis factor. The range of this value is 0.0 to 1.0.

GAIN CONTROL
: Set the gain. The maximum amplitude of the synthesized speech is adjusted to the value of "GAIN CONTROL".

NUMBER OF FORMANTS
: Set the number of formants.

INVERSE FILTERING MODE
: Set the mode of filtering. If the value is "ON", the inverse filtering mode is set.

SOURCE VOICE FILE NAME
: Set the file name of source data. Source data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

When there are no restrictions as to the file length.

FORMANT FREQUENCY FILE NAME
: Set the file name of input formant frequency data. The format of input formant frequency data must be single-floating binary.

FORMANT BANDWIDTH FILE NAME
: Set the file name of input formant bandwidth data. The format of input formant bandwidth data must be single-floating binary.

OUTPUT FILE NAME

Set the file name for the output of synthesized speech. The format of output data is single-floating binary.

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

syn_parcor – PARCOR Synthesizer.

SYNOPSIS

syn_parcor *filename*
syn_parcor –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **syn_parcor**.
The *parameters* of **syn_parcor** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
FRAME PERIOD	: 5 msec
ORDER OF LPC	: 16
GAIN CONTROL	: 80.0 dB
SOURCE FILE NAME	: TMP.SRC float
PARCOR FILE NAME	: TMP.PAR float
OUTPUT FILE NAME	: TMP.SYN float

When **syn_parcor** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency.

FRAME PERIOD
Set the duration of frame period.

ORDER OF LPC
Set the order of LPC (Linear Prediction Coefficients).

GAIN CONTROL
Set the gain. The maximum amplitude of the synthesized speech is adjusted to the value of "GAIN CONTROL".

SOURCE FILE NAME
Set the file name of source data. Source data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

PARCOR FILE NAME
Set the file name of input PARCOR parameters data. The format of input data must be single-floating binary.

OUTPUT FILE NAME
Set the file name for the output of synthesized speech. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), parcor(1)

AUTHOR

Seiichi TENPAKU

NAME

syn_pole – Single formant filter.

SYNOPSIS

syn_pole *filename*
syn_pole –o *parameter* ...

USAGE

filename contains *parameter*s, which are concerned with **syn_pole**. The *parameter*s are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
CENTER FREQUENCY	: 0.0 Hz
BANDWIDTH	: 200.0 Hz
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When the command name is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

CENTER FREQUENCY
Set the center frequency.

BANDWIDTH
Set the bandwidth.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), syn_zero(1), differ(1), hpf1(1), lpf1(1)

AUTHOR

Seiichi TENPAKU

NAME

taper – Tapering the data.

SYNOPSIS

taper *filename*
taper –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **taper** .
The *parameters* of **taper** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
RISE ENVELOPE	: 10 msec
FALL ENVELOPE	: 10 msec
OFFSET	: 0.0 msec
LENGTH	: -1.0 msec
TAPER FUNCTION	: LINEAR
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When **taper** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency.

RISE ENVELOPE
Set the rising duration.

FALL ENVELOPE
Set the falling duration.

OFFSET
Set the offset time of tapering function.

LENGTH
Set the duration of tapering function. When the value of "LENGTH" is negative, the duration is adjusted to the file length of the input data.

TAPER FUNCTION
Set the type of tapering function. The type of tapering function can be chosen from the following table:

LINEAR
SECOND
SINE
SINC

If an unknown type of tapering function is specified, the type of tapering function is automatically reduced to LINEAR without messages.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary

double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

NAME

syn_zero – Single anti-formant filter.

SYNOPSIS

syn_zero *filename*
syn_zero –o *parameter* ...

USAGE

filename contains *parameters*, which are concerned with **syn_zero**. The *parameters* are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
CENTER FREQUENCY	: 0.0 Hz
BANDWIDTH	: 200.0 Hz
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.OUT float

When the command name is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY
Set the sampling frequency of input data.

CENTER FREQUENCY
Set the center frequency.

BANDWIDTH
Set the bandwidth.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1), syn_pole(1), differ(1), hpf1(1), lpf1(1)

AUTHOR

Seiichi TENPAKU

NAME

transpose – Transpose array (X <=> Y).

SYNOPSIS

transpose *filename*
transpose –o *parameter* ...

DESCRIPTION

transpose exchanges a horizontal axis for a vertical axis.

USAGE

filename contains *parameters*, which are concerned with **transpose.**
The *parameters* of **transpose** are listed below;

```
INPUT FILE NAME        : TMP.SRC
OUTPUT FILE NAME       : TMP.DST
SIZE OF DATA           : 2 byte
NUMBER OF X AXIS       : 16
```

When **transpose** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS

INPUT FILE NAME
Set the file name of input data. The format of input data must be binary. There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output of results. The format of output data is binary.

SIZE OF DATA
Set the size of data in byte-order. For example, when the format of input data is short integer binary, the value of "SIZE OF DATA" must be set to 2.

NUMBER OF X AXIS
Set the item number of the horizontal axis.

EXAMPLE

```
example% transpose
Transpose array (X <=> Y).
usage :: transpose filename
       transpose -o arguments

Defaults are as follows.
INPUT FILE NAME        : TMP.SRC
OUTPUT FILE NAME       : TMP.DST
SIZE OF DATA           : 2 byte
NUMBER OF X AXIS       : 16
example% acat -s TMP.SRC
1 2
3 4
5 6
example% transpose -o "NUMBER OF X AXIS : 2"
!12
example% bcat -p 3 -s TMP.DST
1              3              5
2              4              6
```

SEE ALSO

SpeechTools(1), acat(1), bcat(1)

AUTHOR

Seiichi TENPAKU

NAME
zerocrs – Counting the zero cross points.

SYNOPSIS
zerocrs *filename*
zerocrs -o *parameter ...*

USAGE
filename contains *parameters*, which are concerned with **zerocrs**.
The *parameters* of **zerocrs** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
FRAME PERIOD	: 5 msec
INPUT FILE NAME	: TMP.DAT float
OUTPUT FILE NAME	: TMP.ZRO short

When **zerocrs** is entered without any arguments, the command displays the *parameters* and default settings on the standard output.

PARAMETERS
SAMPLING FREQUENCY
Set the sampling frequency of input data.

WINDOW LENGTH
Set the duration of windowing.

FRAME PERIOD
Set the duration of frame period.

INPUT FILE NAME
Set the file name of input data. Input data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

There are no restrictions as to the file length.

OUTPUT FILE NAME
Set the file name for the output of the results. The format of output data is short integer binary.

SEE ALSO
SpeechTools(1), find_zerocrs(1)

AUTHOR
Seiichi TENPAKU

NAME

winfun – Generating time-window functions.

SYNOPSIS

winfun *filename*

winfun –o *parameter ...*

USAGE

filename contains *parameter*s, which are concerned with **winfun.**

The *parameter*s of **winfun** are listed below;

SAMPLING FREQUENCY	: 20.0 kHz
WINDOW LENGTH	: 30 msec
WINDOW TYPE	: HANNING
AMPLITUDE	: 20000
OUTPUT FILE NAME	: TMP.OUT float

When **winfun** is entered without any arguments, the command displays the *parameter*s and default settings on the standard output.

PARAMETERS

SAMPLING FREQUENCY

Set the sampling frequency of input data.

WINDOW LENGTH

Set the duration of windowing function.

WINDOW TYPE

Set the type of windowing function. The type of windowing function can be chosen from the following table:

RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

If an unknown type of windowing function is specified, the type of windowing function is automatically reduced to RECTANGULAR without messages.

AMPLITUDE

Set the maximum amplitude of output data.

OUTPUT FILE NAME

Set the file name of output data. Output data format can be one (only) among :

short	short integer binary
int	integer binary
double	double precision floating binary
float	single precision floating binary

SEE ALSO

SpeechTools(1)

AUTHOR

Seiichi TENPAKU

Library(3)	libcommand.a, libst.a, libmx.a, libtlist – library routines of SpeechTools
alp_cep(3)	alp_cep_float, alp_cep_double, – conversion of linear prediction parameters into cepstrum parameters.
alp_par(3)	alp_par_float, alp_par_double, – conversion of linear prediction parameters into PARCOR parameters.
average(3)	average_float, average_double, average_short, average_int – calculate the average of value in an array.
axis(3)	MelToFreq, FreqToMel, FreqToBark, BarkToFreq, StageToBark, StageToFreq, critical_bandwidth, critical_frequency – frequency axis conversion.
cepstrum(3)	cepstrum, cepstrum_envelope - DFT cepstrum analysis.
column(3)	columnGet – get a string from a column string.
command(3)	CommandInit – command interface initializer for SpeechTools.
complex(3)	cx_zero, cx_add, cx_sub, cx_mul, cx_div, cx_conj, cx_exp, cx_pow, cx_root, cx_tocomplex, cx_abs, cx_arg, cx_print – complex function utilities.
convert(3)	convertArray, convertRead, convertWrite – convert data type of array.
cor_alp(3)	cor_alp_float, cor_alp_double – conversion of autocorrelation coefficients into linear prediction parameters.
correlation(3)	correlation_float, correlation_double – calculate correlation coefficients.
count(3)	countLine, countChar, countWord – calculate a number of lines, words and characters in file.
critical(3)	critical_damp – second order critically damped model.
differential(3)	differential_float, differential_double, differential_short, differential_int – calculate a the differential of an array.
dka(3)	aberth_init, dka_method, dka_solve – solving a high order polynomial expression.
dot(3)	dot_float, dot_double, dot_short, dot_int – calculate an inner product.
fft(3)	fft, ifft – discrete Fourier transform.
file(3)	fopenp, get_file_length, openRead, openWrite, binRead, binStore – file utilities of SpeechTools.
filter(3)	filter_lpf1, filter_hpf1, filter_dif1 – IIR filter function utilities.
find(3)	find_float_zero, find_float_peak, find_float_max, find_float_min, find_float_absmax, find_float_absmin, find_double_zero, find_double_peak, find_double_max, find_double_min, find_double_absmax, find_double_absmin, find_short_zero, find_short_peak, find_short_max, find_short_min, find_short_absmax, find_short_absmin – finding utilities.
gain(3)	gain_control_float, gain_control_double, gain_control_short – amplify the value of array.
getabspath(3)	getabspath – return the absolute pathname.
gethome(3)	gethome – get pathname of user's home directory.
getpwd(3)	getpwd – get the current working directory pathname.
interface(3)	IntrfcLineGet, IntrfcFindUnit, IntrfcPktFind, IntrfcPktAset, IntrfcPktRead, IntrfcPktWrite, IntrfcFileRead, IntrfcFileWrite, IntrfcPktStr, IntrfcStrPkt, IntrfcPktGet, IntrfcPktSet, – command interface utilities of SpeechTools.

interpol(3)	interpol_gettype, interpol_linear, interpol_lagrange, interpol_blend_init, interpol_blend, interpol_spline_init, interpol_spline – interpolate function utilities.
lagwin(3)	lagwin_float, lagwin_double – calculate a binomial window coefficients.
lip(3)	lip_inverse_float, lip_inverse_double – eliminate lip radiation characteristics.
lsp(3)	lsp_calc - calculate LSP (line spectrum pair) coefficients.
lstsq(3)	lstsq_float, lstsq_calc_float, lstsq_double, lstsq_calc_double – least square method.
median(3)	median_float – get the median value fo array.
message(3)	debugMsgSet, debugMsg, errorMsgSet, errorMsg, warningMsgSet, warningMsg – message function utilities of SpeechTools.
meter(3)	meterSwitch, meterSet, meterInc, meterEnd – meter function utilities of SpeechTools.
mxalloc(3)	mx1alloc, mx2alloc, mx3alloc, mx1free, mx2free, mx3free – matrix allocation function utilities.
mxcholesky(3)	mx_cholesky – matrix solve function utilities with Cholesky method.
mxhouse(3)	mx_householder_solve – matrix solve function utilities with Householder method.
mxmul(3)	mx_trans_mul, mx_mx_mul, mx_vc_mul – matrix multiply function utilities.
mxsolve(3)	mx_ldua, mx_solve – matrix solve function utilities.
mxtrans(3)	mx_transpose – transpose matrix.
par_alp(3)	par_alp_float, par_alp_double, – conversion of PARCOR parameters into linear prediction parameters.
pole_alpha(3)	pole_alpha - calculate the parameters of transfer function from the pairs of frequency and bandwidth.
pole_zero(3)	pole_zero, pole_zero_fast - calculate the pairs of frequency and bandwidth from the parameters of transfer function.
random(3)	random_var, random_num, random_seed – pseudo-random number generator.
sig_cor(3)	sig_cor_float, sig_cor_double, – calculate autocorrelation coefficients.
signal(3)	set_signal_handler – set the signal handlers of SpeechTools.
smooth(3)	smooth_moving_ave – smoothing.
stat(3)	stat_arithmetic_mean, stat_correlation, stat_variance – statistics functions.
string(3)	strsave, strrev, string_compare, string_partition – string utilities of SpeechTools.
swap(3)	swap_float, swap_double, swap_int, swap_long, swap_short, swap_char – swap data.
synthe(3)	syn_direct, syn_twomul, syn_pole, syn_zero – speech synthesis function utilities.
taper(3)	taper_gettype, taper_linear, taper_second, taper_sine, taper_sinc – taper function utilities.
tlist(3)	tlist_read, tlist_getword, tlist_search, tlist_parent, tlist_next, tlist_print – tiny list utilities of SpeechTools.

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*

NAME

alp_cep_float, alp_cep_double, – conversion of linear prediction parameters into cepstrum parameters.

SYNOPSIS

```
double  alp_cep_float( ALP, CEP, P, N )
float   *ALP;
float   *CEP;
int     P;
int     N;

double  alp_cep_double( ALP, CEP, P, N )
double  *ALP;
double  *CEP;
int     P;
int     N;
```

DESCRIPTION

alp_cep_float() and **alp_cep_double()** convert linear prediction parameters into cepstrum parameters and return the value of residual error.

ARGUMENT

ALP linear prediction parameters, size of array is P + 1. The value of *ALP[0]* is always 1.

CEP cepstrum parameters, size of array is N + 1. The value of *CEP[0]* is

$c_0 = \ln\sqrt{\sigma^2}$

P order of linear prediction coefficients.

N order of cepstrum parameter coefficients.

NOTE

The transfer function H(z) is related to the linear prediction coefficients α_i by the equation:

$$H(z) = \frac{\sigma^2}{1+\sum_{i=1}^{p}\alpha_i z^{-i}}$$

The cepstrum coefficients c_n are determined by

$$C(z) = \sum_{n=1}^{\infty} c_n z^{-n} = \ln H(z)$$

Therefore,

$$c_0 = \ln\sqrt{\sigma^2}$$

$$c_1 = -\alpha_1$$

$$c_n = \begin{cases} -\alpha_n - \sum_{m=1}^{n-1}\left(1-\frac{m}{n}\right)\alpha_m c_{n-m} & (1<n\le p) \\ -\sum_{m=1}^{p}\left(1-\frac{m}{n}\right)\alpha_m c_{n-m} & (p<n) \end{cases}$$

SEE ALSO

alp_par(3), cor_alp(3), par_alp(3), sig_cor(3)

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

alp_par_float, alp_par_double, – conversion of linear prediction parameters into PARCOR parameters.

SYNOPSIS

```
double  alp_par_float( ALP, PAR, P )
float   *ALP;
float   *PAR;
int     P;

double  alp_par_double( ALP, PAR, P )
double  *ALP;
double  *PAR;
int     P;
```

DESCRIPTION

alp_par_float() and **alp_par_double()** convert linear prediction parameters into PARCOR parameters and return the value of residual error.

ARGUMENT

ALP	linear prediction parameters, size of array is P + 1. The value of *ALP[0]* is always 1.
PAR	PARCOR parameters, size of array is P + 1. The value of *PAR[0]* is always 1.
P	order of linear prediction coefficients.

NOTE

The transfer function H(z) is related to the linear prediction coefficients α_i by the equation:

$$H(z) = \frac{\sigma^2}{1+\sum_{i=1}^{p}\alpha_i z^{-i}}$$

The PARCOR coefficients k_m are calculated using the equations :

$$k_m = -\alpha_m^{(m)}$$

$$\alpha_i^{(m-1)} = \frac{\alpha_i^{(m)} - \alpha_m^{(m)}\alpha_{m-i}^{(m)}}{1 - k_m^2} \quad \left[1 \le i \le m-1\right]$$

where, $m = p, p-1, \cdots, 2, 1$

in initial condition, $\alpha_m^{(p)} = \alpha_m, \ 1 \le m \le p$

The residual error σ^2 is determined by

$$\sigma^2 = \prod_{m=1}^{p}(1 - k_m^2)$$

SEE ALSO

alp_cep(3), cor_alp(3), par_alp(3), sig_cor(3)

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

average_float, average_double, average_short, average_int – calculate the average of value in an array.

SYNOPSIS

```
double  average_float( array, n )
float   *array;
int     n;

double  average_double( array, n )
double  *array;
int     n;

double  average_short( array, n )
short   *array;
int     n;

double  average_int( array, n )
int     *array;
int     n;
```

DESCRIPTION

average_float(), **average_double()**, **average_short()** and **average_int()** return an average of values contained in *array*.

SEE ALSO

stat(3)

EXAMPLE

```
# include       <stdio.h>

void    main()
  {
extern  double  average_float();
static  float   a[3];
        double  ave;

        a[0] = 1.0;
        a[1] = 2.0;
        a[2] = 3.0;

        ave = average_float( a, 3 );
        printf( "ave = %f\n", ave );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

MelToFreq, FreqToMel, FreqToBark, BarkToFreq, PointToBark, PointToFreq, critical_bandwidth, critical_frequency – frequency axis conversion.

SYNOPSIS

```
# include        <axis.h>

double  MelToFreq( Mel )
double  Mel;

double  FreqToMel( freq )
double  freq;

double  BarkToFreq( Bark )
double  Bark;

double  FreqToBark( freq )
double  freq;

double  PointToFreq( n, fft_len, Fs )
int     n, fft_len;
double  Fs;     /* sampling frequency [Hz] */

double  PointToBark( n, fft_len, Fs )
int     n, fft_len;
double  Fs;     /* sampling frequency [Hz] */

double  critical_bandwidth( f )
double  f;

double  critical_frequency( b )
double  b;
```

DESCRIPTION

MelToFreq( ) converts *mel* scale to *Hz* scale.

FreqToMel( ) converts *Hz* scale to *mel* scale.

BarkToFreq( ) converts *Bark* to *Hz.*

FreqToBark( ) converts *Hz* to *Bark.*

PointToFreq( ) converts DFT channel number to *Hz.*

PointToBark( ) converts DFT channel number to *Bark.*

critical_bandwidth( ) calculates a critical bandwidth [**E. Zwicker et al.** (1980)].

critical_frequency( ) calculates a critical frequency [**E. Zwicker et al.** (1980)].

REFERENCE

E. Zwicker and **E. Terhardt** (1980) : *Analytical expressions for critical-band rate and critical bandwidth as a function of frequency*, J. Acoust. Soc. Am. **65**, 1523-1525

EXAMPLE

```
# include        <stdio.h>
# include        <axis.h>

void    main( )
```

```
{
        int       N, L;
        double  S, F, B, M;

        S = 20.0*1000.0;
        L = 1024;
        N = 128;
        F = PointToFreq( N, L, S );
        B = FreqToBark( F );
        M = FreqToMel( M );

        printf( "F = %f [Hz], B = %f [Bark], M = %f [mel]\n", F, B, M );
}
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

cepstrum, cepstrum_envelope - DFT cepstrum analysis.

SYNOPSIS

```
int     cepstrum( SIG, CEP, n )
double  *SIG;
double  *CEP;
int     n;

int     cepstrum_envelope( CEP, ENV, n, m )
double  *CEP;
double  *ENV;
int     n;
int     m;
```

DESCRIPTION

cepstrum() calculates DFT cepstrum parameters from waveform data through DFT log power spectrum using **ifft()**.

cepstrum_envelope() calculates log power spectrum envelope from DFT cepstrum parameters after liftering cepstrum parameters using **fft()**.

ARGUMENT

SIG	waveform data, size of array is n.
CEP	DFT cepstrum parameters, size of array is n.
ENV	spectrum envelope data, size of array is n.
n	length of DFT cepstrum parameters; 2^p (p is an integer number).
m	quefrency window length

RETURN VALUE

TRUE if there is no error

FALSE if there is an error

BUGS

In order to obtain temporary memory space, **cepstrum()** and **cepstrum_envelope()** call **alloca()** in *libcommand.a*.

SEE ALSO

fft(3)

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

columnGet – get a string from a column string.

SYNOPSIS

```
char    *columnGet( src, dst, n )
char    *src, *dst;
int     n;
```

DESCRIPTION

One string would be separated by **SPACE** or **TAB** code at times. For example, "ORANGE APPLE BANANA" consists of three words: "ORANGE", "APPLE" and "BANANA". In this case, "ORANGE APPLE BANANA" has three columns; first column is "ORNAGE", second one is "APPLE" and third one is "ORANGE". **columnGet()** applies ascii table data, each line of which consists of several columns.

columnGet() copies the *n*-th string of *src* to *dst* and returns a pointer to the result. The column number starts at 1. **columnGet()** returns a **NULL** pointer, when the *n*-th string of *src* is not found. **columnGet()** does not check the array size of *dst*.

EXAMPLE

```
# include       <stdio.h>

void    main()
  {
extern  char    *columnGet();
static  char    *str = "ORANGE APPLE BANANA";
static  char    buf[80];
        int     n;

        for ( n = 1 ; n < 4 ; n++ )
                printf( "%d = %s\n", n, columnGet( str, buf, n ) );
  }
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

CommandInit – command interface initializer for SpeechTools.

SYNOPSIS

```
# include      <interface.h>

void   CommandInit( argc, argv, packets, count, doc )
int         argc;
char        *argv[];
IntrfcPkts  *packets;
int         count;
char        *doc;
```

DESCRIPTION

All *SpeechTools* commands except the seven utilities call **CommandInit()** in order to set the command parameters and to set the signal handler functions. *SpeechTools* commands have default values of parameters that are defined by *packets*. At first **CommandInit()** reads **~/.strc** file if you have and then makes following actions:

[1] prints usage briefly and a list of the parameters with their default values when no option is supplied, and terminates.

[2] reads a parameter file given with *argv[1]*.

[3] reads command arguments when **-o** option is supplied.

[4] reads the standard input *stdin* when **-i** option is supplied.

[5] prints a list of the parameters with their default values as one line quoted the symbol of (") when **-x** option is supplied, and terminates.

[6] prints usage, a list of the parameters with their default values and allowing options when **-h** option is supplied, and terminates.

CommandInit() accepts following options:

-d enable debug message

-s disable warning message

-w enable warning message

ARGUMENT

argc	the argument count.
argv	an array of character pointers to the arguments themselves.
packets	an array of **IntrfcPkts** pointers to the command parameters.
count	the count of *packets*.
doc	a string of document of the command.

SEE ALSO

SpeechTools(1), interface(3)

EXAMPLE

```
# include     <stdio.h>
# include     <interface.h>

static  IntrfcPkt packets[] =
   {
{ "SAMPLING FREQUENCY",        "20.0",            "kHz"       },/* 0 */
{ "WINDOW LENGTH",             "30",              "msec"      },/* 1 */
{ "WINDOW TYPE",               "HANNING",         NULL        },/* 2 */
```

```
{ "FFT LENGTH",            "1024",       NULL      }/* 3 */
{ "FRAME PERIOD",          "5",          "msec"    }/* 4 */
{ "PREEMPHASIS",           "0.98",       NULL      }/* 5 */
{ "INPUT FILE NAME",       "TMP.DAT",    "float"   }/* 6 */
{ "OUTPUT FILE NAME",      "TMP.OUT",    "float"   }/* 7 */
  };
static  int    count = sizeof( packets )/sizeof( IntrfcPkt );

static char *doc = "Test Test Test\n";

void    toplevel()
  {
extern  double  atof();
        float   *data;
        double  Fs            = atof( packets[0].value );
        int     wnd_len       = atof( packets[1].value )*Fs;
        char    *window             = (char *)packets[2].value;
        int     fft_len       = atoi( packets[3].value );
        int     shift         = atof( packets[4].value )*Fs;
        double  emp           = atof( packets[5].value );
        char    *inputFile    = (char *)packets[6].value;
        char    *inputType    = (char *)packets[6].unit;
        char    *outputFile   = (char *)packets[7].value;
        long    length;
        FILE    *fp, *fopenp();

        /* check window length and dft length */
        if ( wnd_len > fft_len )
          {     errorMsg( "WINDOW LENGTH must be less than FFT LENGTH" );
                exit( -1 );
          }

        /* read the data file and set the data */
        data = (float *)convertRead( inputFile, inputType, "float", &length );
        if ( length < 0 )
                exit( -1 );

        /* open output stream */
        if ( (fp = fopenp( outputFile, "w" )) == NULL )
          {     errorMsg( "Can't open the file ( %s )", outputFile );
                exit( -1 );
          }

      /* do something .... */

        /* done */
        fclose( fp );
        free( (char *)data );
  }

void    main( argc, argv )
int     argc;
char    *argv[];
```

```
{
    CommandInit( argc, argv, packets, count, doc );
      toplevel( );
    exit( 0 );
}
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

cx_zero, cx_add, cx_sub, cx_mul, cx_div, cx_conj, cx_exp, cx_pow, cx_root, cx_tocomplex, cx_abs, cx_arg, cx_print – complex function utilities.

SYNOPSIS

```
# include         <complex.h>

complex cx_zero( )

complex cx_add( a, b )
complex a, b;

complex cx_sub( a, b )
complex a, b;

complex cx_mul( a, b )
complex a, b;

complex cx_div( a, b )
complex a, b;

complex cx_conj( a )
complex a;

complex cx_exp( a )
complex a;

complex cx_pow( a, n )
complex a;
double  n;

complex cx_root( a, n, m )
complex a;
int     n, m;

complex cx_tocomplex( re, im )
double  re, im;

double  cx_abs( a )
complex a;

double  cx_arg( a )
complex a;

void    cx_print( a )
complex a;
```

DESCRIPTION

These functions operate on the **complex** structure, defined in <complex.h> as:

```
typedef struct
   {     double  re, im;
   }     complex;
```

EXAMPLE

```
# include        <stdio.h>
# include        <complex.h>

int Mandelbrot( x, y, count )
double  x, y;
int     count;
  {
        complex z;

        z = cx_tocomplex( x, y );
      while( (cx_abs( z ) < 2.0) && (count > 0) )
          {     z = cx_sub( cx_mul( z, z ), cx_tocomplex( x, y ) );
                count--;
          }
      return( count );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

convertArray, convertRead, convertWrite – convert data type of array.

SYNOPSIS

```
void    convertArray( input, output, input_type, output_type, length )
char    *input;
char    *output;
char    *input_type, *output_type;
long    length;

char    *convertRead( filename, input_type, output_type, length )
char    *filename;
char    *input_type;
char    *output_type;
long    *length;

int     convertWrite( filename, input_type, output_type, length, p )
char    *filename;
char    *input_type;
char    *output_type;
long    length;
char    *p;
```

DESCRIPTION

convertArray() copies an *input* array to an *output* array casting from an *input_type* to an *output_type*. Since **convertArray()** does not check whether the *input* data is proper for an *output_type* or not, **out-of-range** data cause an error.

convertRead() reads the data named by *filename* using **convertArray()** and returns the pointer to the read data, and sets the *length* which is the size bytes normalized by **sizeof** *output_type*. **convertRead()** obtains file size bytes of space using **malloc()**. If there is an error, **convertRead()** returns a **NULL** pointer and *length* is set to -1.

convertWrite() writes the data to the file named by *filename* using **convertArray()** and returns the number of written data. If there is an error, **convertWrite()** returns 0.

When error occurs in **convertRead()** and **convertWrite()**, error messages are sent to the standard error (*stderr*).

NOTE

input_type and *output_type* allow the following types:

- char
- short
- int
- float
- double

BUGS

In order to obtain temporary memory space, **convertRead()** and **convertWrite()** call **alloca() in** *libcommand.a.* When **alloca()** fails to obtain temporary memory space, display warning messages on the standard error (*stderr*) and retries to obtain temporary memory space by using **malloc()**. Again **malloc()** fails, display error messages on the standard error (*stderr*) and returns.

WARNING

convertRead() and **convertWrite()** display warning messages on the standard error (*stderr*). The warning messages can be disabled using **warningMsgSet()**.

SEE ALSO

getabspath(3), message(3), malloc( ) in *libc.a*

EXAMPLE

```
# include       <stdio.h>

void    main( argc, argv )
int     argc;
char    *argv[ ];
   {
        float   *data;
        char    *convertRead( );
        int     convertWrite( );
        char    *inputfile  = argv[1];
        char    *outputfile = argv[2];

        /* read short integer binary data from inputfile */
        data = (float *)convertRead( inputfile, "short", "float", &length );
        if ( length < 0 )
                exit( -1 );

        /* do smoothing */
        ...............

        /* store short integer binary data to outputfile */
        convertWrite( outputfile, "float", "short", length, data );
        free( (char *)data );
        exit( 0 );
   }
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

cor_alp_float, cor_alp_double – conversion of autocorrelation coefficients into linear prediction parameters.

SYNOPSIS

```
double  cor_alp_float( COR, ALP, P )
float   *COR;
float   *ALP;
int     P;

double  cor_alp_double( COR, ALP, P )
double  *COR;
double  *ALP;
int     P;
```

DESCRIPTION

cor_alp_float() and **cor_alp_double()** convert autocorrelation coefficients into linear prediction parameters using Durbin's recursive algorithm for solving the Toeplitz matrix equation, and return the value of residual error.

ARGUMENT

COR	autocorrelation coefficients, size of array is P + 1. The value of *COR[0]* is always 1.
ALP	linear prediction parameters, size of array is P + 1. The value of *ALP[0]* is always 1.
P	order of linear prediction coefficients.

NOTE

The transfer function H(z) is related to the linear prediction coefficients α_i by the equation:

$$H(z) = \frac{\sigma^2}{1+\sum_{i=1}^{p}\alpha_i z^{-i}}$$

In order to get the linear prediction coefficients α_i from the autocorrelation coefficients ϕ_i , the toeplitz matrix equation :

$$\sum_{i=0}^{p}\alpha_i \phi_{|i-j|} = 0 \quad \left[j = 1,2, \cdots ,p \right]$$

must be solved. However, the Toeplitz matrix equation can be solved by Durbin's recursive algorithm. The ϕ_i are autocorrelation coefficients obtainded by **sig_cor()**.

SEE ALSO

alp_cep(3), alp_par(3), par_alp(3), sig_cor(3)

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

correlation_float, correlation_double – calculate correlation coefficients.

SYNOPSIS

```
double  correlation_float( XX, YY, COR, P, N )
float   *XX, *YY;
float   *COR;
int     P, N;

double  correlation_double( XX, YY, COR, P, N )
double  *XX, *YY;
double  *COR;
int     P, N;
```

DESCRIPTION

correlation_float() and **correlation_double()** calculate correlation coefficients of two vectors *XX* and *YY* normalized as 1.0 and return an energy. Autocorrelation coefficients is obtainded, when *XX* is equal to *YY*.

ARGUMENT

XX	input vector, size of vector is N.
YY	input vector, size of vector is N.
COR	output vector, size of vector is P. The value of *COR[0]* is always 1.
P	order of correlation
N	length of input vectors

NOTE

$$COR[i] = \frac{1}{ENG} \sum_{k=0}^{N-i-1} XX[k] \times YY[k+i] \quad (i = 1,2, \cdots N-1)$$

where,

$$ENG = \sum_{k=0}^{N-1} XX[k] \times YY[k]$$

EXAMPLE

```
# include       <stdio.h>

void    main()
  {
extern  double  correlation_float();
static  float   a[3], b[3];
        double  eng;

        a[0] = 1.0;
        a[1] = 2.0;
        a[2] = 3.0;

        eng = correlation_float( a, a, b, 3, 3 );
        printf( "energy = %f\n", eng );
  }
```

LIBRARY

libst.a

AUTHOR
Seiichi TENPAKU

NAME

countLine, countChar, countWord – calculate a number of lines, words and characters in file.

SYNOPSIS

```
# include       <stdio.h>

int     countLine( fp )
FILE    *fp;

int     countWord( fp )
FILE    *fp;

int     countChar( fp )
FILE    *fp;
```

DESCRIPTION

countLine(), **countWord()** and **countChar()** return the count of lines, words and characters of the file pointer *fp*, respectively.

BUGS

In order to reset the file pointer *fp* after counting, **countLine()**, **countWord()** and **countChar()** automatically call **fseek(fp, 0L,** SEEK_SET) by themselves. Therefore, you do not need to reset the file pointer.

SEE ALSO

fseek( ) in *libc.a*

EXAMPLE

```
# include       <stdio.h>

void    main( argc, argv )
int     argc;
char    *argv[ ];
   {
        char    *filename = argv[1];
        FILE    *fp, *fopen( );

        if ( (fp = fopen( filename, "r" )) != NULL )
           {    l = countLine( fp );
                w = countWord( fp );
                c = countChar( fp );
                printf( "lines = %d, words = %d, chars = %d\n", l, w, c );
                fclose( fp );
           }
   }
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

critical_damp – second order critically damped model.

SYNOPSIS

```
double  critical_damp( t, y0, b, R )
double  t, y0, b, R;
```

DESCRIPTION

critical_damp() returns the value of a second order critically damped model Y:

Y = a(1 + Lt)exp(-Lt) + b

ARGUMENT

t	parameters of the time
y0	the value at t = 0 (= a + b)
b	the value at t = infinity
R	the time constant (= 1/L)

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

differential_float, differential_double, differential_short, differential_int – calculate a the differential of an array.

SYNOPSIS

```
void    differential_float( input, output, A, N )
float   *input;
float   *output;
double  A;
int     N;

void    differential_double( input, output, A, N )
double  *input;
double  *output;
double  A;
int     N;

void    differential_short( input, output, A, N )
short   *input;
short   *output;
double  A;
int     N;

void    differential_int( input, output, A, N )
int     *input;
int     *output;
double  A;
int     N;
```

DESCRIPTION

differential_float(), differential_double(), differential_short() and **differential_int()** calculate a differential.

ARGUMENT

input	input vector, size of vector is N.
output	output vector, size of vector is N.
A	multiply factor.
N	length of input and output vector.

NOTE

$output[i] = input[i] - A \times input[i-1] \quad (i = 1,2, \cdots N-1)$
$output[0] = input[0]$

EXAMPLE

```
# include       <stdio.h>

void    main()
  {
static  float   a[3], b[3];
        int     i;

        a[0] = 1.0;
        a[1] = 2.0;
        a[2] = 3.0;
```

```
        differential_float( a, b, 1.0, 3 );
        for ( i = 0 ; i < 3 ; i++ )
                printf( "b[%d] = %f\n", i, b[i] );
}
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

aberth_init, dka_method, dka_solve – solving a high order polynomial expression.

SYNOPSIS

```
# include       <complex.h>

void    aberth_init( n, a, x )
int     n;
double  *a;
complex *x;

int     dka_method( n, a, rp, ip )
int     n;
double  *a;
double  *rp, *ip;

int     dka_solve( n, a, x, y )
int     n;
double  *a;
complex *x, *y;
```

DESCRIPTION

These functions uses solve a high order polynomial expression with the **DKA** (**Durand Kerner Aberth**) method. In general, a high order polynomial expression P(x) is defined as:

$$
\begin{aligned}
P(x) &= a_0x^n + a_1x^{n-1} + \dots + a_n \\
&= a_0(x - \alpha_1)(x - \alpha_2) \dots (x - \alpha_n),\ a_0 \neq 0.
\end{aligned}
$$

where α_1, α_2, ... α_n are roots of $P(x) = 0$.

aberth_init() calculates Aberth's Initial roots;

$$x[i] = r \times \exp\{ J(2\pi i/n + \pi/2n) \}$$
$$r = n \times \max\{ |a[i]/a[0]|^{1/i} \}$$
$$(i = 1, 2, \dots , n-1)$$

where **J** is an imaginary number unit ($\sqrt{-1}$).

aberth_init() is called by **dka_method()** and **dka_solve()**.

Both **dka_method()** and **dka_solve()** resolve a high order polynomial expression with the **DKA** method. There is no difference between the two functions, except argument and memory allocation strategy.

ARGUMENT

dka_method()

n	order of the polynomial expression
*a	$a[0] \times x^n + a[1] \times x^{n-1} + \dots + a[n]$, a[0] = 1.0 implicitly !
*rp	real parts of the estimated roots, rp[0],...,rp[n-1]
*ip	imaginary parts of the estimated roots, ip[0],...,ip[n-1]

dka_solve()

n	order of the polynomial expression
*a	$a[0] \times x^n + a[1] \times x^{n-1} + \dots + a[n]$, a[0] = 1.0 implicitly !

*x	for working array, x[0],...,x[n-1]
*y	the estimated roots, y[0],...,y[n-1]

RETURN VALUE

TRUE if there is no error

FALSE if there is an error

BUGS

dka_method() uses **alloca()** in *libcommand.a.*

EXAMPLE

```
# include       <stdio.h>
# include       <complex.h>

main()
  {
      int     i;
static  double  a[11];
static  double  rp[10], ip[10];
static  complex x[10], y[10];
      int     n;

      n = 4;
      a[0] = 1.0;
      a[1] = 1.0;
      a[2] = 1.0;
      a[3] = 1.0;
      a[4] = 1.0;

      dka_method( n, a, rp, ip );
      for ( i = 0 ; i < n ; i++ )
            printf( "%f\t%f\n", rp[i], ip[i] );

      dka_solve( n, a, x, y );
      for ( i = 0 ; i < n ; i++ )
            printf( "%f\t%f\n", x[i].re, x[i].im );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

dot_float, dot_double, dot_short, dot_int – calculate an inner product.

SYNOPSIS

```
double  dot_float( XX, YY, n )
float   *XX;
float   *YY;
int     n;

double  dot_double( XX, YY, n )
double  *XX;
double  *YY;
int     n;

double  dot_short( XX, YY, n )
short   *XX;
short   *YY;
int     n;

double  dot_int( XX, YY, n )
int     *XX;
int     *YY;
int     n;
```

DESCRIPTION

dot_float(), **dot_double()**, **dot_short()** and **dot_int()** return an inner product of two vectors *XX* and *YY* :

$$XX[0]\times YY[0] + XX[1]\times YY[1] + \cdots + XX[n-1]\times YY[n-1] = \sum_{i=0}^{n-1} XX[i]\times YY[i]$$

EXAMPLE

```
# include       <stdio.h>

void    main()
  {
extern  double  dot_float();
static  float   a[3], b[3];
        double  dot;

        a[0] = 1.0;
        a[1] = 2.0;
        a[2] = 3.0;

        b[0] = 1.0;
        b[1] = 1.0;
        b[2] = 1.0;

        dot = dot_float( a, b, 3 );
        printf( "dot = %f\n", dot );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

fft, ifft – discrete Fourier transform.

SYNOPSIS

```
int     fft( Re, Im, n )
double  *Re, *Im;
int     n;

int     ifft( Re, Im, n )
double  *Re, *Im;
int     n;
```

DESCRIPTION

fft( ) calculates a discrete Fourier transform.

ifft( ) calculates an inverse discrete Fourier transform.

ARGUMENT

Re	real part
Im	imaginary part
n	number of DFT point; 2^p (p is an integer number)

RETURN VALUE

TRUE if there is no error

FALSE if there is an error

BUGS

Since both **fft**() and **ifft**() use *malloc( )* in order to make a sine-cosine table, when *malloc( )* causes an error they return *FALSE*.

NOTE

fft( ) and **ifft**( ) use a Sande-Tukey algorithm to calculate a discrete Fourier transform function.

The discrete Fourier transform is defined as :

$$X(n) = \sum_{k=0}^{N-1} x(k) W_N^{nk} \quad (n = 0, \pm 1, \pm 2, \cdots)$$

where $W_N = exp(-j \cdot 2\pi / N)$

The inverse discrete Fourier transform is defined as :

$$x(k) = \frac{1}{N} \sum_{n=0}^{N-1} X(n) W_N^{-kn}$$

SEE ALSO

malloc( )

EXAMPLE

```
# include       <stdio.h>

main( )
   {
static  double  Re[256], Im[256];
        int     i, n;
        n = 256;
        for ( i = 0 ; i < n ; i++ )
           {    Re[i] = (double)i;
                Im[i] = 0.0;
           }
        fft( Re, Im, n );
   }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

fopenp, get_file_length, openRead, openWrite, binRead, binStore – file utilities of SpeechTools.

SYNOPSIS

```
# include       <stdio.h>
FILE    *fopenp(filename, type)
char    *filename, *type;

int     get_file_length( filename, &length, size )
char    *filename;
long    length;
int     size;

int     openRead( filename, &fd, &byte_length )
char    *filename;
int     fd;
long    byte_length;

int     openWrite( filename, &fd )
char    *filename;
int     fd;

char    *binRead( filename, size, &length )
char    *filename;
int     size;
long    length;

int     binStore( filename, size, length, array )
char    *filename;
int     size;
long    length;
char    *array;
```

DESCRIPTION

These SpeechTools file utility functions are implemented by using **getabspath**(). Therefore, *filename* relative to current working directory or user's home directory are allowed ('~', '.', '..').

fopenp() opens the file named by *filename* and associates a stream with it by using **getabspath**(). If the open succeeds, **fopenp**() returns a pointer to be used to identify the stream in subsequent operations.

filename
: points to a character string that contains the name of the file to be opened.

type is a character string having one of the following values:

- **r** open for reading
- **w** truncate or create for writing
- **a** append: open for writing at end of file, or create for writing
- **r+** open for update (reading and writing)
- **w+** truncate or create for update
- **a+** append; open or create for update at EOF

get_file_length() calculates the file length divided by *size*. **get_file_length**() returns *TRUE* = 1 when the file exits and the file can be accessed, or *FALSE* = 0 when there is an error.

openRead() opens for reading the file named by *filename*. When the open succeeds, **openRead()** returns *TRUE* = 1 and sets *fd* to a descriptor for that file and *byte_length* to a size in bytes of that file. If there is an error, **openRead()** returns *FALSE* = 0 and sets *byte_length* to -1.

openWrite() opens for writing the file named by *filename*. **openWrite()** returns *TRUE* = 1 and sets *fd* to a descriptor for that file. If there is an error, **openWrite()** returns *FALSE* = 0.

binRead() reads the binary data associated by *filename* and returns the pointer to read data, and sets the *length* which is the size in bytes normalized by *size*. If there is an error, **binRead()** returns a **NULL** pointer.

binStore() writes the binary data of *array* to the file named by *filename* and returns the number of written data. If there is an error, **binStore()** returns negative number.

BUGS

In order to obtain memory space to read data, **binRead()** calls **malloc()**.

SEE ALSO

getabspath()

EXAMPLE

```
# include       <stdio.h>

void    main( argc, argv )
int     argc;
char    *argv[];
  {
extern  char    *binRead();
extern  int     binStore();
        long    length;
        short   *data;
        char    *inputfile  = argv[1];
        char    *outputfile = argv[2];

        /* read binary data */
        data = (short *)binRead( input, sizeof(short), &length );
        if ( length <= 0 )
                exit( -1 );

        /* do something */
        ...............

        /* store binary data */
        length = binStore( output, sizeof(short), length, data );
        if ( length <= 0 )
                exit( -1 );
        free( (char *)data );
        exit( 0 );
  }
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

filter_lpf1, filter_hpf1, filter_dif1 – IIR filter function utilities.

SYNOPSIS

```
void    filter_lpf1( Fs, Fc, D, Xin, &Xout )
double  Fs, Fc, D[1], Xin, Xout;

void    filter_hpf1( Fs, Fc, D, Xin, Xout )
double  Fs, Fc, D[1], Xin, Xout;

void    filter_dif1( Fs, AL, D, Xin, Xout )
double  Fs, AL, D[1], Xin, Xout;
```

DESCRIPTION

filter_lpf1( ) execute a first order IIR low-pass filter.

$$H(z) = \frac{1-b_1}{2} \cdot \frac{1+z^{-1}}{1-b_1 z^{-1}} \left(0<b_1<1\right)$$

$$b_1 = \frac{\varepsilon-1}{\varepsilon+1}$$

$$\varepsilon = tan\,(2Fc\,\pi/Fs)$$

filter_hpf1( ) execute a first order IIR high-pass filter.

$$H(z) = \frac{1-b_1}{2} \cdot \frac{1-z^{-1}}{1+b_1 z^{-1}} \left(0<b_1<1\right)$$

$$b_1 = \frac{\varepsilon-1}{\varepsilon+1}$$

$$\varepsilon = tan\,(2Fc\,\pi/Fs)$$

filter_diff1( ) execute a first order differential filter.

$$H(z) = 1-ALz^{-1}$$

ARGUMENT

Fs	sampling frequency [Hz].
Fc	cut-off frequency [Hz] (Fc <= Fs/4).
AL	filter parameter.
D[1]	memory for delay.
Xin	input signal.
Xout	output signal.

EXAMPLE

```
# include     <stdio.h>

void    do_filtering( input, output, length, Fs, Fc )
float   *input, *output;
int     length;
double  Fs, Fc;
  {
static  double  D[1];
        double  Xin, Xout;
      int     i;

        D[0] = 0.0;      /* initialize for delay */
```

```
        for ( i = 0 ; i < length ; i++ )
          {     Xin = input[i];
                filter_lpf1( Fs, Fc, D, Xin, &Xout );
                output[i] = Xout;
          }
}
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

find_float_zero, find_float_peak, find_float_max, find_float_min, find_float_absmax, find_float_absmin, find_double_zero, find_double_peak, find_double_max, find_double_min, find_double_absmax, find_double_absmin, find_short_zero, find_short_peak, find_short_max, find_short_min, find_short_absmax, find_short_absmin – finding utilities.

SYNOPSIS

```
# include        <find.h>

int     find_float_zero( array, start, end, &point )
float   *array;
int     start, end, point;

int     find_float_peak( mode, array, start, end, &point )
int     mode;   /* 1, 2: maximumpeak, -1, -2: minimum peak */
float   *array;
int     start, end, point;

int     find_float_max( n, array, &value )
int     n;
float   *array, value;

int     find_float_min( n, array, &value )
int     n;
float   *array, value;

int     find_float_absmax( n, array, &value )
int     n;
float   *array, value;

int     find_float_absmin( n, array, &value )
int     n;
float   *array, value;

int     find_double_zero( array, start, end, &point )
double  *array;
int     start, end, point;

int     find_double_peak( mode, array, start, end, &point )
int     mode;   /* 1, 2: maximumpeak, -1, -2: minimum peak */
double  *array;
int     start, end, point;

int     find_double_max( n, array, &value )
int     n;
double  *array, value;

int     find_double_min( n, array, &value )
int     n;
double  *array, value;

int     find_double_absmax( n, array, &value )
int     n;
double  *array, value;
```

```
int     find_double_absmin( n, array, &value )
int     n;
double  *array, value;

int     find_short_zero( array, start, end, &point )
short   *array;
int     start, end, point;

int     find_short_peak( mode, array, start, end, &point )
int     mode;   /* 1, 2: maximumpeak, -1, -2: minimum peak */
short   *array;
int     start, end, point;

int     find_short_max( n, array, &value )
int     n;
short   *array, value;

int     find_short_min( n, array, &value )
int     n;
short   *array, value;

int     find_short_absmax( n, array, &value )
int     n;
short   *array, value;

int     find_short_absmin( n, array, &value )
int     n;
short   *array, value;
```

DESCRIPTION

find_float_zero(), **find_double_zero()** and **find_short_zero()** find the first zero crossing point in between *start* and *end* of the *array* and return 1, or return 0 if there is no zero crossing point in between *start* and *end* of the *array*. When *mode* is positive number, maximum peak is searched. When *mode* is negative number, minimum peak is searched.

find_float_peak(), **find_double_peak()** and **find_short_peak()** find the first peak point in between *start* and *end* of the *array* and return 1, or return 0 if there is no peak point in between *start* and *end* of the *array*.

find_float_max(), **find_double_max()** and **find_short_max()** find a maximum value and return the found position.

find_float_min(), **find_double_min()** and **find_short_min()** find a minimum value and return the found position.

find_float_absmax(), **find_double_absmax()** and **find_short_absmax()** find a absolute maximum value and return the found position.

find_float_absmin(), **find_double_absmin()** and **find_short_absmin()** find a absolute minimum value and return the found position.

EXAMPLE

```
# include     <stdio.h>
# include        <find.h>

int     zerocount( array, length )
float   *array;
```

```
int     length;
  {
        int     start, end, count;

        start = 0;
        end   = length - 1;
        count = 0;
        while( start < end )
          {     if ( find_float_zero( array, start, end, &start ) )
                  {     count++;
                        start++;
                  }
                else
                        break;
          }
        return( count );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

gain_control_float, gain_control_double, gain_control_short – amplify the value of array.

SYNOPSIS

```
void    gain_control_float( gain, length, array )
double  gain;
long    length;
float   *array;

void    gain_control_double( gain, length, array )
double  gain;
long    length;
double  *array;

void    gain_control_short( gain, length, array )
double  gain;
long    length;
short   *array;
```

DESCRIPTION

gain_control_float( ), **gain_control_double( )** and **gain_control_short( )** amplify the value of array.

ARGUMENT

gain	maximum gain [dB].
length	length of data.
array	data array, size of array is length.

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME
getabspath – return the absolute pathname.

SYNOPSIS
```
char    *getabspath( file_name )
char    *file_name;
```

DESCRIPTION
getabspath() resolves all links and all references to '~', '.' and '..' in *file_name* and returns a pointer to the result. **getabspath**() does not check whether files exist or not. This function is implemented by using strlen(), getpwd() and gethome(). The realpath() function is very similar to getabspath().

RETURN VALUE
If links which are resolved reache to the root ('/'), then **getabspath()** returns '/'. **getabspath()** returns a **NULL** pointer, when *file_name* is **NULL**, or when some error is caused by **getpwd**(), **gethome**() or somewhat. If there is no error, **getabspath()** returns the absolute pathname of the *file_name*.

SEE ALSO
getcwd(), getwd()

BUGS
getabspath() operates only on **NULL**-terminated strings. Maximum length of the *file_name* is fixed at 128. **getabspath**() does not check for overflow of the receiving string.

EXAMPLE
```
# include       <stdio.h>

void    main( argc, argv )
int     argc;
char    *argv[];
   {
        char    *name = argv[1];
        char    *path;

        if ( (path = getabspath( name )) != NULL )
                printf( "%s == %s\n", name, path );
        else
                printf( "getabspath() error :: %s \n", name );
   }
```

LIBRARY
libcommand.a

AUTHOR
Seiichi TENPAKU

NAME

gethome – get pathname of user's home directory.

SYNOPSIS

```
char    *gethome( name )
char    *name;
```

DESCRIPTION

gethome() returns a pointer to home directory pathname of user *name*. If *name* is a **NULL** pointer, **gethome()** returns home directory pathname of login user. When *name* is not found in **/etc/passwd**, **gethome()** returns a **NULL** pointer. This function is implemented by using **getenv()** and **getpwnam()**.

SEE ALSO

getenv(), getpwnam() in *libc.a*

FILES

/etc/passwd

EXAMPLE

```
# include         <stdio.h>

void    main()
  {
        char    *home, *gethome();
        char    *name = "tenpaku";

        if ( (home = gethome( name )) == NULL )
                printf( "%s is an unknown user\n", name );
        else
                printf( "%s's home directory is %s\n", name, home );
  }
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

getpwd – get the current working directory pathname.

SYNOPSIS

```
char    *getpwd( name, length )
char    *name;
int     length;
```

DESCRIPTION

getpwd() copies the absolute pathname of the current working directory to *name* and returns a pointer to the result. **getpwd()** returns a NULL pointer if an error occurs.

SEE ALSO

getcwd(), getwd() in *libc.a*

EXAMPLE

```
# include       <stdio.h>

void    main()
   {
        char    name[80];
        char    *p, *getpwd();

        if ( (p = getpwd( name, sizeof(name) )) == NULL )
                printf( "CWD = %s\n", name );
        else
                printf( "ERROR\n");
   }
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

IntrfcLineGet, IntrfcFindUnit, IntrfcPktFind, IntrfcPktAset, IntrfcPktRead, IntrfcPktWrite, IntrfcFileRead, IntrfcFileWrite, IntrfcPktStr, IntrfcStrPkt, IntrfcPktGet, IntrfcPktSet, – command interface utilities of SpeechTools.

SYNOPSIS

```
# include      <interface.h>

char           *IntrfcLineGet( fp, buffer, n )
FILE           *fp;
char           *buffer;
int            n;

int            IntrfcFindUnit( str, key )
char           *str, *key;

int            IntrfcPktFind( packets, count, str )
IntrfcPkt      *packets;
int            count;
char           *item;

int            IntrfcPktAset( packets, count, line )
IntrfcPkt      *packets;
int            count;
char           *line;

int            IntrfcPktRead( fp, packets, count )
FILE           *fp;
IntrfcPkt      *packets;
int            count;

void           IntrfcPktWrite( fp, packets, count )
FILE           *fp;
IntrfcPkt      *packets;
int            count;

int            IntrfcFileRead( file, packets, count )
char           *file;
IntrfcPkt      *packets;
int            count;

int            IntrfcFileWrite( file, packets, count )
char           *file;
IntrfcPkt      *packets;
int            count;

int            IntrfcPktStr( str, packets, count )
char           *str;
IntrfcPkt      *packets;
int            count;

int            IntrfcStrPkt( str, packets, count )
char           *str;
IntrfcPkt      *packets;
```

```
int             count;

char            *IntrfcPktGet( packets, count, item )
IntrfcPkt       *packets;
int             count;
char            *item;

char            *IntrfcPktSet( packets, count, item, value )
IntrfcPkt       *packets;
int             count;
char            *item;
char            *value;
```

DESCRIPTION

These functions operate or are concerned with a **IntrfcPkt** structure, defined in <interface.h> as:

```
# define MAX_STRING            128

typedef
struct  packet
  {  char   *item;              /* full name of parameter       */
     char   value[MAX_STRING];              /* value of parameter            */
     char   *unit;              /* unit of parameter          */
     char   *comment;           /* comment field of parameter   */
  }  IntrfcPkt;
```

IntrfcLineGet( ) reads characters from the input stream associated by *fp* into the array pointed to by *buffer* while converting a **TAB** character into a **SPACE** character, until a **NEWLINE** character is read or an **EOF** condition is encountered. The **NEWLINE** character is discarded and the string is terminated with a null character. **IntrfcLineGet**( ) returns a pointer to the *buffer* or returns a **NULL** pointer when the input stream reaches at the **EOF** condition. Note that the comment field is introduced by a '#' sign and finish at the end of the line, so that the comment field is ignored. Therefore, **IntrfcLineGet**( ) is very useful to read ascii data and to do something line by line, for example:

```
# include         <stdio.h>
# include         <interface.h>

main( argc, argv )
int             argc;
char            *argv[];
  {
                FILE            *fp, *fopen();
                char            buffer[80];

                fp = fopen( argv[1], "r" );
                while ( IntrfcLineGet( fp, buffer, sizeof(buffer) ) )
                                printf( "%s\n", buffer );
  }
```

IntrfcPktFind( ) **IntrfcFindUnit**( ) and **IntrfcPktAset**( ) are utility functions to treat **IntrfcPkt** structure. They are might be used in internal.

IntrfcFindUnit( )
: searches the unit named by *key* in the string named by *str*. If the search fails, **IntrfcFindUnit**( ) returns -1.

IntrfcPktFind( )
: compares the string *str* of argument to the string *item* field in *packets*, returns a number to the

first occurrence of parameter *str* in the array of *packets*, or -1 if *str* does not occur in the *packets*.

IntrfcPktAset( )
changes the parameter in *packets* and returns *TRUE* = 1, if the item of *line* is same as that of the parameter. In order to find the item of *line* in *packets* **IntrfcPktAset( )** calls **IntrfcPktFind( )**. When *line* dose not find in the *packets*, **IntrfcPktAset( )** displays warning messages by using **warningMsg( )**. If *unit* in the *packets* is missing, **IntrfcPktAset( )** displays warning messages by using **warningMsg( )** and changes the *unit* field.

IntrfcPktRead( ) and **IntrfcPktRead( )** read or write *packets* structure from or to a file pointer *fp*.

IntrfcPktRead( )
reads the *packets* from stream *fp*. **IntrfcPktRead( )** returns the number of read *packets*, or -1 if there is an error.

IntrfcPktWrite( )
writes the *packets* to stream *fp*.

IntrfcFileRead( ) and **IntrfcFileWrite( )** read or write *packets* structure from or to a file by using **IntrfcPktRead( )** and **IntrfcPktRead( )**, respectively.

IntrfcFileRead( )
reads the *packets* from the file named by *file* and returns *TRUE* = 1, or returns *FALSE* = 0, if there is an error.

IntrfcFileWrite( )
writes the *packets* to the file named by *file* and returns *TRUE* = 1, or returns *FALSE* = 0, if there is an error.

IntrfcPktStr( ) and **IntrfcStrPkt( )** convert between *packets* structure to *str* string.

IntrfcPktStr( )
converts the *packets* to the string array associated by *str* and returns the length of *str*, or -1 if there is an error.

IntrfcStrPkt( )
converts the string array associated by *str* to the *packets* and returns the count of *packets*, or -1 if there is an error.

IntrfcPktGet( ) and **IntrfcPktSet( )** get or set *packets* structure by using **IntrfcPktFind( )**.

IntrfcPktGet( )
retunrs the *value* field in *packets* when *item* is found in *packets*, or retunrs a **NULL** pointer when *item* is not found in *packets*.

IntrfcPktSet( )
changes the *value* field of *packets* to the *value* of argument and retunrs the changed *value* field in *packets* when *item* is found in *packets*, or retunrs a **NULL** pointer when *item* is not found in *packets*.

SEE ALSO
SpeechTools(1), message(3)

LIBRARY
libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

interpol_gettype, interpol_linear, interpol_lagrange, interpol_blend_init, interpol_blend, interpol_spline_init, interpol_spline – interpolate function utilities.

SYNOPSIS

```
# include      <interpol.h>

int     interpol_gettype( name )
char    *name;

double  interpol_linear( N, Sx, Sy, X )
int     N;
double  *Sx, *Sy;
double  X;

double  interpol_lagrange( N, Sx, Sy, X )
int     N;
double  *Sx, *Sy;
double  X;

int     interpol_blend_init( N, Sx, Sy, Table )
int     N;
double  *Sx, *Sy;
double  *Table;

double  interpol_blend( N, Sx, Sy, Table, X )
int     N;
double  *Sx, *Sy;
double  *Table;
double  X;

int     interpol_spline_init( N, Sx, Sy, Table )
int     N;
double  *Sx, *Sy;
double  *Table;

double  interpol_spline( N, Sx, Sy, Table, X )
int     N;
double  *Sx, *Sy;
double  *Table;
double  X;
```

DESCRIPTION

These functions are concerning with interpolation of sampled data. **interpol_linear( )**, **interpol_lagrange( )**, **interpol_blend( )** and **interpol_spline( )** get the value at *X* from sampled data *Sx* and *Sy* using interpolation method. **interpol_blend( )** and **interpol_spline( )**, **interpol_blend_init( )** and **interpol_spline_init( )** must be called respectively to make *Table*.

interpol_blend_init( ) and **interpol_spline_init( )** make *Table*.

interpol_gettype( ) returns the identifier of interpolate function defined in <interpol.h> as:

```
# define INTERPOL_UNKNOWN      0
# define INTERPOL_LINEAR       1
# define INTERPOL_LAGRANGE     2
```

```
# define INTERPOL_BLEND         3
# define INTERPOL_SPLINE        4
```

ARGUMENT

name	string of interpolate function as follows: LINEAR LAGRANGE BLEND SPLINE
N	number of samples
Sx	sample points.
Sy	sample data.
Table	table for interpolation
X	interpolated point value

BUGS

interpol_blend_init( ) and **interpol_spline_init**( ) use **alloca**( ) in *libcommand.a.*

EXAMPLE

```
# include      <stdio.h>
# include      <interpol.h>

# define PI    (double)3.14159265
extern  double sin()

main()
  {
static  double  x[10], y[10];
static  double  Table1[10], Table2[10];
        double  XX, YY, Y1, Y2, Y3, Y4;
        int     i, N;

        N = 10;
        for ( i = 0 ; i < N ; i++ )
           {    x[i] = 0.1*PI*(double)(i+1);
                y[i] = sin( x[i] );
           }
        interpol_blend_init( N, x, y, Table1 );
        interpol_spline_init( N, x, y, Table2 );

        for ( i = 0 ; i < N ; i++ )
           {    XX = 1.0 + (double)i*0.1;
                Y1 = interpol_blend( N, x, y, Table1, XX );
                Y2 = interpol_spline( N, x, y, Table2, XX );
                Y3 = interpol_linear( N, x, y, XX );
                Y4 = interpol_lagrange( N, x, y, XX );
                YY = sin( XX );
                printf( "XX = %f, YY = %f ", XX, YY );
                printf( "blend = %f ", Y1 );
                printf( "spline = %f ", Y2 );
                printf( "linear = %f ", Y3 );
                printf( "lagrange = %f ", Y4 );
                printf( "\n" );
```

```
                }
        }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

lagwin_float, lagwin_double – calculate a binomial window coefficients.

SYNOPSIS

```
void    lagwin_float( window, h, n )
float   *window;
double  h;
int     n;

void    lagwin_double( window, h, n )
double  *window;
double  h;
int     n;
```

DESCRIPTION

lagwin_float() and **lagwin_double()** calculate binomial windowing coefficients that approximate a Gaussian function when *n* is large.

ARGUMENT

window output coefficients, size of array is length. The value of *window[0]* is always 1.

h ratio window half band width to sampling frequency. For example, if lag window half value band width = 100 Hz and sampling frequency = 20 kHz, then *h* = 100/20000 = 1/200 = 0.005.

n window length

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

lip_inverse_float, lip_inverse_double – eliminate lip radiation characteristics.

SYNOPSIS

```
double  lip_inverse_float( SIG, COR, OUT, N )
float   *SIG;
float   *COR;
float   *OUT;
int     N;

double  lip_inverse_double( SIG, COR, OUT, N )
double  *SIG;
double  *COR;
double  *OUT;
int     N;
```

DESCRIPTION

lip_inverse_float() and **lip_inverse_double()** eliminate lip radiation characteristics from speech waveform with adaptive inverse filtering method and return a coefficient of the lip radiation. **lip_inverse_float()** is implemented by using **correlation_float()** and **cor_alp_float()**. **lip_inverse_double()** is implemented by using **correlation_double()** and **cor_alp_double()**.

ARGUMENT

SIG	input speech waveform, size of array is N.
COR	autocorrelation coefficients, size of array is N.
OUT	output speech waveform, size of array is N.
N	length of input waveform data.

BUGS

The value of *COR[0]* is always 1.

SEE ALSO

correlation(3), cor_alp(3),

REFERENCE

P. Alku, E. Vilkman and U. K. Laine (1990) : *A Comparison of EGG and a New Automatic Inverse Filtering Method in Phonation Change from Breathy to Normal*, ICSLP'90, 197-200

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

lsp_calc - calculate LSP (line spectrum pair) coefficients.

SYNOPSIS

```
int     lsp_calc( Fs, n, A, lsp )
double  Fs;
int     n;
double  *A;
double  *lsp;
```

DESCRIPTION

lsp_calc() calculates LSP (line spectrum pair) coefficients by using **DKA** method to solve a polynomial expression.

ARGUMENT

Fs	sampling frequency [Hz].
n	number of poles.
A	parameters of the transmission function or linear prediction parameters, size of array is n + 1. The value of *A[0]* is always 1.
lsp	LSP parameters on frequency domain [Hz], lsp[0],...,lsp[n-1].
TRUE	if there is no error
FALSE	if there is an error

BUGS

lsp_calc() uses **alloca()** in *libcommand.a* and **dka_solve()** in *libst.a*.

SEE ALSO

dka(3)

EXAMPLE

```
# include       <stdio.h>

main()
  {
        int     i;
static  double  a[11];
static  double  lsp[10];
        int     n = 10;

        a[0] = 1.0;
        a[1] = -1.585643;
        a[2] = 0.467723;
        a[3] = 0.827542;
        a[4] = -0.321415;
        a[5] = -0.225748;
        a[6] = -0.573873;
        a[7] = 0.872691;
        a[8] = 0.302009;
        a[9] = -1.062851;
        a[10] = 0.523458;
        lsp_calc( 8000.0, n, a, lsp );
        for ( i = 0 ; i < n ; i++ )
                printf( "%f\n", lsp[i] );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

lstsq_float, lstsq_calc_float, lstsq_double, lstsq_calc_double – least square method.

SYNOPSIS

```
# include       <lstsq.h>

int     lstsq_float( x, y, c, len, m )
float   *x, *y;
float   *c;
int     len;
int     m;

float   lstsq_calc_float( x, c, m )
float   x;
float   *c;
int     m;

int     lstsq_double( x, y, c, len, m )
double  *x, *y;
double  *c;
int     len;
int     m;

double  lstsq_calc_double( x, c, m )
double  x;
double  *c;
int     m;
```

DESCRIPTION

lstsq_float() and **lstsq_double**() calculate coefficients *c* of polynomial equation form sampled data *y* with the least square method. *x* are sample points. **lstsq_float**() and **lstsq_double**() return *TRUE* = 1 when succeeded, or *FALSE* = 0 when failed.

lstsq_calc_float() and **lstsq_calc_double**() return the equation value with Honer method :

f(x) = ((...(x + c[m])x + c[m-1])x + ...) + c[0]

These functions must be called after calling **lstsq_float**() and **lstsq_double**() to make the coefficients *c* of polynomial equation.

ARGUMENT

x	input samples in x-axis
y	input samples in y-axis
c	output coefficients of polynomial equation
len	number of samples
m	order of polynomial equation (<= 20)

SEE ALSO

median(3), smooth(3)

EXAMPLE

```
# include     <stdio.h>
# include     <lstsq.h>

main()
  {
static   float     x[8] = { -3., -2., -1., 0., 1., 2., 3., 4. };
```

```
static  float   y[8] = { -58., -26., -10., -4., -2., 2., 14., 40. };
        float   c[4];
        int     m, n, i;

        n = 8;
        m = 3;

        lstsq_float( x, y, c, n, m );

        for ( i = 0 ; i <= m ; i++ )
                printf( "c[%d] = %f\n", i, c[i] );
    }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

median_float – get the median value fo array.

SYNOPSIS

```
float   median_float( array, work, n )
float   *array, *work;
int     n;
```

DESCRIPTION

median_float( ) returns the median value of *array*, which is the *n*/2-th value in *array*.

ARGUMENT

array input data, size of array is *n*.

work working space,

n length of input *array*.

NOTE

median_float( ) calls **qsort**( ) in C library functions.

SEE ALSO

qsort( ) in *libc.a*

EXAMPLE

```
# include       <stdio.h>

main( )
  {
extern  float   median_float( );
        float   input[100], output[100], work[5];
        int     start, end, i, j;

        /* set up input array */
        ......................

        start = 5/2;
        end   = 100 - 5/2;

        /* copying data at front and back */
        for ( i = 0 ; i < start ; i++ )
                output[i] = input[i];
        for ( i = end ; i < length ; i++ )
                output[i] = input[i];

        /* median smoothing */
        for ( i = start, j = 0 ; i < end ; i++, j++ )
                output[i] = median_float( &input[j], work, 5 );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

debugMsgSet, debugMsg, errorMsgSet, errorMsg, warningMsgSet, warningMsg – message function utilities of SpeechTools.

SYNOPSIS

```
int     debugMsgSet( on_off )
int     on_off;

int     errorMsgSet( on_off )
int     on_off;

int     warningMsgSet( on_off )
int     on_off;

void    debugMsg( format [ , arg... ] )
char    *format;

void    errorMsg( format [ , arg... ] )
char    *format;

void    warningMsg( format [ , arg... ] )
char    *format;
```

DESCRIPTION

These functions operate three types of message facility as:

DEBUG connected to *stdout* (for debugging messages)

ERROR connected to *stderr* (for error messages)

WARNING
connected to *stderr* (for warning messages)

debugMsgSet(), **errorMsgSet()** and **warningMsgSet()** set the internal flag of each message facility and return the old value of the internal flag. If the internal flag of a message facility is set to 0, the display function is disabled.

debugMsg(), **errorMsg()** and **warningMsg()** output messages on the standard output stream *stdout*, the standard error stream *stderr*, and the standard error stream *stderr*, respectively.

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

meterSwitch, meterSet, meterInc, meterEnd – meter function utilities of SpeechTools.

SYNOPSIS

```
void    meterSwitch( on_off )
int     on_off;

void    meterSet()

void    meterInc()

void    meterEnd()
```

DESCRIPTION

These functions operate meter facility, which counts repetition of loop statement such as for-loop or while-loop.

meterSwitch( ) sets the internal flag of meter facility. If the internal flag of meter facility is set to 0, the display function is disabled. The default value of the internal flag is 1.

meterSet( ) sets the meter indicator to 0.

meterInc( ) increments the meter indicator and displays '.' on the standard out (*stdout*).

meterEnd( ) terminates the meter facility and displays the value of meter indicator on the standard out (*stdout*).

EXAMPLE

```
# include       <stdio.h>

void    main()
  {
        int     i, length;

        /* set length here */
        length = ....;

        /* loop */
        meterSet();
        for ( i = 0 ; i < length ; i++ )
           {    meterInc();
                /* do something */
                ................
           }
        meterEnd();
  }
```

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

mx1alloc, mx2alloc, mx3alloc, mx1free, mx2free, mx3free – matrix allocation function utilities.

SYNOPSIS

```
# include        <mx.h>

char    *mx1alloc( n, size )
unsigned int n, size;

char    **mx2alloc( n, m, size )
unsigned int n, m, size;

char    ***mx3alloc( n, m, l, size )
unsigned int n, m, l, size;

void    mx1free( p )
char    *p;

void    mx2free( p )
char    **p;

void    mx3free( p )
char    ***p;
```

DESCRIPTION

These routines provide a memory allocation package for vector and matrix.

mx1alloc() returns a pointer to a block $n \times size$ bytes.

mx2alloc() returns a pointer to a block $n \times m \times size + 4 \times n$ bytes.

mx3alloc() returns a pointer to a block $n \times m \times l \times size + 4 \times n \times m + 4 \times n$ bytes.

mx1free() releases a previously allocated block. Its argument is a pointer to a block previously allocated by **mx1alloc()**.

mx2free() releases a previously allocated block. Its argument is a pointer to a block previously allocated by **mx2alloc()**.

mx3free() releases a previously allocated block. Its argument is a pointer to a block previously allocated by **mx3alloc()**.

BUGS

mx1alloc(), **mx2alloc()** and **mx3alloc()** are implemented by using **malloc()**. **mx1free()**, **mx2free()** and **mx3free()** are implemented by using **free()**.

LIBRARY

libmx.a

AUTHOR

Seiichi TENPAKU

NAME

mx_cholesky – matrix solve function utilities with Cholesky method.

SYNOPSIS

```
# include       <mx.h>

int      mx_cholesky( A, X, C, N )
double   **A;    /* A[N][N] */
double   *X;     /* X[N]    */
double   *C;     /* C[N]    */
int      N;
```

DESCRIPTION

mx_cholesky() solves system of linear equations with Cholesky method returns *TRUE* = 1 when the given system of linear eqautions are solved, or *FALSE* = 0 when there is an error.

EXAMPLE

```
# include       <stdio.h>
# include       <mx.h>

main()
  {
        double  **a, *x, *c, det;
        int     i, j, n;

        n = 3;
        x = (double *)mx1alloc( n, sizeof(double) );
        c = (double *)mx1alloc( n, sizeof(double) );
        a = (double **)mx2alloc( n, n, sizeof(double) );

        a[0][0] = 1.0;    a[0][1] = 2.0;    a[0][2] = 4.0;
        a[1][0] = 2.0;    a[1][1] = 7.0;    a[1][2] = -2.0;
        a[2][0] = 4.0;    a[2][1] = -2.0;   a[2][2] = 8.0;
        x[0] = 10.0;      x[1] = 16.0;      x[2] = 12.0;

        mx_cholesky( a, x, c, n );
        for ( i = 0 ; i < n ; i++ )
          {     for ( j = 0 ; j < n - 1 ; j++ )
                        printf( "a[%d][%d] = %f\t", i, j, a[i][j] );
                printf( "a[%d][%d] = %f\n", i, j, a[i][j] );
          }
        printf( "\n" );
        for ( i = 0 ; i < n ; i++ )
                printf( "x[%d] = %f\t", i, x[i] );
        printf( "\n" );
        for ( i = 0 ; i < n ; i++ )
                printf( "c[%d] = %f\t", i, c[i] );
        printf( "\n" );
  }
```

LIBRARY

libmx.a

AUTHOR

Seiichi TENPAKU

NAME

mx_householder_solve – matrix solve function utilities with Householder method.

SYNOPSIS

```
# include        <mx.h>

int      mx_householder_solve( N, M, A, B, J, P )
int      N, M;
double   **A;    /* A[N][M+1] */
double   *B;     /* B[M] */
int      *J;     /* J[M] */
double   *P;     /* P[M] */
```

DESCRIPTION

mx_householder_solve() solves system of linear equations with Householder method.

EXAMPLE

```
# include        <stdio.h>
# include        <mx.h>

main()
  {
        double   **a, *x, *p, det;
        int      i, j, *jp;

        x = (double *)mx1alloc( 3, sizeof(double) );
        p = (double *)mx1alloc( 3, sizeof(double) );
        a = (double **)mx2alloc( 3, 4, sizeof(double) );
        jp = (int *)mx1alloc( 3, sizeof(int) );

        a[0][0] = 1.0;    a[0][1] = 2.0;    a[0][2] = 4.0;
        a[1][0] = 2.0;    a[1][1] = 7.0;    a[1][2] = 23.0;
        a[2][0] = 4.0;    a[2][1] = 13.0;   a[2][2] = 47.0;

        a[0][3] = 11.0;
        a[1][3] = 43.0;
        a[2][3] = 85.0;

        mx_householder_solve( 3, 3, a, x, jp, p );
        for ( i = 0 ; i < 3 ; i++ )
          {     for ( j = 0 ; j < 3 ; j++ )
                        printf( "a[%d][%d] = %f\t", i, j, a[i][j] );
                printf( "\n" );
          }
        for ( i = 0 ; i < 3 ; i++ )
                printf( "x[%d] = %f, %d\t", i, x[i], jp[i] );
        printf( "\n" );
  }
```

LIBRARY

libmx.a

AUTHOR

Seiichi TENPAKU

NAME

mx_trans_mul, mx_mx_mul, mx_vc_mul – matrix multiply function utilities.

SYNOPSIS

```
# include        <mx.h>

void    mx_trans_mul( A, C, N, M )
double  **A;    /* A[N][M] */
double  **C;    /* C[M][M] */
int     N, M;

void    mx_mx_mul( A, B, C, N, M )
double  **A;    /* A[N][M] */
double  **B;    /* B[M][N] */
double  **C;    /* C[M][M] */
int     N, M;

void    mx_vc_mul( A, B, C, N, M )
double  **A;    /* A[N][M] */
double  *B;     /* B[M]    */
double  *C;     /* C[N]    */
int     N, M;
```

DESCRIPTION

mx_trans_mul() multiplies a matirx of *A[N][M]* and a transposed matrix of *A[N][M]* and puts the result into a matrix *C[N][N]*.

mx_mx_mul() multiplies a matirx of *A[N][M]* and a matrix of *B[N][M]* and puts the result into a matrix *C[N][N]*.

mx_vc_mul() multiplies a matirx of *A[N][M]* and a vector of *B[M]* and puts the result into a vector *C[N]*.

EXAMPLE

```
# include        <stdio.h>
# include        <mx.h>

main()
  {
        double  **a, **b, **c, *x, *y;
        int     i, j;

        a = (double **)mx2alloc( 4, 3, sizeof(double) );
        b = (double **)mx2alloc( 3, 4, sizeof(double) );
        c = (double **)mx2alloc( 3, 3, sizeof(double) );
        x = (double *)mx1alloc( 4, sizeof(double) );
        y = (double *)mx1alloc( 3, sizeof(double) );

        a[0][0] = 0.0;    a[0][1] = 1.0;    a[0][2] = -1.0;
        a[1][0] = -1.0;   a[1][1] = 0.0;    a[1][2] = 1.0;
        a[2][0] = 1.0;    a[2][1] = -1.0;   a[2][2] = 0.0;
        a[3][0] = 0.0;    a[3][1] = 1.0;    a[3][2] = -1.0;

        x[0] = 0.0;       x[1] = 1.0;       x[2] = 2.0;       x[3] = 3.0;

        mx_transpose( a, b, 4, 3 );
```

```
        for ( i = 0 ; i < 3 ; i++ )
          {     for ( j = 0 ; j < 4 ; j++ )
                        printf( "b[%d][%d] = %3.0f\t", i, j, b[i][j] );
                printf( "\n" );
          }
        printf( "\n" );

        mx_mx_mul( b, a, c, 3, 4 );
        for ( i = 0 ; i < 3 ; i++ )
          {     for ( j = 0 ; j < 3 ; j++ )
                        printf( "c[%d][%d] = %3.0f\t", i, j, c[i][j] );
                printf( "\n" );
          }
        printf( "\n" );

        mx_trans_mul( a, c, 4, 3 );
        for ( i = 0 ; i < 3 ; i++ )
          {     for ( j = 0 ; j < 3 ; j++ )
                        printf( "c[%d][%d] = %3.0f\t", i, j, c[i][j] );
                printf( "\n" );
          }
        printf( "\n" );

        mx_vc_mul( b, x, y, 3, 4 );
        for ( i = 0 ; i < 3 ; i++ )
                printf( "y[%d] = %3.0f\t", i, y[i] );
        printf( "\n" );
    }
```

LIBRARY

libmx.a

AUTHOR

Seiichi TENPAKU

NAME

mx_ldua, mx_solve – matrix solve function utilities.

SYNOPSIS

```
# include       <mx.h>

double  mx_ldua( A, N )
double  **A;    /* A[N][N] */
int     N;

double  mx_solve( A, X, N )
double  **A;    /* A[N][N] */
double  *X;     /* X[N]    */
int     N;
```

DESCRIPTION

mx_ldua() decomposes a matirix *A[N][N]* with **LU** method and returns a determinant of *A[N][N]*.

mx_solve() solves system of linear equations by using **mx_ldua()** and returns a determinant of *A[N][N]*.

EXAMPLE

```
# include       <stdio.h>
# include       <mx.h>

main()
  {
        double  **a, *x, det;
        int     i, j;

        x = (double *)mx1alloc( 3, sizeof(double) );
        a = (double **)mx2alloc( 3, 3, sizeof(double) );

        a[0][0] = 1.0;    a[0][1] = 2.0;    a[0][2] = 4.0;
        a[1][0] = 2.0;    a[1][1] = 7.0;    a[1][2] = 23.0;
        a[2][0] = 4.0;    a[2][1] = 13.0;   a[2][2] = 47.0;

        x[0] = 11.0;
        x[1] = 43.0;
        x[2] = 85.0;

        det = mx_solve( a, x, 3 );
        printf( "det = %f\n", det );
        for ( i = 0 ; i < 3 ; i++ )
          {     for ( j = 0 ; j < 3 ; j++ )
                        printf( "a[%d][%d] = %f\t", i, j, a[i][j] );
                printf( "\n" );
          }
        for ( i = 0 ; i < 3 ; i++ )
                printf( "x[%d] = %f\t", i, x[i] );
        printf( "\n" );
  }
```

LIBRARY

libmx.a

AUTHOR

Seiichi TENPAKU

NAME

mx_transpose – transpose matrix.

SYNOPSIS

```
# include        <mx.h>

void    mx_transpose( A, T, N, M )
double  **A;    /* A[N][M] */
double  **T;    /* T[M][N] */
int     N, M;
```

DESCRIPTION

mx_transpose() transposes a matrix of *A[N][M]* into a matrix of *T[M][N]*.

EXAMPLE

```
# include        <stdio.h>
# include        <mx.h>

main()
  {
        double  **a, **b;
        int     i, j;

        a = (double **)mx2alloc( 4, 3, sizeof(double) );
        b = (double **)mx2alloc( 3, 4, sizeof(double) );

        a[0][0] = 1.0;    a[0][1] = 2.0;    a[0][2] = 3.0;
        a[1][0] = 4.0;    a[1][1] = 5.0;    a[1][2] = 6.0;
        a[2][0] = 7.0;    a[2][1] = 8.0;    a[2][2] = 9.0;
        a[3][0] = 10.0;   a[3][1] = 11.0;   a[3][2] = 12.0;

        mx_transpose( a, b, 4, 3 );

        for ( i = 0 ; i < 3 ; i++ )
          {     for ( j = 0 ; j < 4 ; j++ )
                        printf( "b[%d][%d] = %3.0f\t", i, j, b[i][j] );
                printf( "\n" );
          }
  }
```

LIBRARY

libmx.a

AUTHOR

Seiichi TENPAKU

NAME

par_alp_float, par_alp_double, – conversion of PARCOR parameters into linear prediction parameters.

SYNOPSIS

```
double  par_alp_float( PAR, ALP, P )
float   *PAR;
float   *ALP;
int     P;

double  par_alp_double( PAR, ALP, P )
double  *PAR;
double  *ALP;
int     P;
```

DESCRIPTION

par_alp_float() and **par_alp_double**() convert PARCOR parameters into linear prediction parameters and return the value of the residual error.

ARGUMENT

PAR	PARCOR parameters, size of array is P + 1.
ALP	linear prediction parameters, size of array is P + 1.
P	order of linear prediction coefficients.

NOTE

The transfer function H(z) is related to the linear prediction coefficients α_i by the equation:

$$H(z) = \frac{\sigma^2}{1+\sum_{i=1}^{p}\alpha_i z^{-i}}$$

The linear prediction coefficients α_i are calculated from the PARCOR coefficients k_m to repeat ($m = 1,2,\cdots,p$) the equations :

$$\alpha_m^{(m)} = -k_m$$
$$\alpha_i^{(m)} = \alpha_i^{(m-1)} - k_m\alpha_{m-i}^{(m-i)} \quad \left[1 \le i \le m-1\right]$$

BUGS

The value of *ALP[0]* is always 1.

SEE ALSO

alp_cep(3), alp_par(3), cor_alp(3), sig_cor(3)

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

pole_alpha - calculate the parameters of transfer function from the pairs of frequency and bandwidth.

SYNOPSIS

```
void    pole_alpha( Fs, n, Fn, BW, A, B )
double  Fs;
int     n;
double  *Fn, *BW;
double  *A, *B;
```

DESCRIPTION

pole_alpha( ) calculates the parameters of transfer function from the pairs of frequency and bandwidth.

ARGUMENT

Fs — sampling frequency [Hz].

n — number of formant.

Fn — frequency [Hz], Fn[0],...,Fn[n-1].

BW — bandwidth [Hz], BW[0],...,BW[n-1].

A — parameters of the transfer function, size of array is $2 \times n + 1$. The value of *A[0]* is always 1.

B — working array for internal using, size of array is $2 \times n + 1$.

SEE ALSO

pole_zero(3)

EXAMPLE

```
# include       <stdio.h>

main( )
  {
double  Fn[5], BW[5];
double  A[11], B[11];
        int     i;

        Fn[0] = 295.305386;     BW[0] = 370.793796;
        Fn[1] = 819.660240;     BW[1] = 104.146952;
        Fn[2] = 1250.378488;    BW[2] = 88.845487;
        Fn[3] = 2620.233018;    BW[3] = 80.026439;
        Fn[4] = 3661.993899;    BW[4] = 180.353349;

        pole_alpha( 8000.0, 5, Fn, BW, A, B );

        for ( i = 0 ; i <= 10 ; i++ )
                printf( "A[%d] = %f, %f\n", i, A[i], B[i] );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

pole_zero, pole_zero_fast - calculate the pairs of frequency and bandwidth from the parameters of transfer function.

SYNOPSIS

```
int     pole_zero( Fs, n, A, Fn, BW )
double  Fs;
int     n;
double  *A;
double  *Fn, *BW;

int     pole_zero_fast( Fs, n, A, Fn, BW, WX, WY )
double  Fs;
int     n;
double  *A;
double  *Fn;
double  *BW;
complex *WX, *WY;
```

DESCRIPTION

pole_zero() and **pole_zero_fast()** calculate the pairs of frequency and bandwidth from the parameters of transfer function and return *TRUE* = 1 when there is no error or *FALSE* = 0 when there is an error.

ARGUMENT

Fs	sampling frequency [Hz].
n	order of the transfer function.
A	parameters of the transfer function, size of array is *n* + 1. The value of *A[0]* is always 1.
Fn	pole/zero frequency [Hz], Fn[0],...,Fn[n-1]
BW	pole/zero bandwidth [Hz], BW[0],...,BW[n-1]
WX	working array for internal using, size of *WX* is *n*.
WY	working array for internal using, size of *WY* is *n*.

BUGS

pole_zero() uses **alloca()** in *libcommand.a*, however, **pole_zero_fast()** does not use **alloca()**. Both **pole_zero()** and **pole_zero_fast()** use **dka_solve()** in *libst.a.*

SEE ALSO

pole_alpha(3), dka(3)

EXAMPLE

```
# include       <stdio.h>

main()
  {
        int     i;
static  double  a[11];
static  double  rp[10], ip[10];
        int     n = 10;

        a[0] = 1.0;
        a[1] = -1.585643;
        a[2] = 0.467723;
        a[3] = 0.827542;
        a[4] = -0.321415;
```

```
        a[5] = -0.225748;
        a[6] = -0.573873;
        a[7] = 0.872691;
        a[8] = 0.302009;
        a[9] = -1.062851;
        a[10] = 0.523458;
        pole_zero( 8000.0, n, a, rp, ip );
        for ( i = 0 ; i < n ; i++ )
                printf( "%f\t%f\n", rp[i], ip[i] );
}
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

random_var, random_num, random_seed – pseudo-random number generator.

SYNOPSIS

```
# include          <random.h>

double   random_var()

double   random_num( min, max )
double   min, max;

void     random_seed3( ix, iy, iz )
unsigned int ix, iy, iz;

void     random_seed( seed )
unsigned int seed;
```

DESCRIPTION

random_var() uses a Wichmann and Hill's random number generator method [**B. A. Wichmann et al.** (1982)] with period about 6.95×10^{12} to return successive pseudo-random numbers in the range from 0.0 to 1.0.

random_num() returns successive pseudo-random numbers in the range from *min* to *max*, using **random_var()**.

random_seed3() can be called at any time to reset the random-number generator to a random starting point. In this method, the value of *ix*, *iy*, and *iz* must be in the range from 1 to 30000. The generator is initially seeded with a value of 1.

random_seed() can be called at any time to reset the random-number generator to a random starting point. Since this function calls **random_seed3(** *seed*, *seed*, *seed* **)**, the value of *seed* must be in the range from 1 to 30000.

REFERENCE

B. A. Wichmann and **I. D. Hill** (1982) : *Applied Statistics*, **31**, p.188

NOTE

The Wichmann and Hill's random number generator method is equivalent to a multiplicative congruential random number generator method;

$$x_0 = 918999161 \times ix + 917846887 \times iy + 917362583 \times iz$$
$$x_i = \left[16555425264690 \times x_{i-1} \right] mod 27817185604309 \quad \left[i = 1, 2, \cdots, \infty \right]$$

and calculates $x_i / 27817185604309$.

EXAMPLE

```
# include          <stdio.h>
# include          <random.h>

void     main()
  {
         int        i;

         random_seed( 123 );
         for (i = 0; i < 160; i++)
                printf("%10.7f", random_num( -1.0, 1.0 ) );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

sig_cor_float, sig_cor_double, – calculate autocorrelation coefficients.

SYNOPSIS

```
double  sig_cor_float( SIG, DIF, COR, A, P, N )
float   *SIG;
float   *DIF;
float   *COR;
double  A;
int     P;
int     N;

double  sig_cor_double( SIG, DIF, COR, A, P, N )
double  *SIG;
double  *DIF;
double  *COR;
double  A;
int     P;
int     N;
```

DESCRIPTION

sig_cor_float() and **sig_cor_double()** calculate autocorrelation coefficients from differential of waveform data for pre-emphasizing at higher frequency and return the energy of waveform data.

ARGUMENT

SIG	input waveform data, size of array is N.
DIF	differential of waveform data, size of array is N.
COR	autocorrelation coefficients, size of array is P + 1.
A	preemphasis parameter, which is less than 1.0.
P	order of linear prediction coefficients.
N	length of input waveform data.

NOTE

The autocorrelation coefficients ϕ_i is defined by

$$\phi_i = \sum_{n=0}^{N-1-i} x_n x_{n+i} \quad \left[i \geq 0 \right]$$

where x_n $(n = 0,1, \cdots ,N-1)$ are samples of waveform data.

BUGS

The value of *COR[0]* is always 1.

SEE ALSO

alp_cep(3), alp_par(3), cor_alp(3), par_alp(3),

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

set_signal_handler – set the signal handlers of SpeechTools.

SYNOPSIS

void set_signal_handler()

DESCRIPTION

set_signal_handler() set the signal handlers of SpeechTools. **set_signal_handler**() allows following signals:

SIGHUP	1	hangup
SIGILL	4	illegal instruction
SIGEMT	7	EMT instruction
SIGFPE	8	floating point exception
SIGBUS	10	bus error exception
SIGSEGV	11	segmentation violation
SIGSYS	12	bad system call

If these signals are generated, SpeechTools commands always abort without a core dump and display some messages on the standard error (*stderr*). The messages depend on the machine and operating system.

On SUN3/SUN4 systems, **set_signal_handler**() calls ieee_handler() and sets the SpeechTools signal handler for invalid, overflow, and division exceptions. Therefore, Not-A-Number (**NaN**) signal is not ignor.

On MASSCOMP systems, **set_signal_handler**() calls seterropt( ). Using seterropt( ) disables automatic exit completely, and **ERROR**s and **FATAL**s are printed automatically.

SEE ALSO

sigvec(2), signal(3) in *libc.a*

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

smooth_moving_ave – smoothing.

SYNOPSIS

```
void    smooth_moving_ave( n, m, array, window )
int     n, m;
float   *array;
float   *window;
```

DESCRIPTION

smooth_moving_ave() smoothes an *array* with a moving average method.

ARGUMENT

n	length of *array*.
m	length of *window*.
array	input and output data, size of array is n.
window	windowing weight coefficients, size of array is 2(m + 1).

BUGS

The size of *window* is equal to 2(m+1).

EXAMPLE

```
int     m = 3;
float   window[7];

/* set windowing weight coefficients */
window[0] = window[6] = 1.0/8.0;
window[1] = window[5] = 1.0/4.0;
window[2] = window[4] = 1.0/2.0;
window[3] = 1.0;

/* execute */
smooth_moving_ave( n, m, array, window );
```

SEE ALSO

lstsq(3), median(3)

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

stat_arithmetic_mean, stat_correlation, stat_variance – statistics functions.

SYNOPSIS

```
void    stat_arithmetic_mean( n, array, &mean )
int     n;
float   *array;
float   mean;

void    stat_correlation( n, x, y, xmean, ymean, &covariance, &coefficient )
int     n;
float   *x, *y;
float   xmean, ymean;
float   covariance;
float   coefficient;

void    stat_variance( n, array, mean, &variance )
int     n;
float   *array;
float   mean;
float   variance;
```

DESCRIPTION

stat_arithmetic_mean( ) calculates an arithmetic mean of *array*.

stat_correlation( ) calculates a correlation of *x* and *y*.

stat_variance( ) calculates a variance of *array*.

SEE ALSO

average(3)

EXAMPLE

```
# include       <stdio.h>

void    main( )
  {
        static  float
        x[10] = { 50., 69., 73., 90., 93., 100., 129., 145., 149., 193. },
        y[10] = { 34., 37., 40., 45., 48., 54., 60., 68., 72., 85., };
        int     n = 10;
        float   xm, ym, Vxy, Rxy;

        stat_arithmetic_mean( n, x, &xm );
        stat_arithmetic_mean( n, y, &ym );
        stat_correlation( n, x, y, xm, ym, &Vxy, &Rxy );
        printf( "Vxy = %f\n", Vxy );
        printf( "Rxy = %f\n", Rxy );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

strsave, strrev, string_compare, string_partition – string utilities of SpeechTools.

SYNOPSIS

```
char    *strsave( s1 )
char    *s1;

char    *strrev( s1 )
char    *s1;

int     string_compare( s1, s2 )
char    *s1, *s2;

int     string_partition( string, head, tail, key )
char    *string;
char    *head, *tail;
int     key;
```

DESCRIPTION

strsave() returns a pointer to a new string which is a duplicate of the string pointed to by *s1*. The space for the new string is obtained using **malloc()**. If the new string cannot be created, a NULL pointer is returned.

strrev() reverses the string pointed to by *s1* and returns the pointer to the *s1*.

string_compare() compares its arguments and ignores the spaces at the beginning or end of the string. **string_compare()** returns *TRUE* = 1 or *FALSE* = 0.
For example,

```
s1 = "   abc   ", s2 = " abc " -> TRUE
s1 = "   abc   ", s2 = " abcd" -> FALSE
```

string_partition() separates the string to head and tail by key. This function returns *TRUE* = 1 or *FALSE* = 0. For example, if string is "abc/def" and key is '/', then head is "abc" and tail is "/def".

SEE ALSO

string(3) in *libc.a*

LIBRARY

libcommand.a

AUTHOR

Seiichi TENPAKU

NAME

swap_float, swap_double, swap_int, swap_long, swap_short, swap_char – swap data.

SYNOPSIS

```
void    swap_float( &a, &b )
float   a, b;

void    swap_double( &a, &b )
double  a, b;

void    swap_int( &a, &b )
int     a, b;

void    swap_long( &a, &b )
long    a, b;

void    swap_short( &a, &b )
short   a, b;

void    swap_char( &a, &b )
char    a, b;
```

DESCRIPTION

swap_float(), **swap_double**(), **swap_int**(), **swap_long**(), **swap_short**() and **swap_char**() swap two data.

EXAMPLE

```
# include       <stdio.h>

void    main()
  {
        float   a = 10.0;
        float   b = 20.0;

        printf( "a = %f, b = %f\n", a, b );
        swap_float( &a, &b );
        printf( "a = %f, b = %f\n", a, b );
  }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

syn_direct, syn_twomul, syn_pole, syn_zero – speech synthesis function utilities.

SYNOPSIS

```
void    syn_direct( AC, PC, M, D, Xin, &Xout )
double  *AC, *PC;
int     M;
double  *D;
double  Xin, Xout;

void    syn_twomul( RC, M, D, Xin, &Xout )
double  *RC;
int     M;
double  *D;
double  Xin, Xout;

void    syn_pole( Fs, Fn, Bn, D, Xin, &Xout )
double  Fs, Fn, Bn;
double  D[2];
double  Xin, Xout;

void    syn_zero( Fs, Fn, Bn, D, Xin, &Xout )
double  Fs, Fn, Bn;
double  D[2];
double  Xin, Xout;
```

DESCRIPTION

syn_direct( ) execute a direct form filter.

syn_twomul( ) execute a two multiplier lattice synthesis model filter for PARCOR parameters.

syn_pole( ) execute a second order formant filter.

syn_zero( ) execute a second order anti-formant filter.

ARGUMENT

AC	filter coefficients in **syn_direct( ), size of array is M + 1.**
PC	filter coefficients in **syn_direct( ), size of array is M + 1.**
RC	filter coefficients in **syn_twomul( ), size of array is M + 1.**
M	order of the filter coefficients.
Fs	sampling frequency [Hz].
Fn	formant frequency [Hz].
Bn	formant bandwidth [Hz].
D	memory for delay. In **syn_direct**( ) and **syn_twomul**( ), size of array is M + 1. In **syn_pole**( ) and **syn_zero**( ), size of array is 2.
Xin	input signal.
Xout	output signal.

EXAMPLE

```
# include     <stdio.h>

void    do_one_formant( input, output, length, Fs, Fn, Bn )
float   *input, *output;
int     length;
```

```
double  Fs, Fn, Bn;
  {
static    double  D[2];
          double  Xin, Xout;
       int     i;

          D[0] = D[1] = 0.0;          /* initialize for delay */
          for ( i = 0 ; i < length ; i++ )
            {     Xin = input[i];
                  syn_pole( Fs, Fn, Bn, D, Xin, &Xout );
                  output[i] = Xout;
            }
  }
```

REFERENCE

J. D. Markel and **A. H. Gray, Jr.** (1976) : *Linear Prediction of Speech*, Springer-Verlag, New York
D. H. Klatt (1980) : *Software for Cascade / Parallel Formant Synthesizer*, J. Acoust. Soc. Am. **67**, 971-995

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

taper_gettype, taper_linear, taper_second, taper_sine, taper_sinc – taper function utilities.

SYNOPSIS

```
# include      <taper.h>

int     taper_gettype( name )
char    *name;

double  taper_linear( a, b, p, n )
int     a, b, p, n;

double  taper_second( a, b, p, n )
int     a, b, p, n;

double  taper_sine( a, b, p, n )
int     a, b, p, n;

double  taper_sinc( a, b, p, n )
int     a, b, p, n;
```

DESCRIPTION

These functions treat tapers defined in <taper.h> as:

```
# define TAPER_LINEAR     0     /* linear function */
# define TAPER_SECOND     1     /* second function */
# define TAPER_SINE       2     /* sine function */
# define TAPER_SINC       3     /* sinc function */
```

taper_gettype() returns the identifier of taper function.

taper_linear(), **taper_second()**, **taper_sine()** and **taper_sinc()** return the value between 0.0 to 1.0.

ARGUMENT

name string of taper function as follows:
LINEAR
SECOND
SINE
SINC

a duration of rise envelope.

b duration of fall envelope.

p current point.

n total duration.

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

tlist_read, tlist_getword, tlist_search, tlist_parent, tlist_next, tlist_print – tiny list utilities of SpeechTools.

SYNOPSIS

```
# include     <tlist.h>

Tlist   *tlist_read( t, word, max, fp )
Tlist   *t;      /* Tlist pointer */
char    *word;   /* Buffer array of input */
int     max;     /* Size of the buffer */
FILE    *fp;     /* Input file stream */

int     tlist_getword( word, max, fp )
Tlist   *t;      /* Tlist pointer */
char    *word;   /* Buffer array of input */
int     max;     /* Size of the buffer */
FILE    *fp;     /* Input file stream */

Tlist   *tlist_search( t, name )
Tlist   *t;      /* Tlist pointer */
char    *name;   /* Name of Tlist */

Tlist   *tlist_parent( p, t )
Tlist   *p;      /* Current pointer */
Tlist   *t;      /* Target pointer */

Tlist   *tlist_next( t )
Tlist   *t;      /* Target pointer */

void    tlist_print( t )
Tlist   *t;      /* Target pointer */
```

DESCRIPTION

These functions operate or concerned with a **Tlist** structure, defined in <tlist.h> as:

```
# define _HAS_A_STRING_     0
# define _HAS_A_LIST_       1

typedef
struct          tlist
  {             char        *name;      /* Name of tlist */
                char        v_type;     /* Type of value */
                union
                  {         char        *string;
                            struct      tlist       *list;
                  }         value;      /* Value of tlist */
                struct      tlist       *next;      /* Next pointer to tlist */
  }             Tlist;
```

tlist_read( ) reads tlist data from an input stream associated by *fp* and returns a *Tlist* pointer. Note that **tlist_read**( ) calls an external function **strsave**( ) which is included in **libcommand.a.**

tlist_getword() gets one word from input stream associated by *fp* and returns the top character code of *word*. If the input stream reaches an end-of-file, this function returns **EOF** code.

tlist_search() searches the *name* from the current *Tlist* pointer associated by *t* and returns a *Tlist* pointer if it succeeds, or **NULL** pointer if it fails.

tlist_parent() searches a parent of *Tlist t* under the *Tlist p* and returns a *Tlist* pointer.

tlist_next() searches a next of *Tlist t* and returns a *Tlist* pointer.

tlist_print() prints a *Tlist t* to the standard output device (*stdout*).

SEE ALSO

string(3)

EXAMPLE

[text file]

```
Phrase =
  {            { T0 = 1.0; Ap = 0.5; alpha = +20.0;          },
               { T0 = 2.0; Ap = 0.6; alpha = -10.0;          },
               { T0 = 3.0; Ap = 0.7; alpha = 15.0;           }
  };

Accent =
  {            { T1 = 2.0; Aa = 0.5; beta = 30.0; },
               { T1 = 4.0; Aa = 0.6; beta = 25.0; }
  };
```

[program]

```
# include    <stdio.h>
# include    <tlist.h>

void         main( argc, argv )
int          argc;
char         *argv[];
  {
static       char         word[80];
             Tlist        *root, *p, *a, *s, *t, *u;
             FILE         *fp, *fopen();

             fp = fopen( argv[1], "r" );
             root = tlist_read( root, word, 80, fp );
             fclose( fp );

             p = tlist_search( root, "Phrase" );
             s = p->value.list;
             while ( s != NULL )
               {          p = s;
                          while ( (t = tlist_next( p )) != NULL )
                            {          printf( "%s = %s\n", t->name, t->value.string );
                                       p = t;
                            }
                          s = s->next;
               }

             p = tlist_search( root, "Accent" );
             s = p->value.list;
```

```
            while ( s != NULL )
              {           p = s;
                          while ( (t = tlist_next( p )) != NULL )
                            {           printf( "%s = %s\n", t->name, t->value.string );
                                        p = t;
                            }
                          s = s->next;
              }
    }
```

BUGS

These functions have no syntax checking. A comment line starts from # to the end of line.

LIBRARY

libtlist.a

AUTHOR

Seiichi TENPAKU

DIAGNOSTICS

This manual page is not completed, yet.

NAME

winfun_gettype, winfun_set_func, winfun_set_short_float, winfun_set_short_double, winfun_set_float_float, winfun_set_float_double, winfun_generate_float, winfun_generate_double, – windowing function utilities.

SYNOPSIS

```
# include      <winfun.h>

int     winfun_gettype( name )
char    *name;

void    winfun_set_func( type )
int     type;

int     winfun_set_short_float( input, offset, length, frame, len, wnd, type )
short   *input;
long    offset, length;
float   *frame;
int     len, wnd, type;

int     winfun_set_short_double( input, offset, length, frame, len, wnd, type )
short   *input;
long    offset, length;
double  *frame;
int     len, wnd, type;

int     winfun_set_float_float( input, offset, length, frame, len, wnd, type )
float   *input;
long    offset, length;
float   *frame;
int     len, wnd, type;

int     winfun_set_float_double( input, offset, length, frame, len, wnd, type )
float   *input;
long    offset, length;
double  *frame;
int     len, wnd, type;

void    winfun_generate_float( window, wnd, name )
float   *window;
int     wnd;
char    *name;

void    winfun_generate_double( window, wnd, name )
double  *window;
int     wnd;
char    *name;

double  winfun_rectangular( p, n )
int     p, n;

double  winfun_hanning( p, n )
int     p, n;
```

```
double  winfun_hamming( p, n )
int     p, n;

double  winfun_blackman( p, n )
int     p, n;

double  winfun_bartlett( p, n )
int     p, n;

double  winfun_sinc( p, n )
int     p, n;
```

DESCRIPTION

These functions treat windowing functions defined in <winfun.h> as:

```
# define WINFUN_RECTANGULAR    0
# define WINFUN_HANNING        1
# define WINFUN_HAMMING        2
# define WINFUN_BLACKMAN       3
# define WINFUN_BARTLETT       4
# define WINFUN_SINC           5
```

winfun_gettype() returns the type of windowing function.

winfun_set_func() sets the internal value of *type*. This function might be an internal use.

winfun_set_short_float(), **winfun_set_short_double()**, **winfun_set_float_float()** and **winfun_set_float_double()** cut the input array and make one frame data with windowing function. Note that at the started point with *offset* = 0 and at the ended point with *offset* = the center of which is closest to the end point zero data is padded.

winfun_generate_float() and **winfun_generate_double()** generate windowing function value.

winfun_linear(), **winfun_hanning()**, **winfun_hamming()**, **winfun_blackman()**, **winfun_bartlett()** and **winfun_sinc()** returns the value between 0.0 to 1.0.

ARGUMENT

name string of windowing function as follows:
RECTANGULAR
HANNING
HAMMING
BLACKMAN
BARTLETT
SINC

input array of the input data.

offset offset point of the input data.

length total length of the input data.

frame one segment of the input data.

len frame length.

wnd windowing function length.

type type of windowing function.

window array of the windowing function value.

p	current point.
n	windowing function length.

NOTE

Window functions w_p are:

RECTANGULAR

$$w_p = 1 \quad 0 \le p < n$$

HANNING

$$w_p = 0.5 - 0.5\cos\left[2\pi\frac{p}{n} - 1\right] \quad 0 \le p < n$$

HAMMING

$$w_p = 0.54 - 0.46\cos\left[2\pi\frac{p}{n} - 1\right] \quad 0 \le p < n$$

BLACKMAN

$$w_p = 0.42 - 0.5\cos\left[2\pi\frac{p}{n} - 1\right] + 0.08\cos\left[4\pi\frac{p}{n} - 1\right] \quad 0 \le p < n$$

BARTLETT

$$w_p = \begin{cases} 2\frac{p}{n} - 1 & 0 \le p < \frac{2}{n} - 1 \\ 2 - 2\frac{p}{n} - 1 & \frac{2}{n} - 1 \le p < n \end{cases}$$

SINC

$$w_p = \begin{cases} 1 & t = 0 \\ \frac{sin(t)}{t} & t \ne 0 \end{cases} \quad \text{where } t = \left[\frac{2p}{n-1} - 1\right]\pi$$

EXAMPLE

```
# include        <winfun.h>

/* get window type */
type = winfun_gettype( "HAMMING" );

/* loop */
for ( offset = 0 ; offset < length ; offset += shift )
   {     /* set one frame data */
         winfun_set_short_float( input, offset, length, frame, len, wnd, type );

         /* do something */
         ................
   }
```

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU

NAME

ZpToFB – converts Z-plane data to frequency domain data.

SYNOPSIS

```
double  ZpToFB( Fs, Re, Im, &Fn, &BW )
double  Fs;
double  Re, Im;
double  Fn, BW;
```

DESCRIPTION

ZpToFB() converts Z-plane data to frequency domain data and returns the **Q** = (frequency / bandwidth) value. When there is an error, **ZpToFB**() returns 0.0.

ARGUMENT

Fs	sampling frequency [Hz].
Re	real part of complex data in Z-plane.
Im	imaginary part of complex data in Z-plane.
Fn	center frequency [Hz].
BW	bandwidth frequency [Hz].

LIBRARY

libst.a

AUTHOR

Seiichi TENPAKU