REFERENCE MANUAL for
Speech Signal Processing Toolkit Ver. 3.6

December 25, 2012

The help message for every command can be obtained with the option "-h". The help message brings explanation of the command, how to use, as well as its options.

Example: for the command `mcep` (`%` is the shell prompt)

```
> % mcep -h
>
>  mcep - mel cepstral analysis
>
>   usage:
>        mcep [ options ] [ infile ] > stdout
>   options:
>        -a a  : all-pass constant              [0.35]
>        -m m  : order of mel cepstrum          [25]
>        -l l  : frame length                   [256]
>        -h    : print this message
>      (level 2)
>        -i i  : minimum iteration              [2]
>        -j j  : maximum iteration              [30]
>        -d d  : end condition                  [0.001]
>        -e e  : small value added to periodgram [0]
>   infile:
>        windowed sequences (float)             [stdin]
>   stdout:
>        mel-cepstrum (float)
```

For more information related to this toolkit, please refer to http://sourceforge.net/projects/sp-tk/. In this site, the "Examples of Using Speech Signal Processing Toolkit" documentation file can be downloaded. If you have any bug reports, comments, or questions related this toolkit, please use the bug-tracker on SPTK website. We will try to answer every question, but we cannot guarantee it.

Contents

NAME

acep – adaptive cepstral analysis(4; 5)

SYNOPSIS

acep **[–m** *M* **] [–l** *L* **] [–t** *T* **] [–k** *K* **] [–p** *P* **] [–s] [–e** *E* **] [–P** *Pa* **]**
[***pefile*** **] <** ***infile***

DESCRIPTION

acep uses adaptive cepstral analysis (4), (5), to calculate cepstral coefficients from unframed float data from standard input, sending the result to standard output. If *pefile* is given, *acep* writes the prediction error is written to that file.

Both input and output files are in float format.

The algorithm to calculate recursively the adaptive cepstral coefficients is

$$
\begin{aligned}
\boldsymbol{c}^{(n+1)} &= \boldsymbol{c}^{(n)} - \mu^{(n)}\hat{\nabla}\varepsilon_{\tau}^{(n)} \\
\hat{\nabla}\varepsilon_{0}^{(n)} &= -2e(n)\boldsymbol{e}^{(n)} \qquad (\tau = 0) \\
\hat{\nabla}\varepsilon_{\tau}^{(n)} &= -2(1-\tau)\sum_{i=-\infty}^{n}\tau^{n-i}e(i)\boldsymbol{e}^{(i)} \qquad (0 \le \tau < 1) \\
\hat{\nabla}\varepsilon_{\tau}^{(n)} &= \tau\hat{\nabla}\varepsilon_{\tau}^{(n-1)} - 2(1-\tau)e(n)\boldsymbol{e}^{(n)} \\
\mu^{(n)} &= \frac{k}{M\varepsilon^{(n)}} \\
\varepsilon^{(n)} &= \lambda\varepsilon^{(n-1)} + (1-\lambda)e^2(n)
\end{aligned}
$$

where $\boldsymbol{c} = [c(1), \ldots, c(M)]^{\top}$, $\boldsymbol{e}^{(n)} = [e(n-1), \ldots, e(n-M)]^{\top}$. Also, the gain is expressed by $c(0)$ as follows:

$$c(0) = \frac{1}{2}\log\varepsilon^{(n)}$$

In Figure 1, the system for adaptive cepstral analysis is shown.

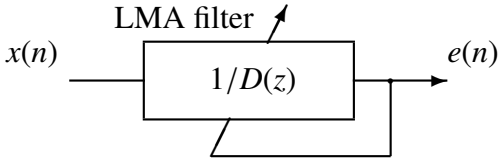


Figure 1: Adaptive cepstral analysis system

OPTIONS

–m	*M*	order of cepstrum	[25]
–l	*L*	leakage factor λ	[0.98]
–t	*T*	momentum constant τ	[0.9]
–k	*K*	step size k	[0.1]
–p	*P*	output period of cepstrum	[1]
–s		output smoothed cepstrum	[FALSE]
–e	*E*	minimum value for $\varepsilon^{(n)}$	[0.0]
–P	*Pa*	number of coefficients of the LMA filter using the Padé approximation. *Pa* should be 4 or 5.	[4]

EXAMPLE

In this example, the speech data is in the file *data.f* in float format, and the cepstral coefficients are written in the file *data.acep* for every block of 100 samples, and the prediction error can be found in *data.er*.

```
acep -m 15 -p 100 data.er < data.f > data.acep
```

SEE ALSO

uels, gcep, mcep, mgcep, amcep, agcep, lmadf

NAME

acorr – obtain autocorrelation sequence

SYNOPSIS

acorr **[–m** *M* **] [–l** *L* **] [** ***infile*** **]**

DESCRIPTION

acorr calculates the m-th order autocorrelation function sequence for each frame of float data from *infile* (or standard input), sending the result to standard output. Namely, the input data is given by

$$x(0), x(1), \ldots, x(L-1),$$

and the autocorrelation is evaluated as

$$r(k) = \sum_{m=0}^{L-1-k} x(m)x(m+k), \qquad k = 0, 1, \ldots, M,$$

and the output is the following autocorrelation function sequence,

$$r(0), r(1), \ldots, r(M)$$

Both input and output files are in float format.

OPTIONS

–m	*M*	order of sequence	[25]
–l	*L*	frame length	[256]

EXAMPLE

In the example below, the input file *data.f* is in float format. Here, the frame length and period are of 256 and 100, respectively. Also, every frame is passed through a Blackman window and the autocorrelation function sequence is sent to *data.acorr*.

```
frame -l 256 -p 100 < data.f | window | acorr -m 10 > data.acorr
```

SEE ALSO

c2acr, levdur

NAME

agcep – adaptive generalized cepstral analysis(9)

SYNOPSIS

agcep **[–m** *M* **]** **[–c** *C* **]** **[–l** *L* **]** **[–t** *T* **]** **[–k** *K* **]** **[–p** *P* **]**
[–s] **[–n]** **[–e** *E* **]** **[** ***pefile*** **]** < ***infile***

DESCRIPTION

agcep uses adaptive generalized cepstral analysis (9) to calculate cepstral coefficients $c_\gamma(m)$ from unframed float data in the standard input, and sends the result to standard output. In the case *pefile* is given, *agcep* writes the prediction error to this file.

Both input and output files are in float format.

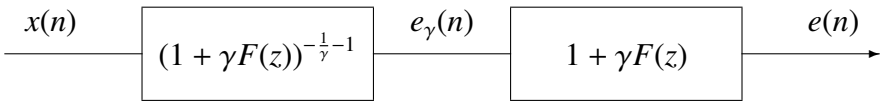
The algorithm which recursively calculates the adaptive generalized cepstral coefficients is shown below.

$$
\begin{aligned}
\boldsymbol{c}_\gamma^{(n+1)} &= \boldsymbol{c}_\gamma^{(n)} - \mu^{(n)}\hat{\nabla}\varepsilon_\tau^{(n)} \\
\hat{\nabla}\varepsilon_0^{(n)} &= -2e_\gamma(n)\boldsymbol{e}_\gamma^{(n)} \qquad (\tau = 0) \\
\hat{\nabla}\varepsilon_\tau^{(n)} &= -2(1-\tau)\sum_{i=-\infty}^{n} \tau^{n-i} e_\gamma(i)\boldsymbol{e}_\gamma^{(i)} \qquad (0 \le \tau < 1) \\
\hat{\nabla}\varepsilon_\tau^{(n)} &= \tau\hat{\nabla}\varepsilon_\tau^{(n-1)} - 2(1-\tau)e_\gamma(n)\boldsymbol{e}_\gamma^{(n)} \\
\mu^{(n)} &= \frac{k}{M\varepsilon^{(n)}} \\
\varepsilon^{(n)} &= \lambda\varepsilon^{(n-1)} + (1-\lambda)e_\gamma^2(n)
\end{aligned}
$$

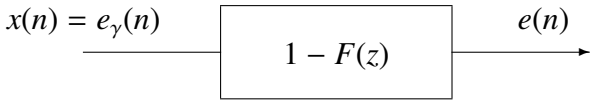
where $\boldsymbol{c}_\gamma = [c_\gamma(1), \ldots, c_\gamma(M)]^\top$, $\boldsymbol{e}_\gamma = [e_\gamma(n-1), \ldots, e_\gamma(n-M)]^\top$. The signal $e_\gamma(n)$ is obtained by passing the input signal $x(n)$ through the filter $(1 + \gamma F(z))^{-\frac{1}{\gamma}-1}$, where

$$
F(z) = \sum_{m=1}^{M} c_\gamma(m) z^{-m}.
$$

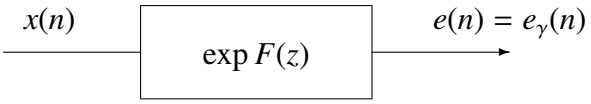
In the case where $\gamma = -1/n$ and n is a natural number, the adaptive generalized cepstral analysis system is as shown in Figure 1. In the case $n = 1$, the adaptive generalized cepstral analysis is equivalent to the LMS linear predictor. Also, when $n \to \infty$, the adaptive generalized cepstral analysis is equivalent to the adaptive cepstral analysis.



(a) $-1 \leq \gamma \leq 0$



(b) $\gamma = -1$



(c) $\gamma = 0$

Figure 1: Adaptive generalized cepstral analysis system

OPTIONS

–m	*M*	order of generalized cepstrum	[25]
–c	*C*	power parameter $\gamma = -1/C$ for generalized cepstrum	[1]
–l	*L*	leakage factor λ	[0.98]
–t	*T*	momentum constant τ	[0.9]
–k	*K*	step size k	[0.1]
–p	*P*	output period of generalized cepstrum	[1]
–s		output smoothed generalized cepstrum	[FALSE]
–n		output normalized generalized cepstrum	[FALSE]
–e	*E*	minimum value for $\varepsilon^{(n)}$	[0.0]

EXAMPLE

In this example, the speech data is in the file *data.f* in float format and the prediction error can be found in *data.er*. The cepstral coefficients are written to the file *data.agcep*,

```
agcep -m 15 data.er < data.f > data.agcep
```

SEE ALSO

acep, amcep, glsadf

NAME

amcep – adaptive mel-cepstral analysis(11; 12)

SYNOPSIS

amcep **[–m** *M* **] [–a** *A* **] [–l** *L* **] [–t** *T* **] [–k** *K* **] [–p** *P* **] [–s] [–e** *E* **]**
[–P *Pa* **] [** ***pefile*** **]** < ***infile***

DESCRIPTION

amcep uses adaptive mel-cepstral analysis to calculate mel-cepstral coefficients $c_\alpha(m)$ from unframed float data in the standard input, sending the result to standard output. In the case *pefile* is given, *amcep* writes the prediction error to this file.

Both input and output files are in float format.

The algorithm which recursively calculates the adaptive mel-cepstral coefficients $b(m)$ is shown below

$$
\begin{aligned}
\boldsymbol{c}^{(n+1)} &= \boldsymbol{b}^{(n)} - \mu^{(n)}\hat{\nabla}\varepsilon_\tau^{(n)} \\
\hat{\nabla}\varepsilon_0^{(n)} &= -2e(n)\boldsymbol{e}_\Phi^{(n)} \qquad (\tau = 0) \\
\hat{\nabla}\varepsilon_\tau^{(n)} &= -2(1-\tau)\sum_{i=-\infty}^{n} \tau^{n-i} e(i)\boldsymbol{e}_\Phi^{(i)} \qquad (0 \le \tau < 1) \\
\hat{\nabla}\varepsilon_\tau^{(n)} &= \tau\hat{\nabla}\varepsilon_\tau^{(n-1)} - 2(1-\tau)e(n)\boldsymbol{e}_\Phi^{(n)} \\
\mu^{(n)} &= \frac{k}{M\varepsilon^{(n)}} \\
\varepsilon^{(n)} &= \lambda\varepsilon^{(n-1)} + (1-\lambda)e^2(n)
\end{aligned}
$$

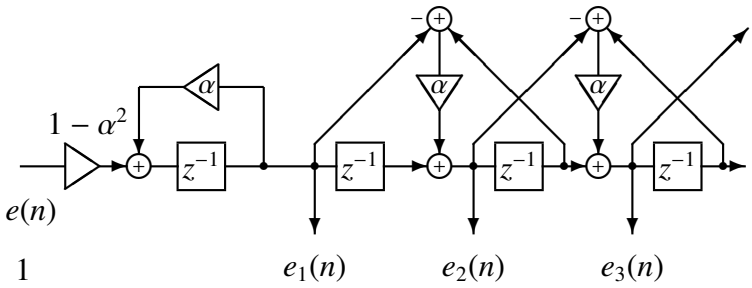


Figure 1: Filter $\Phi_m(z)$

where $\boldsymbol{b} = [b(1), b(2), \ldots, b(M)]^\top$, $\boldsymbol{e}_\Phi^{(n)} = [e_1(n), e_2(n), \ldots, e_M(n)]^T$, $e_m(n)$ is the output of the inverse filter, which is obtained as shown in Figure 1, passing $e(n)$ through the filter $\Phi_m(z)$.

The coefficients $b(m)$ are equivalent to the coefficients of the MLSA filter, and the mel-cepstral coefficients $c_\alpha(m)$ can be obtained from $b(m)$ through a linear transformation (refer to b2mc and mc2b).

Thus, the adaptive mel-cepstral analysis system is shown in figure 2.

The filter $1/D(z)$ is realized by a MLSA filter.

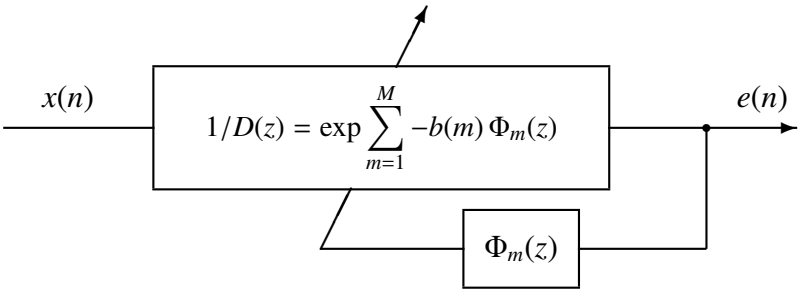


Figure 2: Adaptive mel-cepstral analysis system

OPTIONS

–m	*M*	order of mel-cepstrum	[25]
–a	*A*	all-pass constant α	[0.35]
–l	*L*	leakage factor λ	[0.98]
–t	*T*	momentum constant τ	[0.9]
–k	*K*	step size k	[0.1]
–p	*P*	output period of mel-cepstrum	[1]
–s		output smoothed mel-cepstrum	[FALSE]
–e	*E*	minimum value for $\varepsilon^{(n)}$	[0.0]
–P	*Pa*	number of coefficients of the MLSA filter using the Padé approximation. *Pa* should be 4 or 5.	[4]

EXAMPLE

In this example, the speech data is in the file *data.f* in float format, and the adaptive mel-cepstral coefficients are written to the file *data.amcep* for every block of 100 samples:

```
amcep -m 15 -p 100  < data.f > data.amcep
```

SEE ALSO

acep, agcep, mc2b, b2mc, mlsadf

NAME

average – calculate mean for each block

SYNOPSIS

average **[–l** *L* **] [–n** *N* **] [** ***infile*** **]**

DESCRIPTION

average calculates the mean value for every *L*-length block from *infile* (or standard input), sending the result to standard output.

For the input data

$$x(0), x(1), \ldots, x(L-1)$$

the output is calculated as follows:

$$\frac{x(0) + x(1) + \ldots + x(L-1)}{L}$$

If $L = 0$, then the whole input data is used to calculate the average.

Both input and output files are in float format.

OPTIONS

–l	*L*	number of items contained 1 frame	[0]
–n	*N*	order of items contained 1 frame	[L-1]

EXAMPLE

The output file *data.av* contains the mean taken from the whole data in *data.f*, in float format.

```
average < data.f > data.av
```

SEE ALSO

histogram, vsum, vstat

NAME

b2mc – transform MLSA digital filter coefficients to mel-cepstrum

SYNOPSIS

b2mc **[–m** *M* **] [–a** *A* **] [** ***infile*** **]**

DESCRIPTION

b2mc calculates mel-cepstral coefficients $c_\alpha(m)$ from MLSA filter coefficients $b(m)$ in the *infile* (or standard input), sending the result to standard output.

Input and output data are in float format.

The transformation from $b(m)$ coefficients to mel-cepstral coefficients $c_\alpha(m)$ is as follows:

$$c_\alpha(m) = \begin{cases} b(M) & m = M \\ b(m) + \alpha b(m+1) & 0 \le m < M \end{cases}$$

The command *b2mc* and *mc2b* are in inverse conversion relationship to each other.

OPTIONS

–m	*M*	order of mel cepstrum	[25]
–a	*A*	all-pass constant α	[0.35]

EXAMPLE

The example below converts the coefficients of an MLSA filter, which are in file *data.b* in float format, into mel-cepstral coefficients in file *data.mcep*, with $M = 15$ and $\alpha = 0.35$.

```
b2mc -m 15 < data.b > data.mcep
```

SEE ALSO

mc2b, mcep, mlsadf

NAME

bcp – block copy

SYNOPSIS

bcp **[–l** *l* **] [–L** *L* **] [–n** *n* **] [–N** *N* **] [–s** *s* **] [–S** *S* **] [–e** *e* **] [–f** *f* **]**

[+*type*] [*infile*]

DESCRIPTION

bcp copies data blocks from *infile* (or standard input) to standard output, and reformats them according to the command line options given.

If the input format is ASCII, the basic input unit is a sequence of letters and the output block is partitioned with carriage returns.

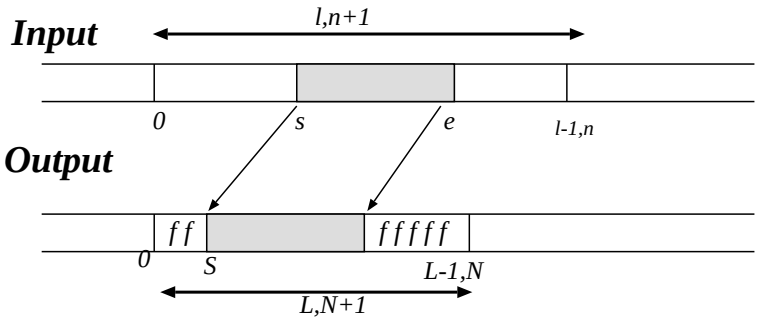


Figure 3: Example of the bcp command

OPTIONS

–l	*l*	number of items contained 1 block	[512]
–L	*L*	number of destination block size	[N/A]
–n	*n*	order of items contained 1 block	[l-1]
–N	*N*	order of destination block size	[N/A]
–s	*s*	start number	[0]
–S	*S*	start number in destination block	[0]
–e	*e*	end number	[EOF]
–f	*f*	fill into empty block	[0]

+t data type [f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)
a	ASCII letter sequence		

EXAMPLE

Assume that a(0), a(1), a(2), ... , a(20) is contained in the input file *data.f*, written in float format. If one wants to copy the array a(1), a(2), ... , a(10), the following command can be used.

```
bcp +f -l 21 -s 1 -e 10 data.f > data.bcp
```

A different example with respect to the same input file *data.f* follows

```
bcp +f -l 21 -s 3 -e 5 -S 6 -L 10 data.f > data.bcp
```

In this example, the output block is

```
0, 0, 0, 0, 0, 0, a(3), a(4), a(5), 0
```

SEE ALSO

bcut, merge, reverse

NAME

bcut – binary file cut

SYNOPSIS

bcut **[–s** *S* **] [–e** *E* **] [–l** *L* **] [–n** *N* **] [*+type*] [*infile*]**

DESCRIPTION

bcut copies a selected portion of *infile* (or standard input) to standard output.

OPTIONS

–s	*S*	start number	[0]
–e	*E*	end number	[EOF]
–l	*L*	block length	[1]
–n	*N*	block order	[L-1]
+t		input data format	[f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)

EXAMPLE

In the example below, the input file *data.f* in float format is cut from the 3rd to the 5th float point:

```
bcut +f -s 3 -e 5 data.f > data.cut
```

For example, if the file *data.f* had the following data

$$1, 2, 3, 4, 5, 6, 7$$

the output file *data.cut* would be

$$4, 5, 6.$$

If the block length is assigned:

```
bcut +f -l 2 data.f -s 1 -e 2 > data.cut
```

then, the output file would contain the following data,

$$3, 4, 5, 6$$

If the stationary part, say from the sample 100, of the output of a digital filter excited with pulse train is desired, then the following command can be used:

```
train -p 10 -l 256 | dfs -a 1 0.8 0.6 | bcut +f -s 100 > data.cut
```

In this case, the file *data.cut* will contain 156 points.

If we generate a *data.f* file passing a sinusoidal signal through a 256-length window as follows

```
sin -p 30 -l 2000 | window > data.f
```

and we want to take only the third window output, we could use the following command:

```
bcut +f -l 256 -s 3 -e 3 < data.f > data.cut
```

SEE ALSO

bcp, merge, reverse

NAME

bell – ring a bell

SYNOPSIS

bell **[** *num* **]**

DESCRIPTION

bell rings a bell *num* times.

OPTIONS

num	number of times bell rings	[1]

EXAMPLE

This example rings bell 10 times:

```
bell 10
```

NAME

c2acr – transform cepstrum to autocorrelation

SYNOPSIS

c2acr **[–m** M_1 **] [–M** M_2 **] [–l** L **] [** ***infile*** **]**

DESCRIPTION

c2acr calculates M_2-th order autocorrelation coefficients from M_1-th order cepstral coefficients in the *infile* (or standard input), writing the result to standard output. Given the cepstral coefficients

$$c(0), c(1), \ldots, c(M_1)$$

the corresponding autocorrelation coefficients are given by

$$r(0), r(1), \ldots, r(M_2)$$

Both input and output files are in float format.

The power spectrum is calculated from the logarithm spectrum, which is obtained from the Fourier transform of the M_1-th order cepstral coefficients. The autocorrelation coefficients are obtained through the inverse Fourier transform of the power spectrum.

OPTIONS

–m	M_1	order of cepstrum	[25]
–M	M_2	order of autocorrelation	[25]
–l	L	FFT length	[256]

EXAMPLE

In the following example, the 15-th order linear prediction coefficients are calculated from the 30-th order cepstral coefficients in *data.cep* and the result is sent to the *data.lpc*.

```
c2acr -m 30 -M 15 < data.cep | levdur -m 15 > data.lpc
```

SEE ALSO

uels, c2sp, c2ir, lpc2c

NAME

c2ir – cepstrum to minimum phase impulse response

SYNOPSIS

c2ir [–l L **] [–m** M_1 **] [–M** M_2 **] [–i] [** ***infile*** **]**

DESCRIPTION

c2ir calculates the minimum phase impulse response from the minimum phase cepstral coefficients in the *infile* (or standard input), sending the result to standard output. For example, if the input sequence is

$$c(0), c(1), c(2), \ldots, c(M_1)$$

then the impulse response is calculated as

$$h(n) = \begin{cases} h(0) = \exp(c(0)) \\ h(n) = \sum_{k=1}^{M_1} \frac{k}{n} c(k) h(n-k) \quad n \geq 1 \end{cases}$$

and the output will be given by

$$h(0), h(1), h(2), \ldots, h(L-1)$$

Both input and output files are in float format.

OPTIONS

–m	M_1	order of cepstrum	[25]
–M	M_2	length of impulse response	[L-1]
–l	L	order of impulse response	[256]
–i		input minimum phase sequence	[FALSE]

If the number of cepstral coefficients M_1 is not assigned and the order of the cepstral analysis is less then L, then the number of coefficients read is made equal to M_1.

EXAMPLE

The output file *data.ir* contains the impulse response in the range $n = 0 \sim 99$ obtained from the 30-th order cepstral coefficients file *data.cep*, in float format:

```
c2ir -l 100 -m 30 data.cep > data.ir
```

SEE ALSO

c2sp, c2acr

NAME

c2sp – transform cepstrum to spectrum

SYNOPSIS

c2sp **[–m** *M* **] [–l** *L* **] [–p] [–o** *O* **] [** ***infile*** **]**

DESCRIPTION

c2sp calculates the spectrum from the minimum phase cepstrum from *infile* (or standard input), sending the result to standard output. Input and output data are in float format.

OPTIONS

–m	*M*	order of cepstrum	[25]
–l	*L*	frame length	[256]
–p		output phase	[FALSE]
–o	*O*	output format	[0]

if the "–p" option is not assigned then

$O = 0$ $20 \times \log |H(z)|$
$O = 1$ $\ln |H(z)|$
$O = 2$ $|H(z)|$

if the "–p" option is assigned then

$O = 0$ $\arg |H(z)| \div \pi$ $[\pi\ rad.]$
$O = 1$ $\arg |H(z)|$ $[rad.]$
$O = 2$ $\arg |H(z)| \times 180 \div \pi$ $[deg.]$

EXAMPLE

The example below takes the 15-th order cepstrum from the file *data.cep* in float format, evaluates the running spectrum, and presents it in the screen:

```
c2sp -m 15 data.cep | grlogsp | xgr
```

SEE ALSO

uels, mgc2sp

NAME

cdist – calculation of cepstral distance

SYNOPSIS

cdist [–m *M* **] [–o** *O* **] [–f]** ***cfile*** **[** ***infile*** **]**

DESCRIPTION

cdist calculates the cepstral distance between the cepstral coefficients in *infile* (or standard input) and the ones in *cfile*, sending the result to standard output. For example, if the cepstral coefficients of the *infile* at frame t are

$$c_{1,t}(0), c_{1,t}(1), c_{1,t}(2), \ldots, c_{1,t}(M)$$

and the cepstral coefficients in *cfile* at frame t are

$$c_{2,t}(0), c_{2,t}(1), c_{2,t}(2), \ldots, c_{2,t}(M)$$

then the squared cepstrum distance for every frame is given by

$$d(t) = \sum_{k=1}^{M} (c_{1,t}(k) - c_{2,t}(k))^2$$

and the total cepstral distance between both files is

$$d = \frac{1}{T} \sum_{t=0}^{T} d(t)$$

If the number of frames in the two files is different, then *cdist* will consider the smallest number for the evaluation.

OPTIONS

–m	*M*	order of minimum-phase cepstrum	[25]
–o	*O*	output format	[0]

$O = 0$ $\frac{10}{\ln 10} \sqrt{2d(t)}$ [db]
$O = 1$ $d(t)$
$O = 2$ $\sqrt{d(t)}$

–f	output frame by frame	[FALSE]

EXAMPLE

In the example below, the squared spectral distance of the 15-th order cepstrum files *data1.cep* and *data2.cep*, both in float formats, is evaluated and displayed:

```
cdist -m 15 data1.cep data2.cep | dmp +f
```

SEE ALSO

acep, agcep, amcep, mcep

NAME

clip – data clipping

SYNOPSIS

clip **[–y** y_{min} y_{max} **] [–ymin** y_{min} **] [–ymax** y_{max} **] [** ***infile*** **]**

DESCRIPTION

clip clips the data from *infile* (or standard input) between the minimum and maximum values specified on the command line, sending the result to standard output.

Input and output data are in float format.

OPTIONS

–y	y_{min} y_{max}	lower bound & upper bound	[−1.0 1.0]
–ymin	y_{min}	lower bound (ymax = inf)	[N/A]
–ymax	y_{max}	upper bound (ymin = -inf)	[N/A]

EXAMPLE

Suppose that the data in *data.f* is in float format and presents the following values,

1.0, 2.0, 3.0, 4.0, 5.0, 6.0

If we type the command

```
clip -y 2.5 5.5 < data.f > data.clip
```

then the output *data.clip* will contain the following values.

2.5, 2.5, 3.0, 4.0, 5.0, 5.5

NAME

da – play 16-bit linear PCM data

SYNOPSIS

da **[–s** *S* **] [–c** *C* **] [–g** *G* **] [–a** *A* **] [–o** *O* **] [–w] [–H** *H* **]**
[–v] [+*type* **] [** ***infile1*** **] [** ***infile2*** **] ...**

DESCRIPTION

da plays a series of input files (or standard input) on a system-dependent audio output device. If the system does not support the specified sampling frequency, *da* up-samples the data to a supported frequency. This command can be used under Linux (i386), FreeBSD (i386 newpcm driver), SunOS 4.1.x, SunOS 5.x (SPARC).

It is possible to change the environment settings through the following options

DA_GAIN	gain
DA_AMPGAIN	amplitude gain
DA_PORT	output port
DA_HDRSIZE	header size
DA_FLOAT	set the input data to float

OPTIONS

–s	*S*	sampling frequency, it can be used the following sampling frequencies 8, 10, 11.025, 12, 16, 20, 22.05, 32, 44.1, 48 (kHz).	[10]
–g	*G*	gain	[0]
–a	*A*	amplitude gain(0..100)	[N/A]
–o	*O*	output port(s : speaker, h : headphone)	[s]
–w		execute byte swap	[FALSE]
–H	*H*	header size in byte	[0]
–v		display filename	[FALSE]
+type		input data format	[f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)

EXAMPLE

In the following example, the speech data file *data.s* is played on the headphone. The sampling frequency is 8 kHz, and the input data is in short format.

```
da +s -s 8 -o h data.s
```

BUGS

In Linux operating systems, the output port can not be assigned.

NAME

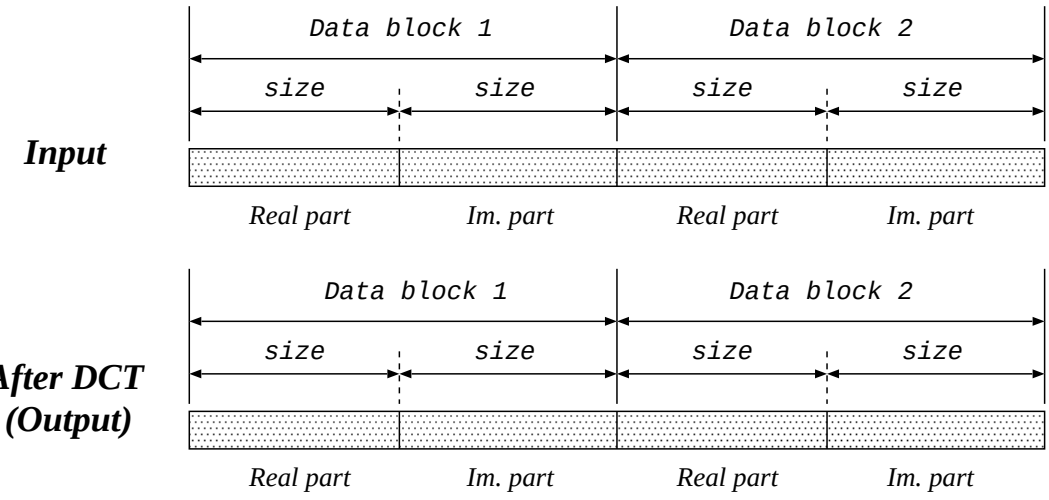
dct – DCT-II

SYNOPSIS

dct [–l *L* **] [–I] [–d] [** ***infile*** **]**

DESCRIPTION

dct calculates the Discrete Cosine Transform II (DCT-II) of the input data in the *infile* (or standard input), sending the results to standard output. The input and output data are both in float format, and arranged as follows.



The Discrete Cosine Transform II can be written as:

$$X_k = \sqrt{\frac{2}{L}} c_k \sum_{l=0}^{L-1} x_l \cos\left\{\frac{\pi}{L} k \left(l + \frac{1}{2}\right)\right\}, \quad l = 0, 1, \cdots, L$$

where

$$c_k = \begin{cases} 1 & (1 \le k \le L-1) \\ 1/\sqrt{2} & (k = 0) \end{cases}$$

OPTIONS

–l	*L*	DCT size	[256]
–I		use complex number	[FALSE]
–d		don't use FFT algorithm	[FALSE]

EXAMPLE

In this example, the DCT is evaluated from a complex-valued data file *data.f* in float format (real part: 256 points, imaginary part: 256 points), and the output is written to *data.dct*:

```
dct data.f -l 256 -I > data.dct
```

SEE ALSO

fft, idct

NAME

decimate – decimation (data skipping)

SYNOPSIS

decimate [–p *P* **] [–s** *S* **] [** ***infile*** **]**

DESCRIPTION

decimate picks up a sequence of input data from *infile* (or standard input) with interval P and start number S, sending the result to standard output.

If the input data is

$$x(0), x(1), x(2), \ldots$$

then the output data is given by:

$$x(S), x(S+P), x(S+2P), x(S+3P), \ldots$$

Input and output data are in float format.

OPTIONS

–p	*P*	decimation period	[10]
–s	*S*	start sample	[0]

EXAMPLE

This example decimates input data from *data.f* file with interval 2, interpolates 0 with interval 2, and then outputs the results to the file *data.di*:

```
decimate -p 2  < data.f | interpolate -p 2 > data.di
```

SEE ALSO

interpolate

NAME

delay – delay sequence

SYNOPSIS

delay **[–s** *S* **] [–f] [** ***infile*** **]**

DESCRIPTION

delay delays the data in *infile* (or standard input) by inserting a specified number of zero samples at the beginning, and sends the result to standard output. For example, if we want to delay the following data

$$x(0), x(1), \ldots, x(T)$$

as in

$$\underbrace{0, \ldots, 0}_{S}, x(0), x(1), \ldots, x(T).$$

We only need to set the "–s" option to S

$$\underbrace{0, \ldots, 0}_{S}, x(0), x(1), \ldots, x(T-S).$$

Both input and output files are in float format.

OPTIONS

–s	*S*	start sample	[0]
–f		keep file length	[FALSE]

EXAMPLE

If we have the following data in the input *data.f* file

$$1.0, 2.0, 3.0, 4.0, 5.0, 6.0$$

and we use the command below

```
delay -s 3 < data.f > data.delay
```

then the output file *data.delay* will be

$$0.0, 0.0, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0$$

As another example, if we want to keep the same size of the input file, we can use the following command,

```
delay -s 3 -f < data.f > data.delay
```

and the output *data.delay* will be

$$0.0, 0.0, 0.0, 1.0, 2.0, 3.0$$

NAME

delta – delta calculation

SYNOPSIS

delta **[–m** M **]** **[–l** L **]** **[–t** T **]** **[–d** (fn | d_0 **[**d_1 ...**]**) **]** **[–r** N_R W_1 **[**W_2**]** **]**

[–R N_R W_{F1} W_{B1} **[**W_{F2} W_{B2}**]]** **[–M** *magic* **]** **[** ***infile*** **]**

DESCRIPTION

delta calculates dynamic features from *infile* (or standard input), sending the result (static and dynamic features) to the standard output. Input and output are of the form:

$$\begin{aligned}&\text{input} \ldots, x_t(0), \ldots, x_t(M), \ldots \\ &\text{output} \ldots, x_t(0), \ldots, x_t(M), \Delta^{(1)}x_t(0), \ldots, \Delta^{(1)}x_t(M), \ldots, \Delta^{(n)}x_t(0), \ldots, \Delta^{(n)}x_t(M), \ldots\end{aligned}$$

Also, input and output data are in float format. The dynamic feature vector $\Delta^{(n)}\boldsymbol{x}_t$ can be obtained from the static feature vector as follows.

$$\Delta^{(n)}\boldsymbol{x}_t = \sum_{\tau=-L^{(n)}}^{L^{(n)}} w^{(n)}(\tau)\boldsymbol{x}_{t+\tau}$$

where n is the order of the dynamic feature vector. For example, when we evaluate the Δ^2 parameter, $n = 2$.

OPTIONS

–m	M	order of vector	[25]
–l	L	length of vector	[M + 1]

–d $(fn \mid d_0\ [d_1\ \ldots])$ [N/A]

fn is the file name of the parameters $w^{(n)}(\tau)$ used when evaluating the dynamic feature vector. It is assumed that the number of coefficients to the left and to the right are the same. In case this is not true, then zeros are added to the shortest side. For example, if the coefficients are given by:

$$w(-1), w(0), w(1), w(2), w(3)$$

then zeros must be added to the left as follows.

$$0, 0, w(-1), w(0), w(1), w(2), w(3)$$

Instead of entering the filename fn, the coefficients(which compose the file fn) can be directly inputted from the command line. When the order of the dynamic feature vector is higher than one, then the sets of coefficients can be inputted one after the other as shown in the example below. This option cannot be used with the –r nor –R options.

–r N_R W_1 [W_2] This option is used when N_R-th order dynamic parameters are used and the weighting coefficients $w^{(n)}(\tau)$ are evaluated by regression. N_R can be made equal to 1 or 2. The variables W_1 and W_2 represent the widths of the first and second order regression coefficients, respectively. The first order regression coefficients for $\Delta \boldsymbol{x}_t$ at frame t are evaluated as follows. [N/A]

$$\Delta \boldsymbol{x}_t = \frac{\sum_{\tau=-W_1}^{W_1} \tau \boldsymbol{c}_{t+\tau}}{\sum_{\tau=-W_1}^{W_1} \tau^2}$$

For the second order regression coefficients, $a_2 = \sum_{\tau=-W_2}^{W_2} \tau^4$, $a_1 = \sum_{\tau=-W_2}^{W_2} \tau^2$, $a_0 = \sum_{\tau=-W_2}^{W_2} 1$ and

$$\Delta^2 \boldsymbol{x}_t = \frac{2 \sum_{\tau=-W_2}^{W_2} (a_0 \tau^2 - a_1) \boldsymbol{x}_{t+\tau}}{a_2 a_0 - a_1^2}$$

This option cannot be used with the –d nor –R options.

–R N_R W_{F1} W_{B1}[W_{F2} W_{B2}] Similarly to the –r option, by using this option, we can obtain N_R-th order dynamic feature parameters and the weighting coefficients will be evaluated by regression. N_R can be made equal to 1 or 2. The variables W_{Fi} and W_{Bi} represent the width of the i-th order regression coefficients in the forward and backward direction, respectively. Combining this option with the –M option, the regression coefficients can be evaluated skipping the magic number from the input. This option cannot be used with the –d nor –r options. [N/A]

–M *magic* The magic number *magic* can be skipped from the input during the calculation of the dynamic features. This option is valid only when the –R option is also specified. [N/A]

EXAMPLE

In the example below, the first and second order dynamic features are calculated from 15-dimensional coefficient vectors from *data.static* using windows whose width are 1. The resultant static and dynamic features are sent to *data.delta*:

```
delta -m 15 -r 2 1 1 data.static > data.delta
```

or

```
echo "-0.5 0 0.5" | x2x +af > delta
echo "1.0 -2.0 1.0" | x2x +af > accel
delta -m 15 -d delta -d accel data.static > data.delta
```

Another example is presented bellow, where the first and second order dynamic features are calculated from the scalar sequence in *data.f0*, sending windows with 2 units width and skipping the magic number -1.0E15.

```
delta -l 1 -R 2 2 2 2 2 -M -1.0E15 data.f0 > data.delta
```

SEE ALSO

mlpg

NAME

df2 – second order standard form digital filter

SYNOPSIS

df2 **[–s** S **] [–p** f_1 b_1 **] [–z** f_2 b_2 **] [** ***infile*** **]**

DESCRIPTION

df2 filters data from *infile* (or standard input) using a second order digital filter in standard form, sending the result to standard output. The central frequency and frequency band can be both assigned through the options, shown bellow. The filter transfer function is given by:

$$H(z) = \frac{1 - 2\exp(-\pi b_2/f_0)\cos(2\pi f_2/f_0)z^{-1} + \exp(-2\pi b_2/f_0)z^{-2}}{1 - 2\exp(-\pi b_1/f_0)\cos(2\pi f_1/f_0)z^{-1} + \exp(-2\pi b_1/f_0)z^{-2}}$$

Also, if this command is used in cascade, an arbitrary filter can be designed by using the options –p and –z. Input and output data are in float format.

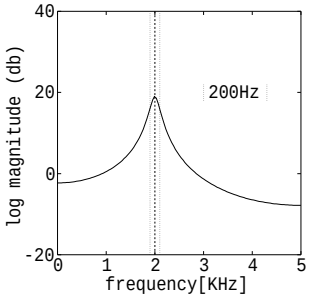
OPTIONS

–s	S	sampling frequency S [kHz]	[10.0]
–p	f_1 b_1	center frequency f_1 [Hz] and band width b_1 [Hz] of pole	[N/A]
–z	f_2 b_2	center frequency f_2 [Hz] and band width b_2 [Hz] of zero	[N/A]

EXAMPLE

The command below gives the impulse response of a filter with a pole at 2000 Hz and a frequency band of 200 Hz:

```
impulse | df2 -p 2000 200 | fdrw | xgr
```



NAME

dfs – digital filter in standard form

SYNOPSIS

dfs **[–a** $K\ a(1)\ \ldots\ a(M)$ **] [–b** $b(0)\ b(1)\ \ldots\ b(N)$ **] [–p *pfile*] [–z *zfile*]**
[*infile*]

DESCRIPTION

dfs filters data from *infile* (or standard output) using a digital filter in standard form, sending the result to standard output. The filter transfer function is given by:

$$H(z) = K \frac{\sum_{n=0}^{N} b(n) z^{-n}}{1 + \sum_{m=1}^{M} a(m) z^{-m}}$$

Both input and output files are in float format.

OPTIONS

–a	$K\ a(1) \ldots a(M)$	denominator coefficients, where K is the gain of the transfer function.	[N/A]
–b	$b(0)\ b(1) \ldots b(N)$	numerator coefficients	[N/A]
–p	$pfile$	denominator coefficients file in float format as follows $K, a(1), \ldots, a(M)$	[NULL]
–z	$zfile$	numerator coefficients file in float format as follows $b(0), b(1), \ldots, b(N)$	[NULL]

If the option **–a** and **–p** specified, then both K and the denominator are set to 1. On the other hand, if the option **–b** and **–z** are not specified, then the numerator is set to 1.

EXAMPLE

In order to visualize the impulse response of the following transfer function

$$H(z) = \frac{1 + 2z^{-1} + z^{-2}}{1 + 0.9z^{-1}}$$

the command below can be used

```
impulse | dfs -a 1 0.9 -b 1 2 1 | dmp +f
```

For visualizing the frequency response plot of the digital filter, whose coefficients are defined in float format by the files *data.p, data.z*, then the following command can be used.

```
impulse | dfs -p data.p -z data.z | spec | fdrw | xgr
```

The files *data.p* and *data.z* can be obtained through the *x2x* command.

NAME

dmp – binary file dump

SYNOPSIS

dmp **[–n** *N* **] [–l** *L* **] [*+type*] [*%form*] [*infile*]**

DESCRIPTION

dmp converts data from *infile* (or standard input) to a human readable form, (one sample per line, with line numbers) and sends the result to standard output.

OPTIONS

–n	*N*	block order (0,...,n)	[EOD]
–l	*L*	block length (1,...,l)	[EOD]
+t		input data format	[f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)

%form	print format (printf style) '+' option must be placed in front of '%' option, without whitespace.	[N/A]

EXAMPLE

In this example, data is read from the input file *data.f* in float format, and the enumerated data is shown on the screen:

```
dmp +f data.f
```

For example, if the *data.f* file has the following values in float format

$$1, 2, 3, 4, 5, 6, 7$$

then the following output will be displayed on the screen:

```
0       1
1       2
2       3
3       4
4       5
```

```
5       6
6       7
```

In case one wants to assign a block length the following command can be used.

```
dmp +f -n 2 data.f
```

And the output will be given by:

```
0       1
1       2
2       3
0       4
1       5
2       6
0       7
```

Some other examples are provided bellow:

Print the unit impulse response of a digital filter on the screen:

```
impulse | dfs -a 1 0.9 | dmp +f
```

Print a sine wave using the %e option of *printf*:

```
sin -p 30 | dmp +f%e
```

Print the same sine wave represented by three decimal points:

```
sin -p 30 | dmp +f%.3e
```

SEE ALSO

x2x, fd

NAME

dtw – dynamic time warping

SYNOPSIS

dtw **[–m** *M* **]** **[–l** *L* **]** **[–t** *T* **]** **[–r** *R* **]** **[–n** *N* **]** **[–p** *P* **]**
[–s *Scorefile* **]** **[–v** *Vitfile* **]** ***reffile*** **[** ***infile*** **]**

DESCRIPTION

dtw carries out dynamic time warping between the test data vectors from *infile* (or standard input) and the reference data vectors from *reffile*, and sends the result to standard output. The result is the concatenated sequence of the test and the reference data vectors along with the Viterbi path. If –s option is specified, the score calculated by dynamic time warping, that is, the distance between the test data and the reference data is output and sent to *Scorefile*. If –v option is specified, the concatenated frame number sequence along the Viterbi path is output and sent to *Vitfile*.

For example, suppose that the test and the reference data vectors are

$$\begin{aligned} \text{test}: &\ \boldsymbol{x}(0), \boldsymbol{x}(1), \ldots, \boldsymbol{x}(T_x - 1), \boldsymbol{x}(T_x), \\ \text{reference}: &\ \boldsymbol{y}(0), \boldsymbol{y}(1), \ldots, \boldsymbol{y}(T_y - 1), \boldsymbol{y}(T_y), \end{aligned}$$

where T_x and T_y are the length of the test and reference data vectors, respectively,p and the following Viterbi sequences

$$\begin{aligned} \text{test}: &\ \boldsymbol{x}(\phi_x(0)), \boldsymbol{x}(\phi_x(1)), \ldots, \boldsymbol{x}(\phi_x(T_x - 1)), \boldsymbol{x}(\phi_x(T_x)), \\ \text{reference}: &\ \boldsymbol{y}(\phi_y(0)), \boldsymbol{y}(\phi_y(1)), \ldots, \boldsymbol{y}(\phi_y(T_y - 1)), \boldsymbol{y}(\phi_y(T_y)), \end{aligned}$$

are obtained, where $\phi_x(\cdot)$ and $\phi_x(\cdot)$ are the function which maps the frame number of test/reference data into the corresponding Viterbi frame number, respectively. In addition, the relation $\phi_x(T_x) = \phi_y(T_y)$ holds. Then, the following sequence

$$\boldsymbol{x}(\phi_x(0)), \boldsymbol{y}(\phi_y(0)), \boldsymbol{x}(\phi_x(1)), \boldsymbol{y}(\phi_y(1)), \ldots, \boldsymbol{x}(\phi_x(T_x)), \boldsymbol{y}(\phi_y(T_y))$$

are sent to the standard output. If –v option is specified, the following sequence

$$\phi_x(0), \phi_y(0), \phi_x(1), \phi_y(1), \ldots, \phi_x(T_x), \phi_y(T_y)$$

are sent to the *Vitfile*.

Both input and output files are in float format. However, the *Vitfile* which contains the Viterbi frame number sequence is in int format.

OPTIONS

–m	*M*	order of vector	[0]
–l	*L*	dimention of vector	[M+1]
–t	*T*	number of test vectors	[N/A]
–r	*R*	number of reference vectors	[N/A]
–n	*N*	type of norm used for calculation of local cost	[2]
		$N = 1$ L_1-norm $N = 2$ L_2-norm	
–p	*P*	local path constraint candidates of constraint are shown in figure 4.	[5]
–s	*Scorefile*	output score of the dynamic time warping to *Scorefile*.	[FALSE]
–v	*Vitfile*	output frame number sequence along the Viterbi path to *Vitfile*.	[FALSE]

EXAMPLE

In the example below, a dynamic time warping between the scalar sequence from *data.test* and the sequence from *data.ref* is carried out and the concatenated sequence are written to *data.out*.

```
dtw -l 1 data.ref < data.test > data.out
```

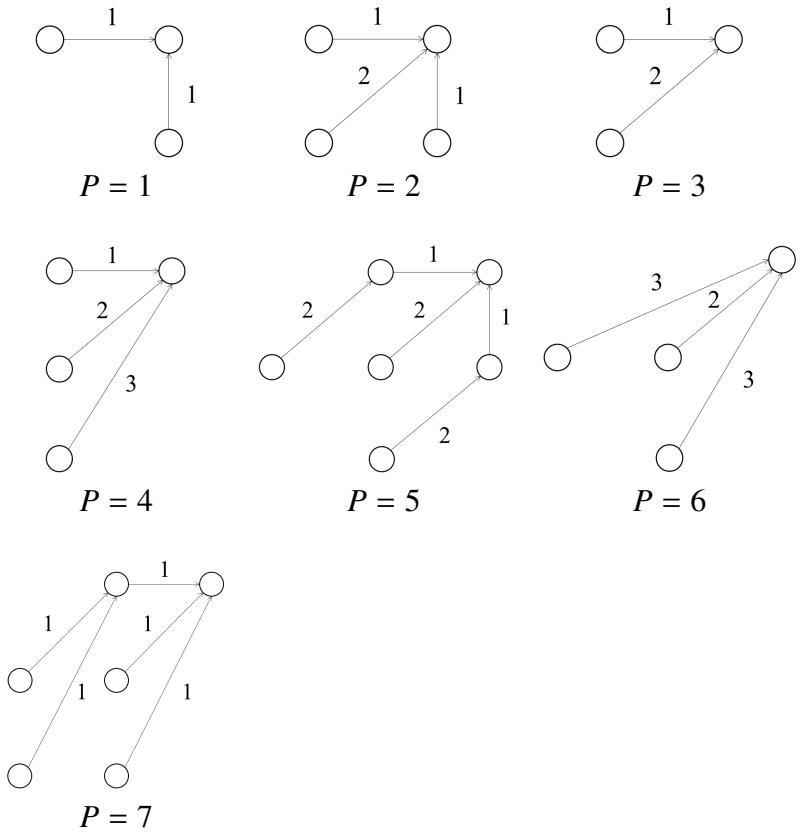


Figure 4: candidates of local path constraint

NAME

ds – down-sampling

SYNOPSIS

ds **[–s** *S* **]** **[** ***infile*** **]**

DESCRIPTION

ds down-samples data from *infile* (or standard input), and sends the result to standard output.

Both input and output files are in float format.

The following filter coefficients can be used.

$S = 21$	\$SPTK/share/SPTK/lpfcoef.2to1
$S = 32$	\$SPTK/share/SPTK/lpfcoef.3to2
$S = 43$	\$SPTK/share/SPTK/lpfcoef.4to3
$S = 52, s = 54$	\$SPTK/share/lpfcoef.5to2up \$SPTK/share/lpfcoef.5to2dn (\$SPTK is the directory where toolkit was installed.)

Filter coefficients are in ASCII format.

OPTIONS

–s *S* conversion type [21]

$S = 21$	down-sampling by 2 : 1
$S = 32$	down-sampling by 3 : 2
$S = 43$	down-sampling by 4 : 3
$S = 52$	down-sampling by 5 : 2
$S = 54$	down-sampling by 5 : 4

EXAMPLE

In this example, the speech data in the input file *data.16*, which was sampled at 16 kHz in float format, is downsampled to 8 kHz:

```
ds data.16 > data.8
```

SEE ALSO

us, uscd, us16

NAME

echo2 – echo arguments to the standard error

SYNOPSIS

echo2 [–n] [argument]

DESCRIPTION

echo2 sends its command line arguments to standard error.

OPTIONS

–n	no output newline	[FALSE]

EXAMPLE

This example prints ”error!” in the standard error output:

```
echo2 -n "error!"
```

NAME

excite – generate excitation

SYNOPSIS

excite [–p *P* **] [–i** *I* **] [–n] [–s** *S* **] [** ***infile*** **]**

DESCRIPTION

excite generates an excitation sequence from the pitch period information in *infile* (or standard input), and sends the result to standard output. When the pitch period is nonzero (i.e. voiced), the excitation sequence consists of a pulse train at that pitch. When the pitch period is zero (i.e. unvoiced), the excitation sequence consists of Gaussian or M-sequence noise.

Input and output data are in float format.

OPTIONS

–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–n		gauss/M-sequence for unvoiced default is M-sequence	[FALSE]
–s	*S*	seed for nrand for Gaussian noise	[1]

EXAMPLE

In the example below, the excitation is generated from the *data.p* file and passed through a LPC synthesis filter whose coefficients are in the *data.lpc* file. The speech signal is outputted to the *data.syn* file.

```
excite < data.p | poledf data.lpc > data.syn
```

The following command can be used for generating an unvoiced sound by using Gaussian noise:

```
excite -n < data.p | poledf data.lpc > data.syn
```

SEE ALSO

poledf

NAME

extract – extract vector

SYNOPSIS

extract [–l *L* **] [–i** *I* **]** ***indexfile*** **[** ***infile*** **]**

DESCRIPTION

extract extracts selected vectors from *infile* (or standard input), and sends the result to standard output. *indexfile* contains a previously-computed sequence of codebook indexes corresponding to the input vectors. Only those input vectors whose codebook index (from *indexfile*) matches the index given by the "–i" option are sent to the standard output.

OPTIONS

–l	*L*	order of vector	[10]
–i	*I*	codebook index	[0]

EXAMPLE

In the example below, a 10-th order vector file *data.v* in float format is quantized using a previously obtained codebook *data.idx* and are written to the output file *data.ex* quantized to the index 0 codeword.

```
extract -i 0 data.idx data.v > data.ex
```

SEE ALSO

ivq, vq

NAME

fd – file dump

SYNOPSIS

fd **[–a** *A* **] [–n** *N* **] [–m** *M* **] [** ***–ent*** **] [** ***+type*** **] [** ***%form*** **] [** ***infile*** **]**

DESCRIPTION

fd converts data from *infile* (or standard input) to a human-readable multi-column format, and sends the result to standard output.

OPTIONS

–a	*A*	address	[0]
–n	*N*	initial value for numbering	[0]
–m	*M*	modulo for numbering	[EOF]
–ent		number of data in each line	[0]
+t		data type	[c]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)

%form print format (printf style) [N/A]
'+' option must be placed in front of '%' option, without whitespace.

EXAMPLE

This example displays the speech data in "sample.wav" with the corresponding addresses:

```
fd +c -a 0 sample.wav
```

Results:

```
000000  52 49 46 46 9a 15 00 00 57 41 56 45 66 6d 74 20 |RIFF....WAVEfmt |
000010  10 00 00 00 01 00 01 00 40 1f 00 00 40 1f 00 00 |........@...@...|
000020  01 00 08 00 64 61 74 61 76 15 00 00 8a 8a 8f 99 |....datav.......|
```

⋮

SEE ALSO

dmp

NAME

fdrw – draw a graph

SYNOPSIS

fdrw **[–F** *F* **] [–R** *R* **] [–W** *W* **] [–H** *H* **] [–o** *xo yo* **] [–g** *G* **] [–m** *M* **]**
[–l *L* **] [–p** *P* **] [–j** *J* **] [–n** *N* **] [–t** *T* **] [–y** *ymin ymax* **] [–z** *Z* **] [–b]**
[***infile*** **]**

DESCRIPTION

fdrw converts float data from *infile* (or standard input) to a plot formatted according to the FP5301 protocol, and sends the result to standard output. One can control the details of the plot layout by setting the options bellow:

OPTIONS

–F	*F*	factor	[1]
–R	*R*	rotation angle	[0]
–W	*W*	width of figure (×100 mm)	[1]
–H	*H*	height of figure (×100 mm)	[1]
–o	*xo yo*	origin in mm	[20 25]
–g	*G*	draw grid (0 ∼ 2) (see also fig)	[1]
–m	*M*	line type (1 ∼ 5) 1: solid 2: dotted 3: dot and dash 4: broken 5: dash	[0]
–l	*L*	line pitch	[0]
–p	*P*	pen number (1 ∼ 10)	[1]
–j	*J*	join number (0 ∼ 2)	[1]
–n	*N*	number of samples	[0]
–t	*T*	rotation of coordinate axis. When $T = -1$, the reference point is on the top-left. When $T = 1$ the reference point is on the bottom-right.	[0]
–y	*ymin ymax*	scaling factor for *y* axis	[-1 1]
–z	*Z*	This option is used when data is written recursively in the *y* axis. The distance between two graphs in the *y* axis is given by *Z*.	[0]
–b		bar graph mode	[FALSE]

The *x* axis scaling is automatically done so that every point in the input file is plotted in equally spaced interrals for the assigned width. When the **–n** option is omitted and the number of input samples is below 5000, then the block size is made equal to the number of samples. When the number of samples is above 5000, then the block size is made equal to 5000.
When the **–y** option is omitted, the input data minimum value is set to *ymin* and the maximum value is set to *ymax*.

EXAMPLE

In the example below, the impulse response of a digital filter is drawn on the X window environment:

```
impulse | dfs -a 1 0.8 0.5 | fdrw -H 0.3 | xgr
```

The graph width is 10cm and its height is 3cm.

The next example draws the magnitude of the frequency response of a digital filter on the X window environment:

```
impulse | dfs -a 1 0.8 0.5 | spec | fdrw -y -60 40 | xgr
```

The *y* axis goes from −60 dB to 40 dB.

The running spectrum can be draw on the X window environment by:

```
fig -g 0 -W 0.4 << EOF
~~~~x 0 5
~~~~xscale 0 1 2 3 4 5
~~~~xname "FREQUENCY (kHz)"
EOF
spec < data |\
fdrw -W 0.4 -H 0.2 -g 0 -n 129 -y -30 30 -z 3 |\
xgr
```

The command *psgr* prints the output to a laser printer in the same manner as it is printed on the screen. Since the *fdrw* command includes a sequence of commands for a plotter machine (FP5301 protocol) in the output file, its output can be directly sent to a printer.

SEE ALSO

fig, xgr, psgr

NAME

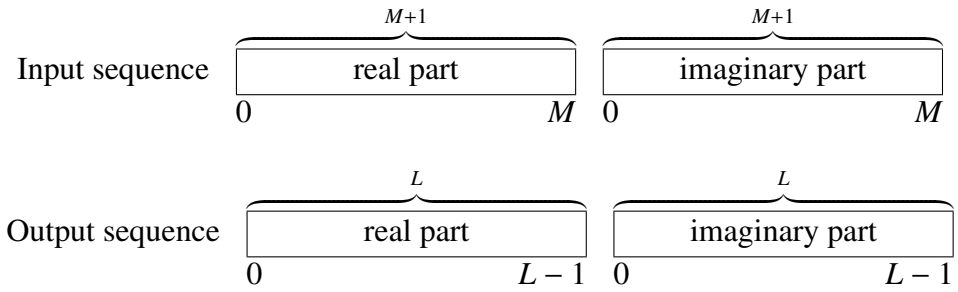
fft – FFT for complex sequence

SYNOPSIS

fft [–l *L* **] [–m** *M***] [–{ A | R | I | P }] [** ***infile*** **]**

DESCRIPTION

fft uses the Fast Fourier Transform (FFT) algorithm to calculate the Discrete Fourier Transform (DFT) of complex-valued input data from *infile* (or standard input), and sends the result to standard output. The input and output data is in float format, and arranged as follows.



OPTIONS

–l	*L*	FFT size power of 2	[256]
–m	*M*	order of sequence	[L-1]
–A		amplitude	[FALSE]
–R		real part	[FALSE]
–I		imaginary part	[FALSE]
–P		output power spectrum	[FALSE]

EXAMPLE

This example reads a sequence of complex numbers in float format from *data.f* file (real part with 256 points and imaginary part with 256 points), evaluates its DFT and outputs it to the *data.dft* file:

```
fft data.f -l 256 -A > data.dft
```

SEE ALSO

fftr, spec, phase

NAME

fft2 – 2-dimensional FFT for complex sequence

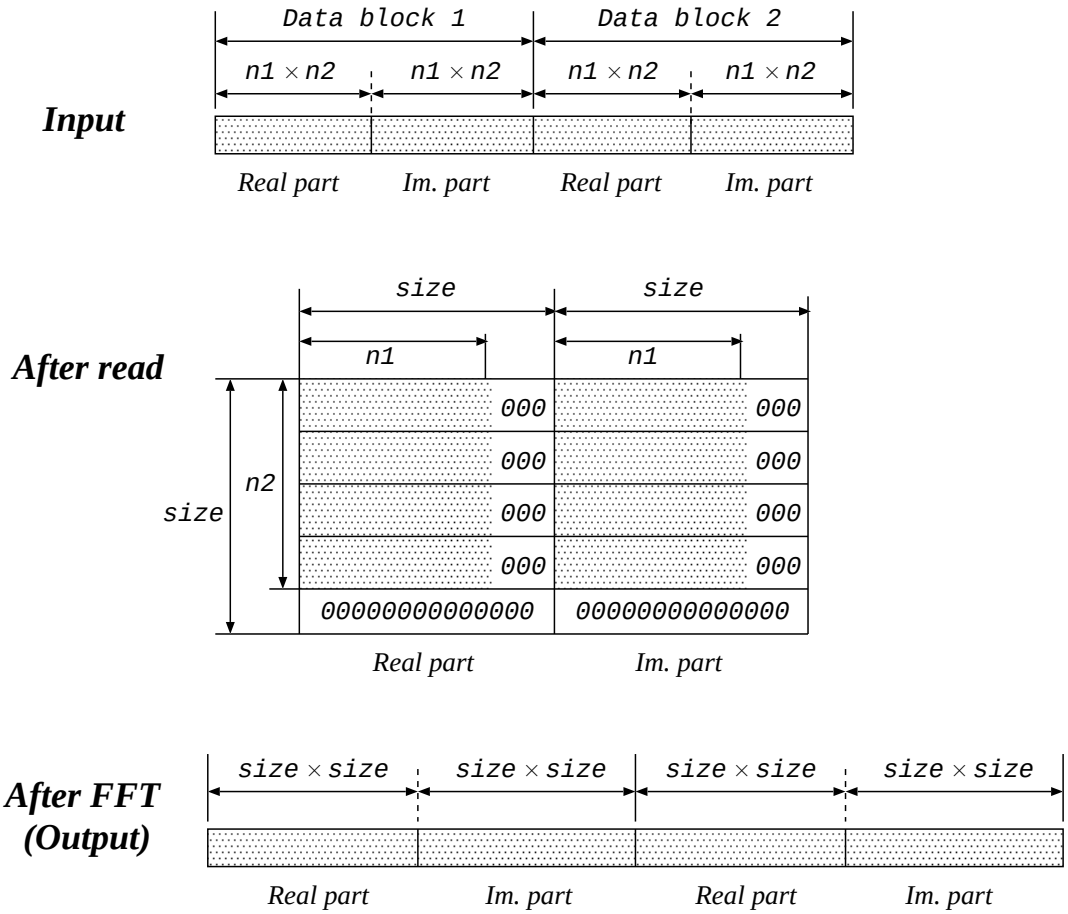
SYNOPSIS

fft2 **[–l** *L* **] [–m** M_1 M_2 **] [–t] [–c] [–q] [–{ A | R | I | P }]**

[***infile*** **]**

DESCRIPTION

fft2 uses the 2-dimensional Fast Fourier Transform (FFT) algorithm to calculate the 2-dimensional Discrete Fourier Transform (DFT) of complex-valued input data from *infile* (or standard input), and sends the result to standard output. The input and output data is in float format, arranged as follows.

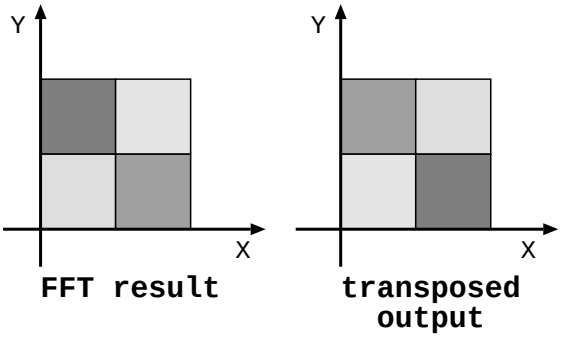


OPTIONS

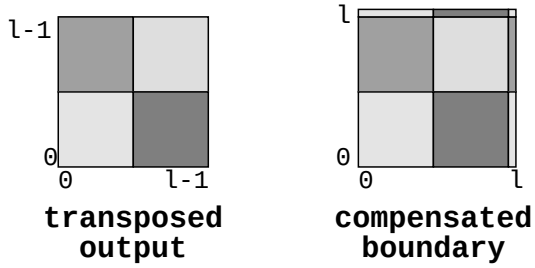
–l *L* FFT size power of 2 [64]

–m M_1 M_2 order of sequence ($M_1 \times M_2$). If file size k is smaller than $64^2 \times 2$ and $\sqrt{k \div 2}$ is an integer value, $M_1 = M_2 = \sqrt{k \div 2}$. Otherwise, an output error message is sent to standard error output and the command is terminated. [64, M_1]

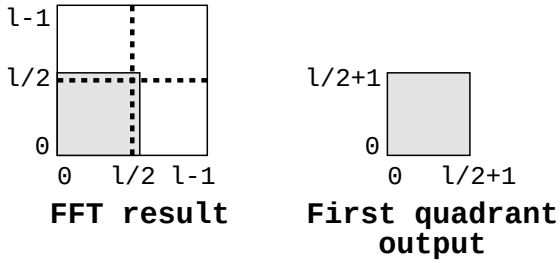
–t Output results in transposed form. [FALSE]



–c When results are transposed, 1 boundary data is copied from the opposite side, and then $(L + 1) \times (L + 1)$ data is outputted. [FALSE]



–q Output first 1/4 data of FFT results only. As in the above c option, boundary data is compensated and $(\frac{L}{2} + 1) \times (\frac{L}{2} + 1)$ data is outputted. [FALSE]



–A amplitude [FALSE]

–R	real part	[FALSE]
–I	imaginary part	[FALSE]
–P	output power spectrum	[FALSE]

EXAMPLE

This example reads a sequence of 2-dimensional complex numbers in float format from *data.f* file, evaluates its 2-dimensional DFT and outputs it to *data.dft* file:

```
fft2 -A data.f > data.dft
```

SEE ALSO

fft, fftr2, ifft

NAME

fftcep – FFT cepstral analysis

SYNOPSIS

fftcep [–m *M* **] [–l** *L* **] [–j** *J* **] [–k** *K* **] [–e** *E* **] [** ***infile*** **]**

DESCRIPTION

fftcep uses FFT cepstral analysis to calculate the cepstrum from windowed framed input data in *infile* (or standard input), sending the result to standard output. The windowed input time domain sequence of length L is of the form:

$$x(0), x(1), \ldots, x(L-1)$$

Input and output data are in float format.

Also, the improved cepstral analysis method (1) may be used if the number of iterations J and the acceleration factor K are given.

OPTIONS

–m	*M*	order of cepstrum	[25]
–l	*L*	frame length	[256]
–j	*J*	number of iteration	[0]
–k	*K*	acceleration factor	[0.0]
–e	*E*	epsilon	[0.0]

EXAMPLE

In the example below, speech data in float format is read from *data.f* and the cepstral coefficients are output to *data.cep*:

```
frame < data.f | window | fftcep > data.cep
```

SEE ALSO

uels

NAME

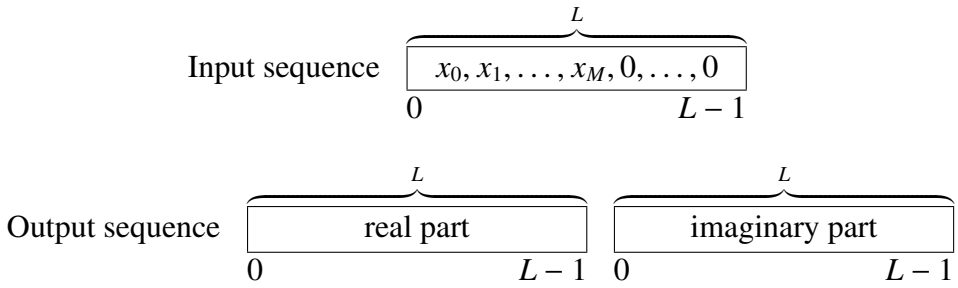
fftr – FFT for real sequence

SYNOPSIS

fftr [–l *L* **] [–m** *M***] [–{ A | R | I | P }] [–H] [** ***infile*** **]**

DESCRIPTION

fftr uses the Fast Fourier Transform (FFT) algorithm to calculate the Discrete Fourier Transform (DFT) of real-valued input data in *infile* (or standard input), and sends the result to standard output. When the –m option is omitted and the input data sequence length is less than the FFT size, the input data is padded with zeros. The input and output data is in float format, arranged as below.



OPTIONS

–l	*L*	FFT size power of 2	[256]
–m	*M*	order of sequence	[L-1]
–A		output magnitude	[FALSE]
–R		output real part	[FALSE]
–I		output imaginary part	[FALSE]
–P		output power spectrum	[FALSE]
–H		output half size	[FALSE]

EXAMPLE

In the example below, a sine wave is passed through a Blackman window, its DFT is evaluated and the magnitude is plotted:

```
sin -p 30 | window | fftr -A | fdrw | xgr
```

SEE ALSO

fft, fft2, fftr2, ifft ifftr ifft2 spec, phase

NAME

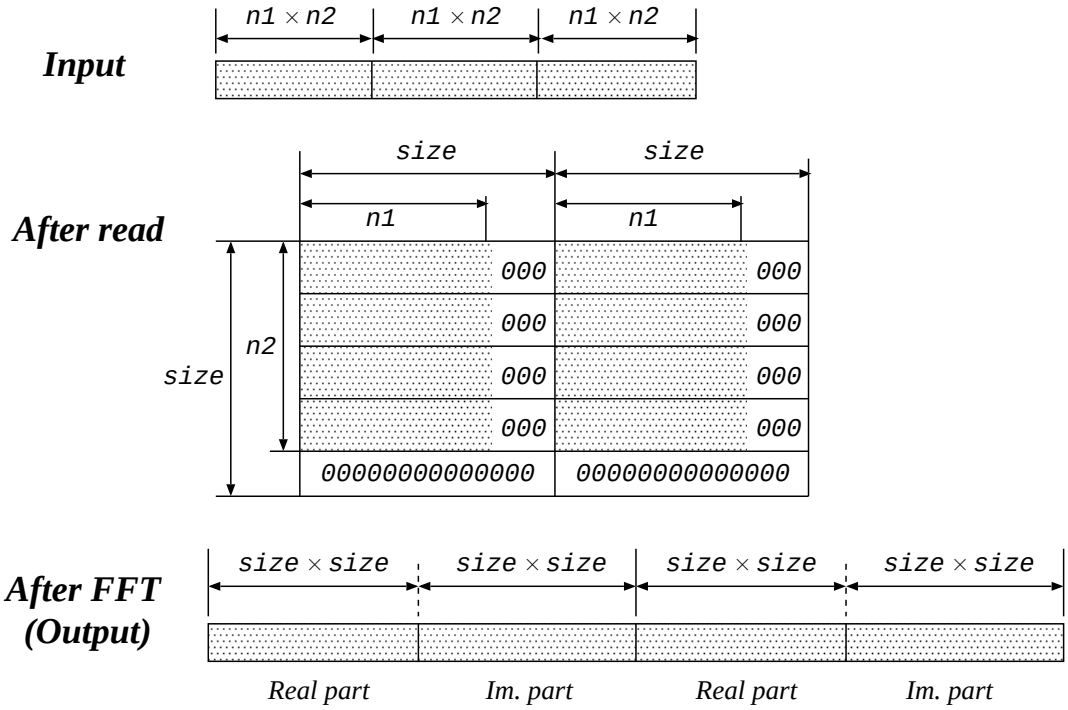
fftr2 – 2-dimensional FFT for real sequence

SYNOPSIS

fftr2 **[–l** *L* **] [–m** M_1 M_2 **] [–t] [–c] [–q] [–{ A | R | I | P }] [** ***infile*** **]**

DESCRIPTION

fftr2 uses the 2-dimensional Fast Fourier Transform (FFT) algorithm to calculate the 2-dimensional Discrete Fourier Transform (DFT) of real-valued input data in *infile* (or standard input), and sends the result to standard output. The input and output data is in float format, arranged as follows.



OPTIONS

–l	*L*	FFT size power of 2	[64]
–m	M_1 M_2	order of sequence ($M_1 \times M_2$). If the file size k is smaller than 64^2 and $\sqrt{k}$ is an integer value, then $M_1 = M_2 = \sqrt{k}$. Otherwise, output error message is sent to standard error output and then the command terminates.	[64, M_1]
–t		Output results in transposed form (see also fft2).	[FALSE]
–c		When results are transposed, 1 boundary data is copied from the opposite side, and then data whose size is $(L + 1) \times (L + 1)$ is output. (see also fft2).	[FALSE]

–q	Output first 1/4 data of FFT results only. As in –c option, boundary data is compensated and data whose size is $(\frac{L}{2} + 1) \times (\frac{L}{2} + 1)$ is output (see also fft2).	[FALSE]
–A	amplitude	[FALSE]
–R	real part	[FALSE]
–I	imaginary part	[FALSE]
–P	output power spectrum	[FALSE]

EXAMPLE

This example reads a sequence of 2-dimensional real numbers in float format from *data.f* file, evaluates its 2-dimensional DFT and outputs results to *data.dft* file:

```
fftr2 -A data.f > data.dft
```

SEE ALSO

fft, fft2, fftr, ifft ifft2 ifftr

NAME

fig – plot a graph

SYNOPSIS

fig **[–F** *F* **] [–R** *R* **] [–W** *W* **] [–H** *H***] [–o** *xo yo* **] [–g** *G* **] [–p** *P* **] [–j** *J* **]**
[–s *S* **] [–f** *file* **] [–t] [** ***infile*** **]**

DESCRIPTION

fig draws a graph using information from *infile* (or standard input), sending the result in FP5301 plot format to standard output. This command is similar to the Unix command "graph" but includes some labeling functions. The output can be printed directly on a printer that supports the FP5301 protocol, displayed on an X11 display with the xgr command, or converted to PostScript format with the psgr command.

OPTIONS

–F	*F*	factor	[1]
–R	*R*	rotation angle	[0]
–W	*W*	width of figure (×100mm)	[1]
–H	*H*	height of figure (×100mm)	[1]
–o	*xo yo*	origin in mm	[20 20]
–g	*G*	draw grid (0 ~ 2)	[2]
–p	*P*	pen number (1 ~ 10)	[1]
–j	*J*	join number (0 ~ 2)	[0]
–s	*S*	font size (1 ~ 4)	[1]
–f	*file*	The file assigned after this option is read before *infile*, that is, this option gives preference.	[NULL]
–t		transpose *x* and *y* axes	[FALSE]

EXAMPLE

In the example below, data in *data.fig* file is plotted in an X terminal:

```
fig data.fig |xgr
```

In this example, data in *data.fig* file is converted to postscript format and visualized with ghostview:

```
fig data.fig | psgr | ghostview -
```

USAGE

COMMAND

The input data file can contain commands and data. Commands can be used for labeling, scaling, etc. Data is written in the (x y) coordinate pair form. Command values can be overwritten by entering new command values.

COMMAND LINES

x [mel α] *xmin xmax* [*xa*]
y [mel α] *ymin ymax* [*ya*]

Assigns x and y scalings. Marks can be specified in x and y axes through *xa* and *ya*. If no setting of *xa* and *ya* is done, then *xa* is set to *xmin* and *ya* to *ymin*. If the optional "mel α", where α must be a number (for example, mel 0.35), is used, then labeling is undertaken as a frequency transformation of a minimum phase first order all-pass filter.

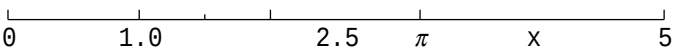
xscale x_1 x_2 x_3 ...
yscale y_1 y_2 y_3 ...

Assigns values to the points $x_1, x_2, x_3, \ldots$ and $y_1, y_2, y_3, \ldots$ in x and y axes. These points can be assigned with numbers or marks, Also, when one wants to specify points which consist of numeric and non-numeric characters all together (like in '2,*.3.14), then the following function should be used:

s	draws marks with half size.
\	only writes number.
@	does not write anything but assigns positions of marks.
none of the above	only marks are written.

Whenever the character is inside quotes, it appears in the position assigned by the string that precedes it. Please refer to the commands x/yname for information on special characters.
(Example)

```
x 0 5
xscale 0 1.0 s1.5 '2 \2.5 '3.14 "\pi" @4 "x" 5
```

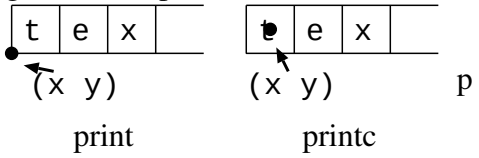


xname "*text*"
yname "*text*"

Labels x and y axes. *text* should appear between the quotes. Within *text*, TEXcommands can be used. Also, characters, such as those that can be obtained with TEX, can be written with this command.

print x y "*text*" [*th*]
printc x y "*text*" [*th*]

This command writes *text* in the position (x y) assigned. The option *th* sets the rotation degree.



title x y "*text*" [*th*]
titlec x y "*text*" [*th*]

This command does the same as print(c). However, the basic unit is expressed in the mm, evaluated as absolute value. The reference point is on the bottom-left side.

csize h [w]

This command sets the character width and height (in mm), to be used in the following commands:
x/yscale, x/yname, print/c, title/c
When the value of *w* is omitted, *w* is made equal to *h*. The default values for the option **–s** are as follows:

–s	w	h
1	2.5	2.2
2	5	2.6
3	2.5	4.4
4	5	4.4

pen *penno*

This command chooses the variable *penno*. $1 \leq penno \leq 10$ Please refer to appendix.

join *joinno*

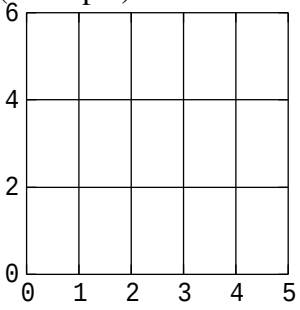
This command chooses the variable *joinno*. $0 \leq joinno \leq 2$ Please refer to the appendix.

line *ltype* [*lpt*]

This command sets the type *ltype* of the line which will connect data as well as the *lpt* pace. *lpt* is in mm. When *ltype*=0: no line is used to connect coordinate points. 1: solid 2: dotted 3: dot and dash 4: broken 5: dash Please refer to the appendix.

xgrid x_1 x_2 ...
ygrid y_1 y_2 ...

This command causes grids to be drawn in the positions x_1 x_2 ..., y_1 y_2 ....
(Example)




```
x 0 5
y 0 6
xscale 0 1 2 3 4 5
yscale 0 2 4 6

xgrid 1 2 3 4
ygrid 2 4
```

mark *label* [*th*]

This command draws a mark in the assigned coordinate position. The option *th* specifies the angle(degree) in which the string will be draw. If *label* is assigned with \0, the mark is released. A detailed explanation on writing marks and special characters to graphs is provided at the label section.

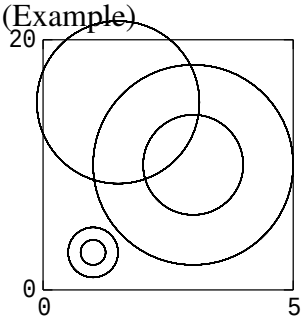
height h [w]
italic *th*

The height command defines the size of the label through its height h(mm) and width w(mm). The labels may also be written in italic by using the italic command.

circle x y r_1 r_2 ...
xcircle x y r_1 r_2 ...
ycircle x y r_1 r_2 ...

These commands write circles with radius r_1 r_2 ... and center on the coordinate (x, y). Also, the radius r_x is given in mm. As for the xcircle and ycircle commands, the units considered for the radius are the scales of the x axis and y axis, respectively, as shown in the figure below.

(Example)



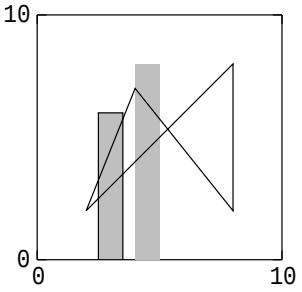

```
x 0 5
y 0 20
xscale 0 5
yscale 0 20

xcircle 3 10 1 2
ycircle 1 3 1 2
circle 1.5 15 13
```

box x_0 y_0 x_1 y_1 [x_2 y_2 ...]
paint *type*

This command draws a rectangle with paint *type* connecting (x_0 y_0) and (x_1 y_1) through a solid line. The line which connects (x_0 y_0) and (x_1 y_1) forms the diagonal of the rectangle. Also, if x_2 y_2 ... are assigned, a polygon is draw connecting the points (x_0 y_0),(x_1 y_1),(x_2 y_2),.... In this case, Please do not set the paint *type* to any value different from the default. The default value is 1.

(Example)



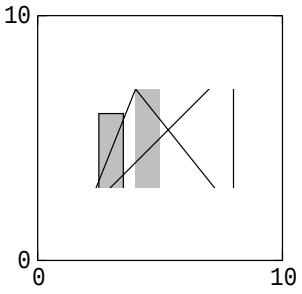

```
x 0 10
y 0 10
xscale 0 10
yscale 0 10

paint 18
box 2.5 0 3.5 6
paint -18
box 4 0 5 8
paint 1
box 2 2 8 8 8 2 4 7
```

clip x_0 y_0 x_1 y_1

This command allows for drawing only inside the box defined by (x_0 y_0), (x_1 y_1). When the coordinates (x_0 y_0), (x_1 y_1) are omitted, then the clip command is skipped.

(Example)




```
x 0 10
y 0 10
xscale 0 10
yscale 0 10

clip 2 3 9 7
paint 18
box 2.5 0 3.5 6
paint -18
box 4 0 5 8
paint 1
box 2 2 8 8 8 2 4 7
```

any comment

This is used for writing comment lines. Whatever is written after the symbol # is ignored by the fig command.

DATA LINES

x y [*label* [*th*]]

The coordinates (x y) are scaled by the values specified in the command line. If a string is written to *label*, then it will be written in the (x y) position. There should be no empty characters (e.g., space) in the beginning of the label setting. When *label* is given in the mark command, the *label* replacement will take place only for this coordinate. The option *th* assigns the angle.
If \n, where $0 \leq n \leq 15$, is assigned to *label*, the corresponding mark is draw (refer to the appendix for the types of marks). When a minus sign is written before mark number, then the connecting line between marks passes through the center of each mark.
If a minus sign is not included, then connecting lines do not pass through the center of each mark. When $n = 16$(\16), a small circle is written with diameter defined by the hight command. Also, special character and ASCII character can be written through code number when $n > 32$.

eod
EOD

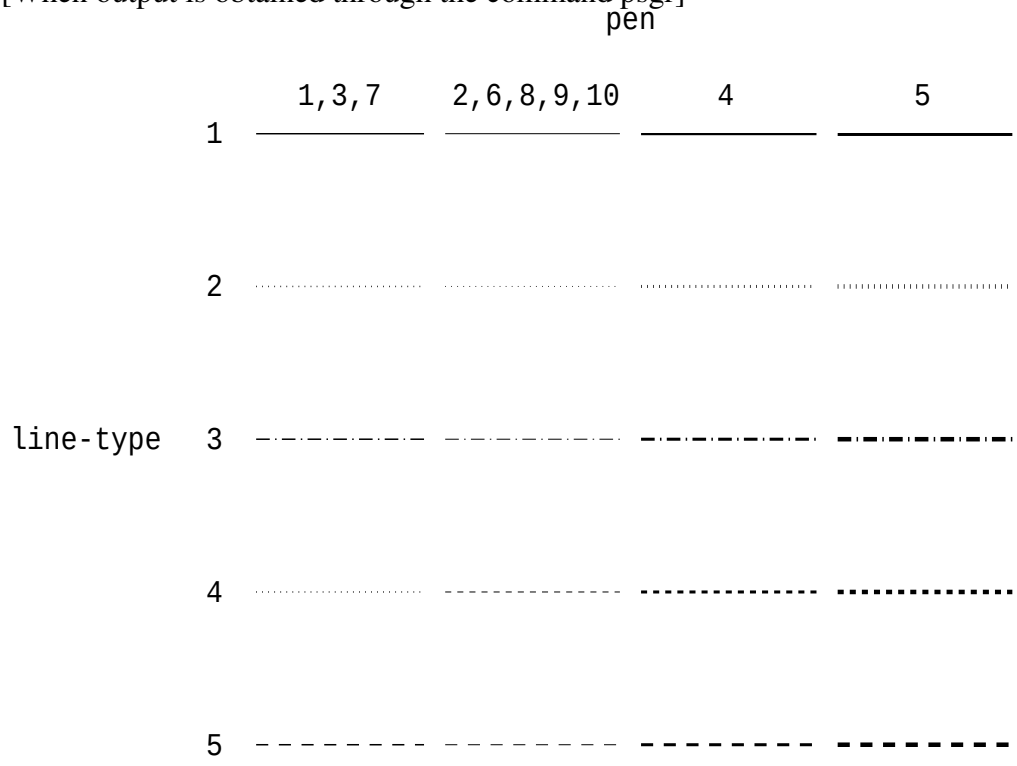
This is the end of data sign. Coordinates before and after the eod sign are not connected.

APPENDIX

- The following type of marks can be defined through *label*:

0	1	2	3	4	5	6	7
	·	×	□	△	○	◊	×
8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
+	⊗	⊕	■	▲	●	♦	*

- The following types of pen and line can be defined:
  [When output is obtained through the command psgr]



ps: The types of output generated by the pen command depend on the printer (Please try printing this page).

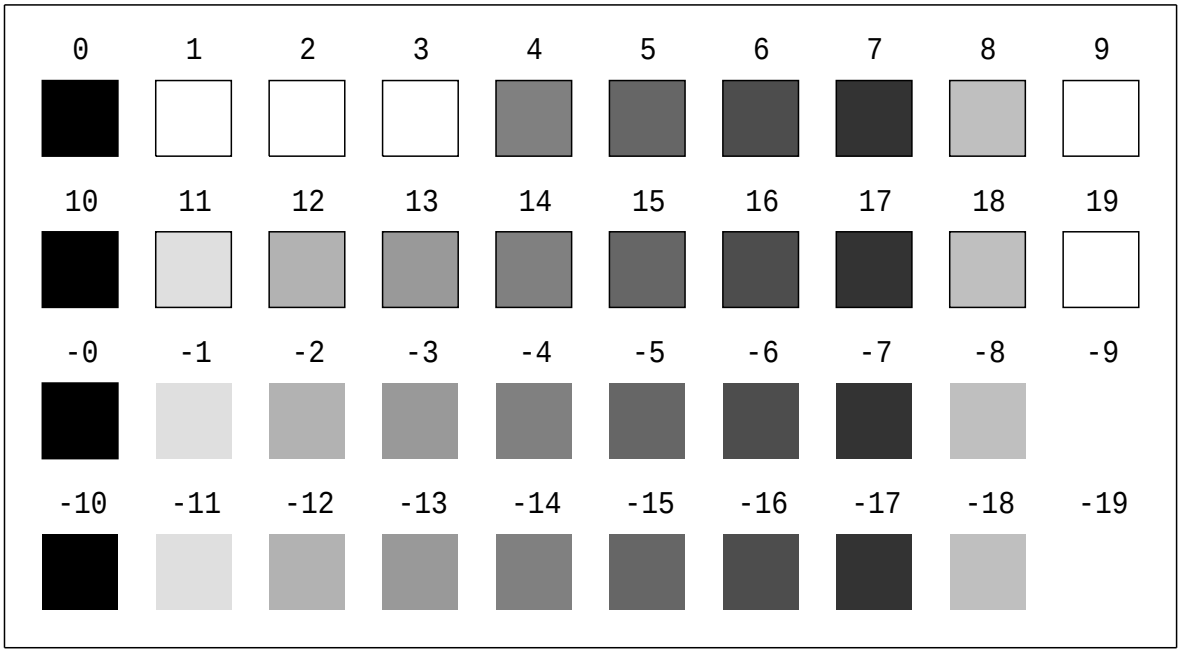
[When output is obtained through the command xgr]
The following colors can be used.

pen type	1	2	3	4	5	6	7	8	9	10
color	black	blue	red	green	pink	orange	emerald	gray	brown	dark blue

- The following types of joins can be defined:

join type	0 Miter join	1 Round join	2 Bevel join
example			

- paint type:



ps: From $1 \sim 3$ only a frame is draw, and for -9 and -19 the center is white and no frame is draw.

NAME

frame – extract frame from data sequence

SYNOPSIS

frame **[–l** *L* **] [–n] [–p** *P* **] [** ***infile*** **]**

DESCRIPTION

frame converts a sequence of input data from *infile* (or standard input) to a series of possibly-overlapping frames with period P and length L, and sends the result to standard output. If the input data is $x(0), x(1), \ldots, x(T)$, then the output data will be given by :

$$\begin{array}{ccccccccc} 0 & , & 0 & , & \ldots & , & x(0) & , \ldots , & x(L/2) \\ x(P-L/2) & , & x(P-L/2+1) & , & \ldots & , & x(P) & , \ldots , & x(P+L/2) \\ x(2P-L/2) & , & x(2P-L/2+1) & , & \ldots & , & x(2P) & , \ldots , & x(2P+L/2) \\ & & & & & & \vdots & & \end{array}$$

OPTIONS

–l	*L*	frame length	[256]
–p	*P*	frame period	[100]
–n		This option is used when, instead of having x(0) as the center point in the first frame, one want to make x(0) as the first point of the first frame	[FALSE]

EXAMPLE

In the example below, data is read from *data.f* file, The frame period is of 80 and Blackman window is used. Also, linear prediction analysis is applied. The output is written in *data.lpc* file:

```
frame -p 80 < data.f | window | lpc > data.lpc
```

SEE ALSO

bcp, x2x, bcut, window

NAME

freqt – frequency transformation

SYNOPSIS

freqt [**–m** M_1] [**–M** M_2] [**–a** A_1] [**–A** A_2] [***infile***]

DESCRIPTION

freqt converts a M_1-th order minimum phase sequence from *infile* (or standard input) into a frequency-transformed M_2-th order sequence, sending the result to standard output.

Given the input sequence

$$c_{\alpha_1}(0), c_{\alpha_1}(1), \ldots, c_{\alpha_1}(M_1)$$

the frequency transform is given by:

$$\alpha = (\alpha_1 - \alpha_2)/(1 - \alpha_1\alpha_2)$$

$$c^{(i)}_{\alpha_2}(m) = \begin{cases} c_{\alpha_1}(-i) + \alpha\, c^{(i-1)}_{\alpha_2}(0) & m = 0 \\ (1-\alpha^2)\, c^{(i-1)}_{\alpha_2}(0) + \alpha\, c^{(i-1)}_{\alpha_2}(1) & m = 1 \\ c^{(i-1)}_{\alpha_2}(m-1) + \alpha \left(c^{(i-1)}_{\alpha_2}(m) - c^{(i)}_{\alpha_2}(m-1) \right) & m = 2, \ldots, M_2 \end{cases}$$
$$i = -M_1, \ldots, -1, 0 \qquad (1)$$

And the M_2-th order frequency transformed output sequence is of the form:

$$c^{(0)}_{\alpha_2}(0), c^{(0)}_{\alpha_2}(1), \ldots, c^{(0)}_{\alpha_2}(M_2)$$

Input and output data are in float format.

OPTIONS

–m	M_1	order of minimum phase sequence	[25]
–M	M_2	order of warped sequence	[25]
–a	A_1	all-pass constant of input sequenceα_1	[0]
–A	A_2	all-pass constant of output sequenceα_2	[0.35]

EXAMPLE

In the following example, the linear prediction coefficients in float format are read from *data.lpc* file, transformed in 30-th order LPC mel-cepstral coefficients, and written in *data.lpcmc* file:

```
lpc2c < data.lpc | freqt -m 30 > data.lpcmc
```

SEE ALSO

mgc2mgc

NAME

gc2gc – generalized cepstral transformation

SYNOPSIS

gc2gc **[–m** M_1 **]** **[–g** G_1 **]** **[–c** C_1 **]** **[–n]** **[–u]**
[–M M_2 **]** **[–G** G_2 **]** **[–C** C_2 **]** **[–N]** **[–U]** **[** ***infile*** **]**

DESCRIPTION

gc2gc uses a regressive equation to transform a sequence of generalized cepstral coefficients with power parameter γ_1 from *infile* (or standard input) into generalized cepstral coefficients with power parameter γ_2, sending the result to standard output.

Input and output data are in float format.

The regressive equation for the generalized cepstral coefficients is as follows.

$$c_{\gamma_2}(m) = c_{\gamma_1}(m) + \sum_{k=1}^{m-1} \frac{k}{m}(\gamma_2 c_{\gamma_1}(k)c_{\gamma_2}(m-k) - \gamma_1 c_{\gamma_2}(k)c_{\gamma_1}(m-k)), \qquad m > 0.$$

For the above equation, in case $\gamma_1 = -1, \gamma_2 = 0$, then LPC cepstral coefficients are obtained from the LPC coefficients, in case $\gamma_1 = 0, \gamma_2 = 1$, a minimum phase impulse response is obtained from the cepstral coefficients.

If the coefficients $c_\gamma(m)$ have not been normalized, then the input and output will be represented by

$$1 + \gamma c_\gamma(0), \gamma c_\gamma(1), \ldots, \gamma c_\gamma(M)$$

The following applies to the case the coefficients are normalized,

$$K_\alpha, \gamma c'_\gamma(1), \ldots, \gamma c'_\gamma(M)$$

OPTIONS

–m	M_1	order of generalized cepstrum (input)	[25]
–g	G_1	gamma of generalized cepstrum (input) $\gamma_1 = G_1$	[0]
–c	C_1	gamma of generalized cepstrum (input) $\gamma_1 = -1/(\text{int})C_1$ C_1 must be $C_1 \geq 1$	
–n		regard input as normalized cepstrum	[FALSE]
–u		regard input as multiplied by γ_1	[FALSE]
–M	M_2	order of generalized cepstrum (output)	[25]
–G	G_2	gamma of generalized cepstrum (output) $\gamma_2 = G_2$	[1]
–C	C_2	gamma of mel-generalized cepstrum (output) $\gamma_2 = -1/(\text{int})G_2$ C_2 must be $C_2 \geq 1$	

–N	regard output as normalized cepstrum	[FALSE]
–U	regard output as multiplied by γ_1	[FALSE]

EXAMPLE

In the following example, generalized cepstral coefficients with $M = 10$ and $\gamma_1 = -0.5$ are read in float format from *data.gcep* file, transformed into 30-th order cepstral coefficients, and written to *data.cep*:

```
gc2gc -m 10 -c 2 -M 30 -G 0 < data.gcep > data.cep
```

SEE ALSO

gcep, mgcep, freqt, mgc2mgc, lpc2c

NAME

gcep – generalized cepstral analysis(6; 7; 8)

SYNOPSIS

gcep **[–m** *M* **] [–g** *G* **] [–c** *C* **] [–l** *L* **] [–q** *Q* **] [–n] [–i** *I* **] [–j** *J* **] [–d** *D* **]**
[–e *e* **] [–E** *E* **] [–f** *F* **] [** ***infile*** **]**

DESCRIPTION

gcep uses generalized cepstral analysis to calculate normalized cepstral coefficients $c'_\gamma(m)$ from *L*-length framed windowed input data from *infile* (or standard input), sending the result to standard output. The windowed input sequence of length *L* is of the form:

$$x(0), x(1), \ldots, x(L-1)$$

Input and output data are in float format.

In the generalized cepstral analysis, the speech spectrum is estimated by the *M*-th order generalized cepstrum $c_\gamma(m)$ or by normalized generalized cepstrum $c'_\gamma(m)$ using the log spectrum through the unbiased estimation method showed below.

$$\begin{aligned} H(z) &= s_\gamma^{-1}\left(\sum_{m=0}^{M} c_\gamma(m) z^{-m}\right) \\ &= K \cdot s_\gamma^{-1}\left(\sum_{m=1}^{M} c'_\gamma(m) z^{-m}\right) \\ &= \begin{cases} K \cdot \left(1 + \gamma \sum_{m=1}^{M} c'_\gamma(m) z^{-m}\right)^{1/\gamma}, & -1 \le \gamma < 0 \\ K \cdot \exp \sum_{m=1}^{M} c'_\gamma(m) z^{-m}, & \gamma = 0 \end{cases} \end{aligned}$$

In order to find the minimum value of the cost function, the linear prediction method is used for $\gamma = -1$. Otherwise, the Newton–Raphson method is applied.

OPTIONS

–m	*M*	order of generalized cepstrum	[25]
–g	*G*	gamma of generalized cepstrum $\gamma = G$	[0]
–c	*C*	gamma of generalized cepstrum $\gamma = -1/(\text{int})C$ *C* must be $C \ge 1$	
–l	*L*	frame length	[256]
–n		output normalized cepstrum	[FALSE]

–q *Q* input data style [0]

$Q = 0$ windowed data sequence
$Q = 1$ $20 \times \log |f(w)|$
$Q = 2$ $\ln |f(w)|$
$Q = 3$ $|f(w)|$
$Q = 4$ $|f(w)|^2$

Usually, the options below do not need to be assigned.

–i *I* minimum iteration [2]
–j *J* maximum iteration [30]
–d *D* Newton-Raphson method end condition. The default value is $D = 0.001$. In this case, the end point is achieved when the evaluation rate of $\varepsilon^{(i)}$ is 0.001, that is, when its value changes in a rate smaller than 0.1%. [0.001]
–e *e* small value added to periodgram [0]
–E *E* floor in db calculated per frame [N/A]
–f *F* mimimum value of the determinant of the normal matrix [0.000001]

EXAMPLE

In the following example, speech data is read in float format from *data.f* file, and a 15-th order generalized cepstral analysis is applied. The results are written to *data.gcep*:

```
frame < data.f | window | gcep -m 15 > data.gcep
```

SEE ALSO

uels, mcep, mgcep, glsadf

NAME

glogsp – draw a log spectrum graph

SYNOPSIS

glogsp **[–F** *F* **] [–O** *O* **] [–x** *X* **] [–y** *ymin ymax* **] [–ys** *YS* **] [–p** *P* **] [–ln** *LN* **]**
[–s *S* **] [–l** *L* **] [–c** *comment* **] [** ***infile*** **]**

DESCRIPTION

glogsp converts float-format log spectral data from *infile* (or standard input) to FP5301 plot format, sending the result to standard output. The output can be visualized with xgr.

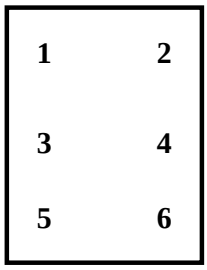
glogsp is implemented as a shell script that uses the fig and fdrw commands.

OPTIONS

–F *F* factor [1]

–O *O* origin of graph [1]

1 (40,205) [mm]
2 (125,205) [mm]
3 (40,120) [mm]
4 (125,120) [mm]
5 (40, 35) [mm]
6 (125, 35) [mm]



–x *X* *x* scale [1]

1 normalized frequency (0 ~ 0.5)
2 normalized frequency (0 ~ π)
4 frequency (0 ~ 4 kHz)
5 frequency (0 ~ 5 kHz)
8 frequency (0 ~ 8 kHz)
10 frequency (0 ~ 10 kHz)

–y *ymin ymax* *y* scale[dB] [0 100]

–ys *YS* Y-axis scaling factor [20]

–p *P* pen number(1 ~ 10) [1]

–ln *LN* kind of line style(0 ~ 5) (see also fig) [1]

–s *S* start frame number [0]

–l *L* frame length [256]

–c	comment	comment for the graph	[N/A]

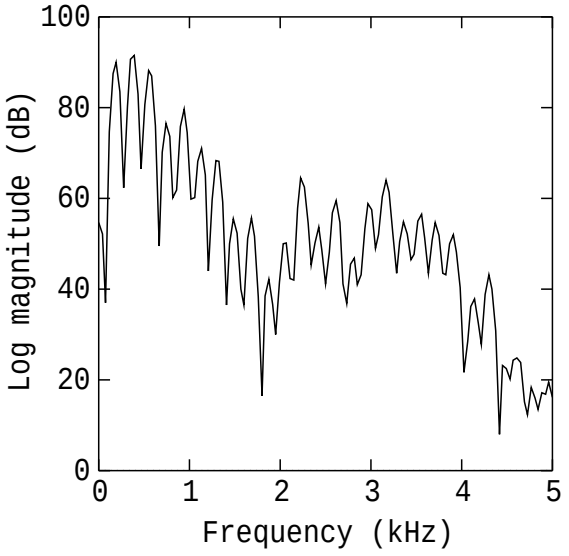
Usually, the options below do not need to be assigned.

–W	*W*	width of the graph (mm)	[0.6]
–H	*H*	height of the graph (mm)	[0.6]
–v		over write mode	[FALSE]
–o	*xo yo*	origin of the graph. if -o option exists, -O is not effective	[40 205]
–g	*G*	type of frame of the graph (0 ~ 2) (see also fig)	[2]
–f	*file*	additional data file for fig	[NULL]
–help		print help in detail	

EXAMPLE

In the example below, speech data sampled at 10 kHz is read in short format from *data.s* file, the magnitude of its log spectrum is evaluated and plotted on the screen:

```
x2x +sf data.s | bcut +f -s 4000 -e 4255 | window -n 2| spec |\
glogsp -x 5 | xgr
```



SEE ALSO

fig, fdrw, xgr, psgr, grlogsp, gwave

NAME

glsadf – GLSA digital filter for speech synthesis(18)

SYNOPSIS

glsadf **[–m** *M* **] [–c** *C* **] [–p** *P* **] [–i** *I* **] [–v] [–t] [–n] [–k] [–P** *Pa* **]** ***gcfile*** **[** ***infile*** **]**

DESCRIPTION

glsadf derives a Generalized Log Spectral Approximation digital filter from normalized generalized cepstral coefficients in *gcfile* and uses it to filter an excitation sequence from *infile* (or standard input) to synthesize speech data, sending the result to standard output. The cepstral coefficients can be be represented as $K, c'_\gamma(1), \ldots, c'_\gamma(M)$.

Input and output data are in float format.

The transfer function $H(z)$ are synthesis filter based on an M order normalized generalized cepstral coefficients $c'_\gamma(m)$ is

$$
\begin{aligned}
H(z) &= K \cdot D(z) \\
&= \begin{cases} K \cdot \left(1 + \gamma \sum_{m=1}^{M} c'_\gamma(m) z^{-m}\right)^{1/\gamma}, & 0 < \gamma \leq -1 \\ K \cdot \exp \sum_{m=1}^{M} c'_\gamma(m) z^{-m}, & \gamma = 0 \end{cases}
\end{aligned}
$$

In this case, we are considering only values for the power parameter $\gamma = -1/C$, where C is a natural number. The filter $D(z)$ can be realized through a C level cascade as shown in figure1, where

$$
\frac{1}{C(z)} = \frac{1}{1 + \gamma \sum_{m=1}^{M} c'_\gamma(m) z^{-m}}
$$

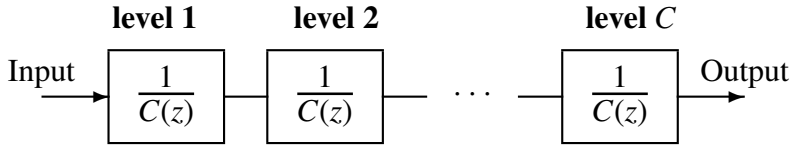


Figure 1: Structure of filter $D(z)$

OPTIONS

–m	*M*	order of generalized cepstrum	[25]
–c	*C*	power parameter $\gamma = -1/C$ for generalized cepstrum if $C == 0$ then the LMA filter is used	[1]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–n		regard input as normalized generalized cepstrum	[FALSE]
–v		inverse filter	[FALSE]
–t		transpose filter	[FALSE]
–k		filtering without gain	[FALSE]

The option below only works if $C == 0$.

–P	*Pa*	order of the Padé approximation *Pa* should be 4 or 5	[4]

EXAMPLE

In this example, excitation is generated through the pitch data in the file *data.pitch* in float format, passed through a GLSA filter based on the generalized cepstral coefficients file *data.gcep*, and the synthesized speech is output to *data.syn*:

```
excite < data.pitch | glsadf data.gcep > data.syn
```

SEE ALSO

ltcdf, lmadf, lspdf, mlsadf, mglsadf

NAME

gmm – GMM parameter estimation

SYNOPSIS

gmm **[–l** *L* **] [–m** *M* **] [–t** *T* **] [–s** *S* **] [–a** *A* **] [–b** *B* **] [–e** *E* **] [–v** *V* **] [–w** *W* **] [–f]**
[–F *gmmfile* **] [** ***infile*** **]**

DESCRIPTION

gmm uses the expectation maximization (EM) algorithm to estimate Gaussian mixture model (GMM) parameters with diagonal covariance matrices, from a sequence of vectors in the *infile* (or standard input), sending the result to standard output.

The input sequence $\boldsymbol{X}$ consists of T float vectors $\boldsymbol{x}$, each of size L:

$$\begin{aligned}\boldsymbol{X} &= [\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots, \boldsymbol{x}(T-1)]\,,\\ \boldsymbol{x}(t) &= [x_t(0), x_t(1), \ldots, x_t(L-1)]\,.\end{aligned}$$

The result is GMM parameters λ consisting of M mixture weights $\boldsymbol{w}$ and M Gaussians with mean vector $\boldsymbol{\mu}$ and variance vector $\boldsymbol{v}$, each of length L:

$$\lambda = [\boldsymbol{w},\ \boldsymbol{\mu}(0), \boldsymbol{v}(0), \boldsymbol{\mu}(1), \boldsymbol{v}(1), \ldots, \boldsymbol{\mu}(M-1), \boldsymbol{v}(M-1)]\,,$$

$$\begin{aligned}\boldsymbol{w} &= [w(0), w(1), \ldots, w(M-1)]\,,\\ \boldsymbol{\mu}(m) &= [\mu_m(0), \mu_m(1), \ldots, \mu_m(L-1)]\,,\\ \boldsymbol{v}(m) &= \left[\sigma_m^2(0), \sigma_m^2(1), \ldots, \sigma_m^2(L-1)\right],\end{aligned}$$

where

$$\sum_{m=0}^{M-1} w(m) = 1.$$

The GMM parameter set λ is initialized by an LBG algorithm and the following EM steps are used iteratively to obtain the new parameter set $\hat{\lambda}$:

$$\hat{w}(m) = \frac{1}{T}\sum_{t=0}^{T-1} p(m \mid \boldsymbol{x}(t), \lambda),$$

$$\hat{\boldsymbol{\mu}}(m) = \frac{\sum_{t=0}^{T-1} p(m \mid \boldsymbol{x}(t), \lambda)\boldsymbol{x}(t)}{\sum_{t=0}^{T-1} p(m \mid \boldsymbol{x}(t), \lambda)},$$

$$\hat{\sigma}_m^2(l) = \frac{\sum_{t=0}^{T-1} p(m \mid \boldsymbol{x}(t), \lambda)x_t^2(l)}{\sum_{t=0}^{T-1} p(m \mid \boldsymbol{x}(t), \lambda)} - \hat{\mu}_m^2(l),$$

where $p(m \mid \boldsymbol{x}(t), \lambda)$ is the posterior probability of being in the m-th component at time t and is given by:

$$p(m \mid \boldsymbol{x}(t), \lambda) = \frac{w(m)\mathcal{N}(\boldsymbol{x}(t) \mid \boldsymbol{\mu}(m), \boldsymbol{v}(m))}{\sum_{k=0}^{M-1} w(k)\mathcal{N}(\boldsymbol{x}(t) \mid \boldsymbol{\mu}(k), \boldsymbol{v}(k))},$$

where

$$\mathcal{N}(\boldsymbol{x}(t) \mid \boldsymbol{\mu}(m), \boldsymbol{v}(m)) = \frac{1}{(2\pi)^{L/2} |\Sigma(m)|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}(t) - \boldsymbol{\mu}(m))' \Sigma(m)^{-1} (\boldsymbol{x}(t) - \boldsymbol{\mu}(m))\right\}$$
$$= \frac{1}{(2\pi)^{L/2} \prod_{l=0}^{L-1} \sigma_m(l)} \exp\left\{-\frac{1}{2} \sum_{l=0}^{L-1} \frac{(x_t(l) - \mu_m(l))^2}{\sigma_m^2(l)}\right\},$$

and $\Sigma(m)$ is a diagonal matrix with diagonal elements $\boldsymbol{v}(m)$:

$$\Sigma(m) = \begin{bmatrix} \sigma_m^2(0) & & & 0 \\ & \sigma_m^2(1) & & \\ & & \ddots & \\ 0 & & & \sigma_m^2(L-1) \end{bmatrix}.$$

Also, the Average log-likelihood for training data X

$$\log P(\boldsymbol{X}) = \frac{1}{T} \sum_{t=0}^{T-1} \log \sum_{m=0}^{M-1} w(m) \mathcal{N}(\boldsymbol{x}(t) \mid \boldsymbol{\mu}(m), \boldsymbol{v}(m))$$

is increased by iterating the above steps. The average log-probability $\log P(\boldsymbol{X})$ at each iterative step is printed on the standard error output. The EM steps are iterated at least A times and stopped at the B-th iteration or when there is a small absolute change in $\log P(\boldsymbol{X})$ ($\leq E$).

OPTIONS

–l	*L*	length of vector	[26]
–m	*M*	number of Gaussian components	[16]
–t	*T*	number of training vectors	[N/A]
–s	*S*	seed of random variable for LBG algorithm	[1]
–a	*A*	minimum number of EM iterations	[0]
–b	*B*	maximum number of EM iterations (A≤ B)	[20]
–e	*E*	end condition for EM iteration	[0.00001]
–v	*V*	flooring value for variances	[0.001]
–w	*W*	flooring value for weights (1/M)*W	[0.001]
–f		full covariance	[FALSE]
–F	*fn*	GMM initial parameter file	[N/A]

EXAMPLE

In the following example, a GMM with 8 Gaussian components is generated from training vectors *data.f* in float format, and GMM parameters are written to *gmm.f*.

```
gmm -m 8 data.f > gmm.f
```

If one wants to model GMMs with full covariances, one can use the -f option.

```
gmm -m 8 -f data.f > gmm.f
```

The -F option can be used to specify GMM initial parameter file *gmm.init*.

```
gmm -m 8 -f data.f -F gmm.init > gmm.f
```

SEE ALSO

gmmp, lbg

NAME

gmmp – calculation of GMM log-probability

SYNOPSIS

gmmp [–l *L* **] [–m** *M* **] [–a]** ***gmmfile*** **[** ***infile*** **]**

DESCRIPTION

gmmp calculates GMM log-probabilities of input vectors from *infile* (or standard input). The *gmmfile* has the same file format as the one generated by the *gmm* command, i.e., *gmmfile* consists of M mixture weights $\boldsymbol{w}$ and M Gaussians with mean vector $\boldsymbol{\mu}$ and diagonal variance vector $\boldsymbol{v}$, each of length L:

$$\lambda = [\boldsymbol{w},\ \boldsymbol{\mu}(0), \boldsymbol{v}(0), \boldsymbol{\mu}(1), \boldsymbol{v}(1), \ldots, \boldsymbol{\mu}(M-1), \boldsymbol{v}(M-1)],$$

$$\begin{aligned} \boldsymbol{w} &= [w(0), w(1), \ldots, w(M-1)], \\ \boldsymbol{\mu}(m) &= [\mu_m(0), \mu_m(1), \ldots, \mu_m(L-1)], \\ \boldsymbol{v}(m) &= \left[\sigma_m^2(0), \sigma_m^2(1), \ldots, \sigma_m^2(L-1)\right]. \end{aligned}$$

The input sequence consists of T float vectors $\boldsymbol{x}$, each of size L:

$$\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots, \boldsymbol{x}(T-1).$$

The result is a sequence of log-probabilities of input vectors:

$$\log b(\boldsymbol{x}(0)), \log b(\boldsymbol{x}(1)), \ldots, \log b(\boldsymbol{x}(T-1)),$$

or an average log-probability (if -a option is used):

$$\log P(\boldsymbol{X}) = \frac{1}{T} \sum_{t=0}^{T-1} \log b(\boldsymbol{x}(t)),$$

where

$$b(\boldsymbol{x}(t)) = \sum_{m=0}^{M-1} w(m) \mathcal{N}(\boldsymbol{x}(t)\ ;\ \boldsymbol{\mu}(m), \boldsymbol{v}(m)),$$

$$\mathcal{N}(\boldsymbol{x}(t)\ ;\ \boldsymbol{\mu}(m), \boldsymbol{v}(m)) = \frac{1}{(2\pi)^{L/2} \prod_{l=0}^{L-1} \sigma_m(l)} \exp\left\{-\frac{1}{2} \sum_{l=0}^{L-1} \frac{(x_t(l) - \mu_m(l))^2}{\sigma_m^2(l)}\right\}.$$

OPTIONS

–l	*L*	length of vector	[26]
–m	*M*	number of Gaussian components	[16]
–a		print average log-probability	[FALSE]

EXAMPLE

In the following example, frame log-probabilities of input data *data.f* for GMM with 8 Gaussians *gmm.f* are written to *probs.f*.

```
gmmp -m 8 gmm.f data.f > probs.f
```

SEE ALSO

gmm

NAME

gnorm – gain normalization

SYNOPSIS

gnorm **[–m** *M* **] [–g** *G* **] [–c** *C* **] [** ***infile*** **]**

DESCRIPTION

gnorm normalizes generalized cepstral coefficients $c_\gamma(m)$ from *infile* (or standard input), sending the normalized generalized cepstral coefficients to standard output.

Both input and output files are in float format.

The normalized generalized cepstral coefficients $c'_\gamma(m)$ can be written as

$$c'_\gamma(m) = \frac{c_\gamma(m)}{1 + \gamma c_\gamma(0)}, \qquad m > 0$$

Also, the gain $K = c'_\gamma(0)$ is given by:

$$K = \begin{cases} \left(\dfrac{1}{1 + \gamma c_\gamma(0)}\right)^{1/\gamma}, & 0 < |\gamma| \leq 1 \\ \exp c_\gamma(0), & \gamma = 0 \end{cases}$$

OPTIONS

–m	*M*	order of generalized cepstrum	[25]
–g	*G*	power parameter γ of generalized cepstrum, $\gamma = G$	[0]
–c	*C*	power parameter γ of generalized cepstrum, $\gamma = -1/(\text{int})C$ C must be $C \geq 1$	

EXAMPLE

In this example, generalized cepstral coefficients in float format are read from file *data.gcep* ($M = 15, \gamma = -0.5$), normalized and output to *data.ngcep*:

```
gnorm -m 15 -c 2 < data.gcep > data.ngcep
```

SEE ALSO

ignorm, gcep, mgcep, gc2gc, mgc2mgc, freqt

NAME

grlogsp – draw a running log spectrum graph

SYNOPSIS

grlogsp **[–t] [–F** *F*] **[–O** *O* **] [–x** *X* **] [–y** *ymin* **] [–yy** *YY* **] [–yo** *YO* **] [–p** *P* **]**

[–ln *LN* **] [–s** *S* **] [–e** *E* **] [–n** *N* **] [–l** *L* **]**

[–c *comment*1 **] [–c2** *comment*2 **] [–c3** *comment*3 **] [** ***infile*** **]**

DESCRIPTION

grlogsp converts a sequence of float-format log spectra from *infile* (or standard input) to a running spectrum plot in FP5301 plot format, sending the result to standard output. The output can be visualized with xgr.

grlogsp is implemented as a shell script that uses the fig and fdrw commands.

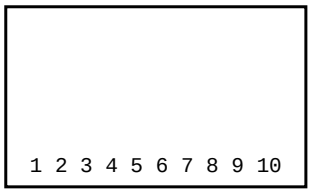
OPTIONS

–t		transpose x and y axes	[FALSE]
–F	*F*	factor	[1]
–O	*O*	origin of graph	[1]

if *O* is more than 6, drawing area is over A4

range

1	(25,*YO*)	[mm]
2	(60,*YO*)	[mm]
3	(95,*YO*)	[mm]
4	(130,*YO*)	[mm]
5	(165,*YO*)	[mm]
6	(200,*YO*)	[mm]
7	(235,*YO*)	[mm]
8	(270,*YO*)	[mm]
9	(305,*YO*)	[mm]
10	(340,*YO*)	[mm]



(*YO* + 100, *X*) [mm] if -t is specified.

–x	*X*	*x* scale 1 normalized frequency (0 ~ 0.5) 2 normalized frequency (0 ~ π) 4 frequency (0 ~ 4 kHz) 5 frequency (0 ~ 5 kHz) 8 frequency (0 ~ 8 kHz) 10 frequency (0 ~ 10 kHz)	[1]
–y	*ymin*	*y* minimum	[-100]
–yy	*YY*	*y* scale [dB/10mm]	[100]
–yo	*YO*	*y* offset	[30]
–p	*p*	type of pen (1 ~ 10)	[2]
–ln	*LN*	style of line (0 ~ 5) (see also fig)	[1]
–s	*S*	start frame number	[0]
–e	*E*	end frame number	[EOF]
–n	*N*	number of frame	[EOF]
–l	*L*	frame length. Actually $\frac{L}{2}$ data are plotted.	[256]
–c, c2, c3	*comment*1 ~ 3	comment for the graph	[N/A]

Usually, the options below do not need to be assigned.

–W	*W*	width of the graph (×100 mm)	[0.25]
–H	*H*	height of the graph (×100 mm)	[1.5]
–z	*Z*	This option is used when data is written recursively in the *y* axis. the distance between two graphs in the *y* axis are given by *Z*. If Z is not given, Z is as same as F	
–o	*xo yo*	origin of the graph. if -o option exists, -O is not effective.	[95 30]
–g	*G*	type of frame of the graph (0 ~ 2) (see also fig)	[2]
–cy	*cy*	first comment position	[-8]
–cy2	*cy*2	second comment position	[-14]
–cy3	*cy*3	third comment position	[-20]
–cs	*cs*	font size of the comments	[1]
–f	*f*	additional data file for fig	[NULL]

EXAMPLE

In this example, the magnitude of log spectrum is evaluated from data in *data.f* file in float format, and the graph with the running spectrum is sent in Postscript format to *data.ps* file:

```
frame < data.f | window |\
uels -m 15 | c2sp -m 15 |\
grlogsp | psgr > data.ps
```

SEE ALSO

fig, fdrw, xgr, psgr, glogsp, gwave

NAME

grpdelay – group delay of digital filter

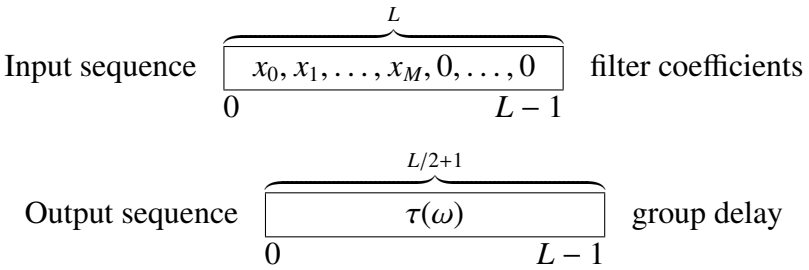
SYNOPSIS

grpdelay **[–l** *L* **] [–m** *M* **] [–a] [** ***infile*** **]**

DESCRIPTION

grpdelay computes the group delay of a sequence of filter coefficients from *infile* (or standard input), sending the result to standard output. Input and output data are in float format.

If the **–m** option is omitted and the length of an input data sequence is less than FFT size, the input file is padded with 0's and the FFT is evaluated as exemplified below. When the **–a** option is given, the gain is obtained from zero order input.



OPTIONS

–l	*L*	FFT size power of 2	[256]
–m	*M*	order of filter	[L-1]
–a		ARMA filter	[FALSE]

EXAMPLE

This example plots in the screen the group delay of impulse response of the filter with the following transfer function.

$$H(z) = \frac{1}{1 + 0.9z^{-1}}$$

```
impulse | dfs -a 1 0.9 | grpdelay | fdrw | xgr
```

SEE ALSO

delay, phase

NAME

gseries – draw a discrete series

SYNOPSIS

gseries **[–F** *F***] [–s** *S* **] [–e** *E* **] [–n** *N* **] [–i** *I* **] [–y** *ymax* **] [–y2** *ymin* **] [–m** *M* **]**
[–p *P* **] [–magic** *magic* **] [–MAGIC** *MAGIC* **] [+*type*] [*infile*]**

DESCRIPTION

gseries converts discrete series data from *infile* (or standard input) to FP5301 plot format, sending the result to standard output. The output can viewed with xgr.

gseries is implemented as a shell script that uses the fig command.

OPTIONS

–F	*F*	factor	[1]
–s	*S*	start point	[0]
–e	*E*	end point	[EOF]
–n	*N*	data number of one screen if this option is omitted, all of the data is plotted on one screen.	[N/A]
–i	*I*	number of screen	[5]
–y	*ymax*	maximum amplitude if this option is omitted, ymax is maximum value of the input data.	[N/A]
–y2	*ymin*	minimum amplitude	[-YMAX]
–m	*M*	mark type	[1]
–p	*P*	pen type(1 ~ 10)	[1]
–magic	*magic*	remove magic number	[FALSE]
–MAGIC	*MAGIC*	replace magic number by *MAGIC* if -magic option is not given, return error. if -magic or -MAGIC option is given multiple times, also return error.	[FALSE]
+*t*		Input data format	[f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)
de	long double (12 bytes)		

EXAMPLE

In the following example, *gseries* reads impulse response in float format from *data.f* and writes the output in encapsulated Postscript format to *data.eps*.

```
gseries +f < data.f | psgr > data.eps
```

SEE ALSO

fig, fdrw, xgr, psgr, glogsp, grlogsp, gwave

NAME

gwave – draw a waveform

SYNOPSIS

gwave **[–F** *F* **] [–s** *S* **] [–e** *E* **] [–n** *N* **] [–i** *I* **] [–y** *ymax* **] [–y2** *ymin* **] [–p** *P* **]**
[*+type*] [*infile*]

DESCRIPTION

gwave converts speech waveform data from *infile* (or standard input) to FP5301 plot format, sending the result to standard output. The output can viewed with xgr.

gwave is implemented as a shell script that uses the fig and fdrw commands.

OPTIONS

–F	*F*	factor	[1]
–s	*S*	start point	[0]
–e	*E*	end point	[EOF]
–n	*N*	data number of one screen if this option is omitted, all of the data is plotted on one screen.	[N/A]
–i	*I*	number of screen	[5]
–y	*ymax*	maximum amplitude if this option is omitted, ymax is maximum value of the input data.	[N/A]
–y2	*ymin*	minimum amplitude	[-YMAX]
–p	*P*	pen type(1 ~ 10)	[1]
+t		Input data format	[f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)
de	long double (12 bytes)		

EXAMPLE

This example reads speech waveform file in float format from *data.f* and writes the output in Postscript format to *data.ps*.

```
gwave +f < data.f | psgr > data.ps
```

SEE ALSO

fig, fdrw, xgr, psgr, glogsp, grlogsp

NAME

histogram – histogram

SYNOPSIS

histogram **[–l** *L* **] [–i** *I* **] [–j** *J* **] [–s** *S* **] [–n] [** ***infile*** **]**

DESCRIPTION

histogram makes histograms of frames of input data from *infile* (or standard input), sending the results to standard output.

Input and output data are in float format. The output can be graphed with fdrw.

If an input value is outside the specified interval, the exit status of histogram will be nonzero, but the output histogram will still be created.

OPTIONS

–l	*L*	frame size	[0]
		$L > 0$ evaluate the histogram for every frame	
		$L = 0$ evaluate the histogram for the whole file	
–i	*I*	infimum	[0.0]
–j	*J*	supremum	[1.0]
–s	*S*	step size	[0.1]
–n		normalization	[FALSE]

EXAMPLE

The example below plots the histogram of the speech waveform file *data.f* in float format.

```
histogram -i -16000 -j 16000 -s 100 data.f | fdrw | xgr
```

SEE ALSO

average

NAME

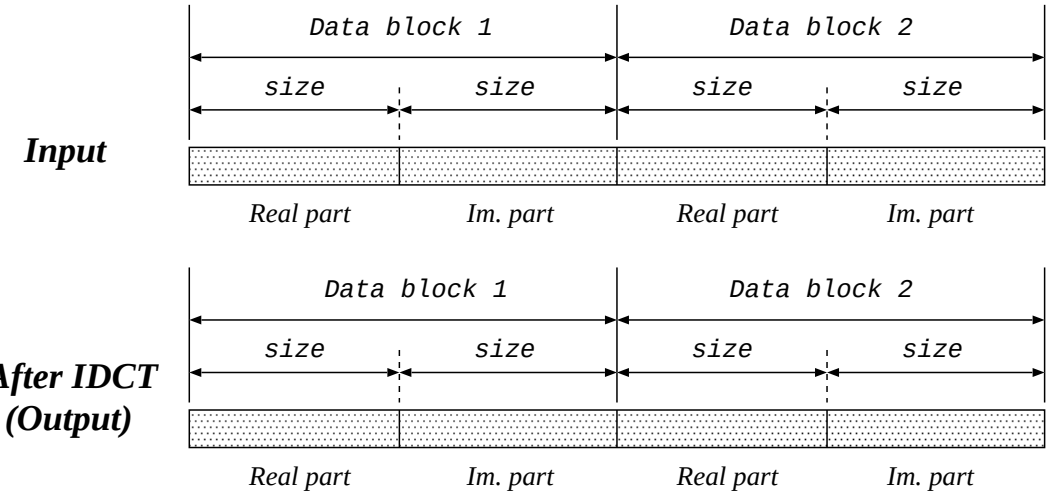
idct – Inverse DCT-II

SYNOPSIS

idct [–l *L* **] [–c] [–d] [** ***infile*** **]**

DESCRIPTION

idct calculates the Inverse Discrete Cosine Transform II (IDCT-II) of input data in *infile* (or standard input), sending the results to standard output. The input and output data is in float format, arranged as follows.



The Inverse Discrete Cosine Transformation II is given by

$$x_l = \sqrt{\frac{2}{L}} c_l \sum_{k=0}^{L-1} X_k \cos\left\{\frac{\pi}{L}\left(k+\frac{1}{2}\right)l\right\}, \quad l = 0, 1, \cdots, L$$

where

$$c_l = \begin{cases} 1 & (1 \le l \le L-1) \\ 1/\sqrt{2} & (l = 0) \end{cases}$$

OPTIONS

–l	*L*	IDCT size	[256]
–c		use complex number	[FALSE]
–d		don't use FFT algorithm	[FALSE]

EXAMPLE

In this example, the IDCT is evaluated from a complex-valued data file *data.f* in float format (real part: 256 points, imaginary part: 256 points), and the output is written to *data.idct*:

```
idct data.f -l 256 -c > data.idct
```

SEE ALSO

fft, dct

NAME

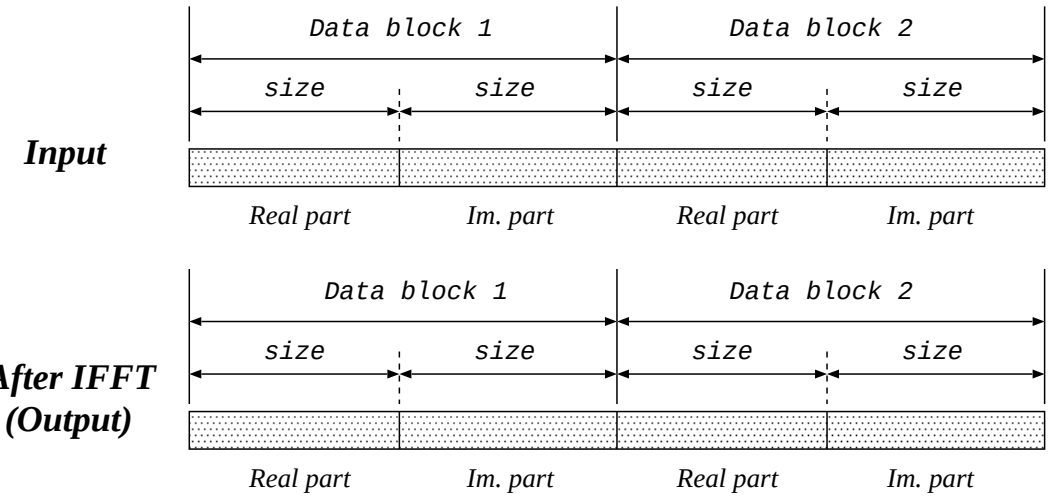
ifft – inverse FFT for complex sequence

SYNOPSIS

ifft [–l *L* **] [–{ R | I }] [** ***infile*** **]**

DESCRIPTION

ifft calculates the Inverse Discrete Fourier Transform (IDFT) of complex-valued data from *infile* (or standard input), sending the results to standard output. The input and output data is in float format, arranged as follows.



OPTIONS

–l	*L*	FFT size power of 2	[256]
–R		output only real part	[FALSE]
–I		output only imaginary part	[FALSE]

EXAMPLE

In this example, the inverse DFT is evaluated from a data file *data.f* in float format (real part: 256 points, imaginary part: 256 points), and the output is written to *data.ifft*:

```
ifft data.f -l 256 > data.ifft
```

SEE ALSO

fft, fft2, fftr, fftr2, ifftr ifft2

NAME

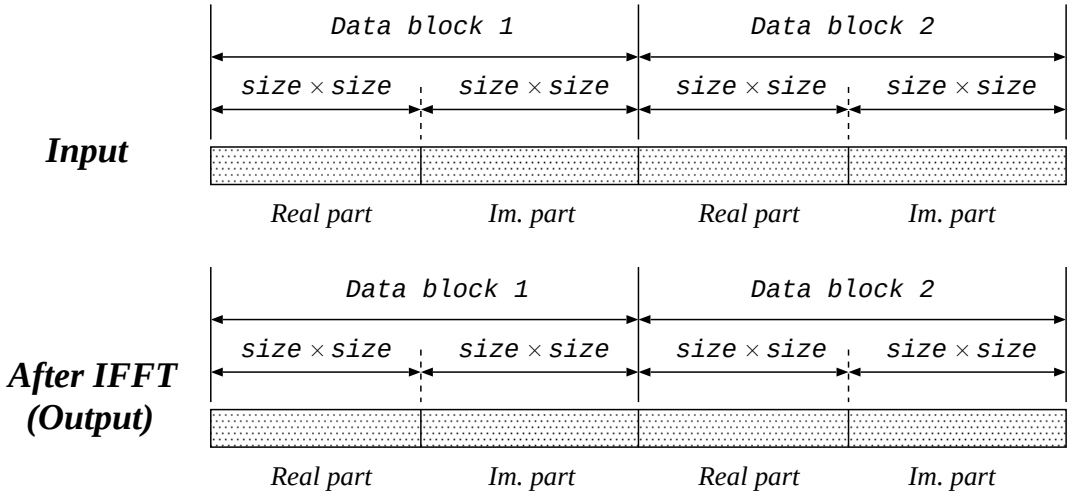
ifft2 – 2-dimensional inverse FFT for complex sequence

SYNOPSIS

ifft2 [–l *L* **] [+r] [–t] [–c] [–q] [–{ R | I }] [** ***infile*** **]**

DESCRIPTION

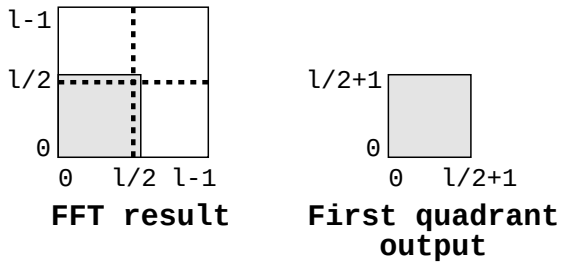
ifft2 calculates the 2-dimensional Inverse Discrete Fourier Transform (IDFT) of complex-valued data from *infile* (or standard input), sending the results to standard output. The input and output data is in float format, arranged as follows.



OPTIONS

–l	*L*	FFT size power of 2	[64]
+r		regard input as real values rather than complex values	[FALSE]
–t		Output results in transposed form (see also fft2).	[FALSE]
–c		When results are transposed, 1 boundary data is copied from the opposite side, and then output $(L + 1) \times (L + 1)$ data (see also fft2).	[FALSE]

–q Output first 1/4 of data of FFT results only. As in the above c option, boundary data is compensated and $(\frac{L}{2}+1)\times(\frac{L}{2}+1)$ data are output. [FALSE]



–R output only real part [FALSE]
–I output only imaginary part [FALSE]

EXAMPLE

This example reads a sequence of 2-dimensional complex numbers in float format from *data.f* file, evaluates its 2-dimensional IDFT and outputs it to *data.dft* file:

```
ifft2 < data.f > data.ifft2
```

SEE ALSO

fft, fft2, fftr, fftr2, ifft ifftr

NAME

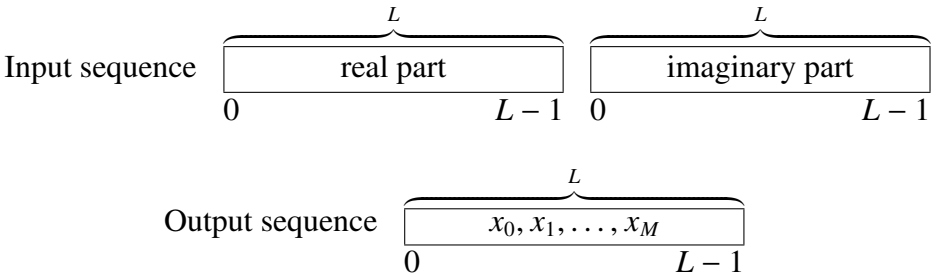
ifftr – inverse FFT for real sequence

SYNOPSIS

ifftr [–l *L* **] [–m** *M* **] [** ***infile*** **]**

DESCRIPTION

ifftr calculates the Inverse Discrete Fourier Transform (IDFT) of real-valued data from *infile* (or standard input), sending the results to standard output. The input and output data is in float format, arranged as follows.



OPTIONS

–l	*L*	FFT size power of 2	[256]
–m	*M*	order of sequence	[L-1]

EXAMPLE

In this example, IDFT is evaluated from a data file *data.f* in float format (real part: 256 points, imaginary part: 256 points), and the output is written to *data.ifftr*:

```
ifftr data.f -l 256 > data.ifftr
```

SEE ALSO

fft, fft2, fftr, fftr2, ifft ifft2

NAME

ignorm – inverse gain normalization

SYNOPSIS

ignorm **[–m** *M* **] [–g** *G* **] [–c** *C* **] [** ***infile*** **]**

DESCRIPTION

ignorm reads normalized generalized cepstral coefficients $c_\gamma(m)$ from *infile* (or standard input), and outputs the unnormalized coefficients to standard output.

Both input and output files are in float format.

To convert normalized generalized cepstral coefficients $c'_\gamma(m)$ into not-normalized generalized cepstral coefficients $c_\gamma(m)$, the following equation can be used.

$$c_\gamma(m) = \left(c'_\gamma(0)\right)^\gamma c'_\gamma(m), \qquad m > 0$$

Also, the gain $K = c_\gamma(0)$ is

$$c_\gamma(0) = \begin{cases} \dfrac{\left(c'_\gamma(0)\right)^\gamma - 1.0}{\gamma}, & 0 < |\gamma| \leq 1 \\ \log c'_\gamma(0), & \gamma = 0 \end{cases}$$

OPTIONS

–m	*M*	order of generalized cepstrum	[25]
–g	*G*	power parameter γ of generalized cepstrum $\gamma = G$	[0]
–c	*C*	power parameter γ of generalized cepstrum $\gamma = -1/(\text{int})C$ C must be $C \geq 1$	

EXAMPLE

In this example below, normalized generalized cepstral coefficients in float format are read from *data.ngcep* ($M = 15, \gamma = -0.5$), and the not-normalized generalized cepstral coefficients are output to *data.gcep*.

```
ignorm -m 15 -c 2 < data.ngcep > data.gcep
```

SEE ALSO

gcep, mgcep, gc2gc, mgc2mgc, freqt

NAME

impulse – generate impulse sequence

SYNOPSIS

impulse **[–l** *L* **] [–n** *N* **]**

DESCRIPTION

impulse generates the unit impulse sequence of length L, sending the output to standard output. The output is in float format as follows.

$$\underbrace{1, 0, 0, \ldots, 0}_{L}$$

If both –l and –n options are given, the last one is used.

OPTIONS

–l	*L*	length of unit impulse if $L < 0$ then endless sequence is generated.	[256]
–n	*N*	order of unit impulse	[255]

EXAMPLE

In the example below, an unit impulse sequence is passed through a digital filter and the results are shown on the screen.

```
impulse | dfs -a 1 0.9 -b 1 2 1 | dmp +f
```

SEE ALSO

step, train, ramp, sin, nrand

NAME

imsvq – decoder of multi stage vector quantization

SYNOPSIS

imsvq [–l *L* **] [–n** *N* **] [–s** *S* ***cbfile*** **] [** ***infile*** **]**

DESCRIPTION

imsvq decodes multi-stage vector-quantized data from a sequence of codebook indexes from *infile* (or standard input), using codebooks specified by multiple –s options, sending the result to standard output. The number of decoder stages is equal to the number of –s options.

Input data is in int format, and output data is in float format.

OPTIONS

–l	*L*	length of vector	[26]
–n	*N*	order of vector	[L-1]
–s	*S cbfile*	codebook	[N/A N/A]

S codebook size
cbfile codebook file

EXAMPLE

In the example below, the decoded vector *data.ivq* is obtained from the first stage codebook *cbfile1* and the second stage codebook *cbfile2*, both of size 256, as well as from the index file *data.vq*.

```
imsvq -s 256 cbfile1 -s 256 cbfile2 < data.vq > data.ivq
```

SEE ALSO

msvq, ivq, vq

NAME

interpolate – interpolation of data sequence

SYNOPSIS

interpolate [–p *P* **] [–s** *S* **] [–d] [** ***infile*** **]**

DESCRIPTION

This function interpolates data points into the input data, with interval P and start number S, and sends the result to standart output. The results are as follows:

$$x(0), x(1), x(2), \ldots$$

then the output data will be

$$\underbrace{0, 0, \ldots, 0}_{S-1}, \underbrace{x(0), 0, 0, \ldots, 0}_{P}, \underbrace{x(1), 0, 0, \ldots, 0}_{P}, x(2), \ldots$$

If the –d option is given, the output data will be

$$\underbrace{0, 0, \ldots, 0}_{S-1}, \underbrace{x(0), x(0), x(0), \ldots, x(0)}_{P}, \underbrace{x(1), x(1), x(1), \ldots, x(1)}_{P}, x(2), \ldots$$

Input and output data are in float format.

OPTIONS

–p	*P*	interpolation period	[10]
–s	*S*	start sample	[0]
–d		pad input data rather than 0	[FALSE]

EXAMPLE

This example decimates input data from *data.f* file with interval 2, interpolates 0 with interval 2, and then outputs it to *data.di* file:

```
decimate -p 2  < data.f | interpolate -p 2 > data.di
```

SEE ALSO

decimate

NAME

ivq – decoder of vector quantization

SYNOPSIS

ivq [–l *L* **] [–n** *N* **]** ***cbfile*** **[** ***infile*** **]**

DESCRIPTION

ivq decodes vector-quantized data from a sequence of codebook indexes from *infile* (or standard input), using the codebook *cbfile*, sending the result to standard output. The decoded output vector is of the form:

$$c_i(0), c_i(1), \ldots, c_i(L-1).$$

Input data is in int format, and output data is in float format.

OPTIONS

–l	*L*	length of vector	[26]
–n	*N*	order of vector	[L-1]

EXAMPLE

In the following example, the decoded 25-th order output file *data.ivq* is obtained through the index file *data.vq* and codebook *cbfile*.

```
ivq cbfile data.vq > data.ivq
```

SEE ALSO

vq, imsvq, msvq

NAME

lbg – LBG algorithm for vector quantizer design

SYNOPSIS

lbg **[–l** *L* **]** **[–n** *N* **]** **[–t** *T* **]** **[–s** *S* **]** **[–e** *E* **]** **[–F** *F* **]** **[–i** *I* **]** **[–m** *M* **]** **[–S** *s* **]**
[–c *C* **]** **[–d** *D* **]** **[–r** *R* **]** **[** ***indexfile*** **]** < ***infile***

DESCRIPTION

lbg uses the LBG algorithm to train a codebook from a sequence of vectors from *infile* (or standard input), sending the result to standard output.

The input sequence consists of T float vectors $\boldsymbol{x}$, each of size L

$$\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots, \boldsymbol{x}(T-1).$$

The result is a codebook consisting of E float vectors, each of length L,

$$\boldsymbol{C}_E = \{\boldsymbol{c}_E(0), \boldsymbol{c}_E(1), \ldots, \boldsymbol{c}_E(E-1)\},$$

generated by the following algorithm.

step.0 When an initial codebook $\boldsymbol{C}_S$ is not assigned, the initial codebook is obtained from the whole collection of training data as follows,

$$\boldsymbol{c}_1(0) = \frac{1}{T}\sum_{n=0}^{T-1} \boldsymbol{x}(n)$$

and the initial codebook with $S = 1$ is $\boldsymbol{C}_1 = \{\boldsymbol{c}_1(0)\}$.

step.1 From codebook $\boldsymbol{C}_S$ obtain $\boldsymbol{C}_{2S}$. For this step, the normalized random vector of size L and the splitting factor R are used as follows,

$$\boldsymbol{c}_{2S}(n) = \begin{cases} \boldsymbol{c}_S(n) + R \cdot \mathbf{rnd} & (0 \le n \le S-1) \\ \boldsymbol{c}_S(n-S) - R \cdot \mathbf{rnd} & (S \le n \le 2S-1) \end{cases}$$

and we make $D_0 = \infty$, $k = 0$.

step.2 First, make sure that $k \le I$ where I is the maximum iterations number specified by –i option. If it is true, proceed to the following steps. If not, then go to **step.4**. The present codebook $\boldsymbol{C}_{2S}$ is now applied to the training vectors. After that, the mean Euclidean distance D_k is evaluated from every training vector and their corresponding code vector. If the following condition

$$\left|\frac{D_{k-1} - D_k}{D_k}\right| < D$$

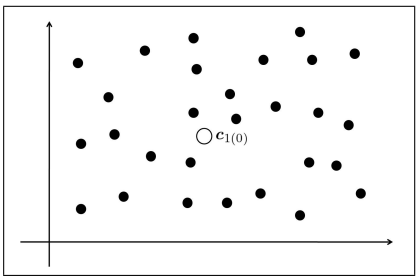


Figure 2: **step.0**: initialize codebook

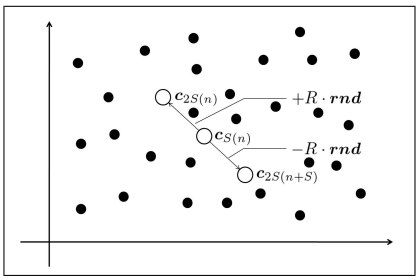


Figure 3: **step.1**: split codebook $\boldsymbol{C}_S$ into $\boldsymbol{C}_{2S}$

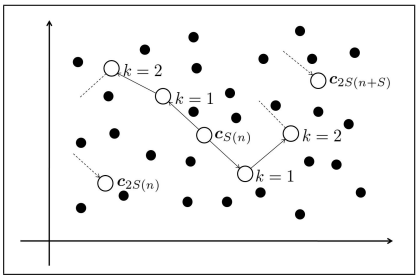


Figure 4: **step.2**: update codebook

is met, then go to **step.4**. If it is not met, then go to **step.3**. The steps 0, 1, and 2 are illustrated in figure 2, 3, and 4, respectivelly.

$$\left| \frac{D_{k-1} - D_k}{D_k} \right| < D$$

step.3 Centroids are evaluated from the results obtained in **step.2**. Then, the codebook $\boldsymbol{C}_{2S}$ is updated. Also, if a cell has less than M training vectors, then the corresponding code vector is erased from the codebook, and a new code vector is generated from either: 1) the code vector $\boldsymbol{c}_{2S}(j)$ corresponding to the cell with more training vectors , as follows.

$$\boldsymbol{c}_{2S}(i) = \boldsymbol{c}_{2S}(j) + R \cdot \mathbf{rnd}$$

Also, $\boldsymbol{c}_{2S}(j)$ is modified as follows.

$$\boldsymbol{c}_{2S}(j) = \boldsymbol{c}_{2S}(j) - R \cdot \mathbf{rnd}$$

2) the vector $\boldsymbol{p}$, which internally divides two centroids proportionally the number of training vectors for the cell. They are split from the same parent centroid. The vector $\boldsymbol{p}$ is given by:

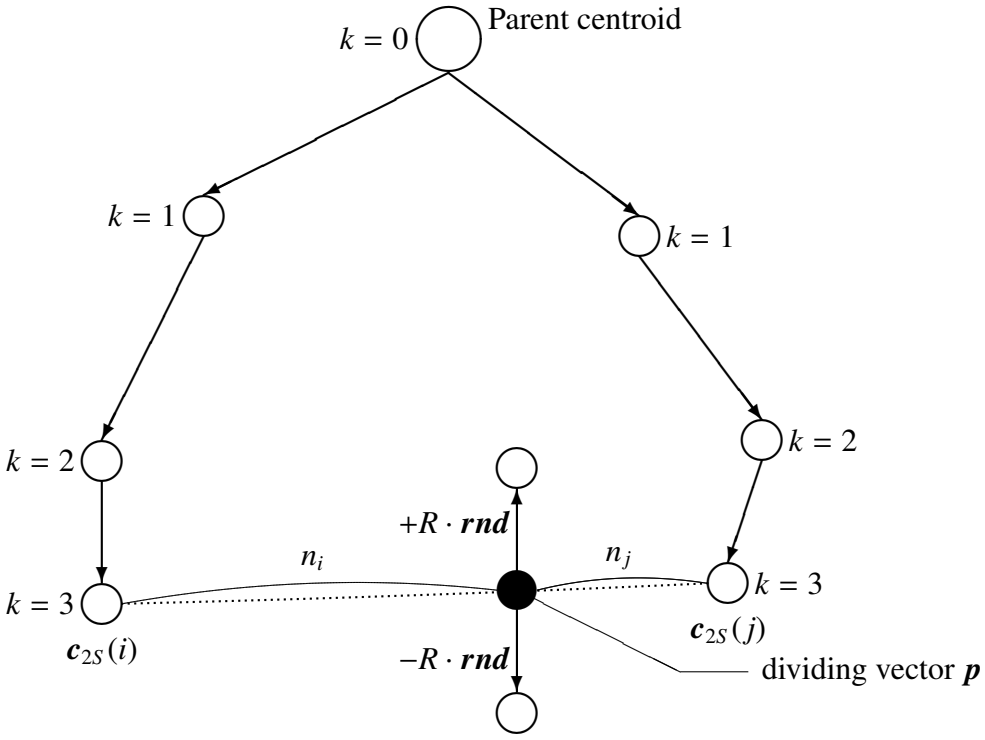
$$\boldsymbol{p} = \frac{n_j \boldsymbol{c}_{2S}(i) + n_i \boldsymbol{c}_{2S}(j)}{n_i + n_j},$$

where n_i and n_j represent the number of training vectors for the cells $\boldsymbol{c}_{2S}(i) and \boldsymbol{c}_{2S}(j)$, respectivelly. The update method is as follows.

$$\boldsymbol{c}_{2S}(i) = \boldsymbol{p} + R \cdot \mathbf{rnd},$$

$$\boldsymbol{c}_{2S}(j) = \boldsymbol{p} - R \cdot \mathbf{rnd}.$$

If the number of traning vectors for the cell is less than M when $k = 3$, the dividing vector $\boldsymbol{p}$ and the update results are given as follows:



The type of split can be specified by the –c option. After that, we assign $k = k + 1$ and then go back to **step.2**

step.4 If $2S = E$ then, end. If not, then make $S = 2S$ and go back to **step.1**.

OPTIONS

–l	*L*	length of vector	[26]
–n	*N*	order of vector	[L−1]
–t	*T*	number of training vector	[N/A]
–s	*S*	initial codebook size	[1]
–e	*E*	final codebook size	[256]
–F	*F*	initial codebook filename	[NULL]
–i	*I*	maximum number of iteration for centroid update	[1000]
–m	*M*	minimum number of training vectors for each cell	[1]
–S	*s*	seed for normalized random vector	[1]

–c *C* type of exception procedure for centroid update [1]
when the number of training vectors for the cell is less than *M*

C = 1 split the centroid with most training vectors
C = 2 split the vector which internally divide
two centroids sharing the same parent centroid,
in proportion to the number of training vectors for the cell.

Usually, the options below do not need to be assigned.

–d *D* end condition [0.0001]
–r *R* splitting factor [0.0001]

EXAMPLE

In the following example, a codebook of size 1024 is generated from the 39-th order training vector *data.f* in float format. It is also specified that the iterations for the centroid update are at most 100 times, that each centroid contains at least 10 training vectors and that random vectors for the centroid update are generated with seed 5. The output is written to *cbfile*.

```
lbg -n 39 -e 1024 -i 100 -m 10 -S 5 < data.f > cbfile
```

SEE ALSO

vq, ivq, msvq

NAME

levdur – solve an autocorrelation normal equation using Levinson-Durbin method

SYNOPSIS

levdur **[–m** *M* **] [–f** *F* **] [** ***infile*** **]**

DESCRIPTION

levdur calculates linear prediction coefficients (LPC) from the autocorrelation matrix from *infile* (or standard input), sending the result to standard output.

The input is the M-th order autocorrelation matrix

$$r(0), r(1), \ldots, r(M).$$

levdur uses the Levinson-Durbin algorithm to solve a system of linear equations obtained from the autocorrelation matrix.

Input and output data are in float format.

The linear prediction coefficients are the set of coefficients $K, a(1), \ldots, a(M)$ of an all-pole digital filter

$$H(z) = \frac{K}{1 + \sum_{i=1}^{M} a(k) z^{-i}}.$$

The linear prediction coefficients are evaluated by solving the following set of linear equations, which were obtained through the autocorrelation method,

$$\begin{pmatrix} r(0) & r(1) & \ldots & r(M-1) \\ r(1) & r(0) & & \vdots \\ \vdots & & \ddots & \\ r(M-1) & & \ldots & r(0) \end{pmatrix} \begin{pmatrix} a(1) \\ a(2) \\ \vdots \\ a(M) \end{pmatrix} = - \begin{pmatrix} r(1) \\ r(2) \\ \vdots \\ r(M) \end{pmatrix}$$

The Durbin iterative and efficient algorithm is used to solve the system above. It takes advantage of the Toeplitz characteristic of the autocorrelation matrix:

$$E^{(0)} = r(0)$$

$$k(i) = \frac{-r(i) - \sum_{j=1}^{i} a^{(i-1)}(j) r(i-j)}{E^{(i-1)}}$$

$$a^{(i)}(i) = k(i)$$

$$a^{(i)}(j) = a^{(i-1)}(j) + k(i) a^{(i-1)}(i-j), \qquad 1 \le j \le i-1 \tag{1}$$

$$E^{(i)} = (1 - k^2(i)) E^{(i-1)} \tag{2}$$

Also, for $i = 1, 2, \ldots, M$, equations (1) and (2) are applied recursively, and the gain K is calculated as follows.

$$K = \sqrt{E^{(M)}}$$

OPTIONS

–m	*M*	order of correlation	[25]
–f	*F*	mimimum value of the determinant of the normal matrix	[0.000001]

EXAMPLE

In this example, input data is read in float format from *data.f* and linear prediction coefficients are written to *data.lpc*:

```
frame < data.f | window | acorr -m 25 | levdur > data.lpc
```

SEE ALSO

acorr, lpc

NAME

linear_intpl – linear interpolation of data

SYNOPSIS

linear_intpl **[–l** L **]** **[–m** M **]** **[–x** x_{min} x_{max} **]** **[–i** x_{min} **]** **[–j** x_{max} **]** **[** ***infile*** **]**

DESCRIPTION

linear_intpl reads a 2-dimensional input data sequence from *infile* (or standard input) in which the x-axis values are linearly interpolated by equally-spaced $L-1$ points, and outputs the y-axis values.

If the input data is

$$\begin{matrix} x_0, y_0 \\ x_1, y_1 \\ \vdots \\ x_K, y_K \end{matrix}$$

then the output data will be

$$y_0, y_1, \ldots, y_{L-1}$$

Input and output data are in float format.

This command can also interpolate data sequence in wchich the x-axis values are not equally-spaced, such as digital filter characteristics.

OPTIONS

–l	L	output length	[256]
–m	M	number of interpolation points	[L-1]
–x	x_{min} x_{max}	minimum and maximum values of x-axis in input data	[0.0 0.5]
–i	x_{min}	minimum values of x-axis in input data	[0.0]
–j	x_{max}	maximum values of x-axis in input data	[0.5]

EXAMPLE

This example decimates input data from *data.f* file with interval 2, interpolates 0 with interval 2, and then outputs it to *data.di* file:

When input data *data.f* contains the following data,

$$\begin{matrix} 0, 2 \\ 2, 2 \\ 3, 0 \\ 5, 1 \end{matrix}$$

this example linearly interpolates input data and outputs it to *data.intpl*

```
linear_intpl -m 10 -x 0 5 < data.f > data.intpl
```

And the result is given by:

$$2, 2, 2, 2, 2, 1, 0, 0.25, 0.5, 0.75, 1$$

NAME

lmadf – LMA digital filter for speech synthesis(5; 17)

SYNOPSIS

lmadf [–m *M* **] [–p** *P* **] [–i** *I* **] [–P** *Pa* **] [–v] [–t] [–k]** ***cfile*** **[** ***infile*** **]**

DESCRIPTION

lmadf derives a Log Magnitude Approximation filter from the cepstral coefficients $c(0), c(1), \ldots, c(M)$ in *cfile* and uses it to filter an excitation sequence from *infile* (or standard input) in order to synthesize speech data, sending the result to standard output.

Input and output data are in float format.

The LMA filter is an extremely precise approximation of the exponential transfer function obtained from M-th order cepstral coefficients $c(m)$ as follows.

$$H(z) = \exp \sum_{m=0}^{M} c(m) z^{-m}$$

If we remove the gain $K = \exp c(0)$ from the transfer function $H(z)$, then we obtain the following transfer function

$$D(z) = \exp \sum_{m=1}^{M} c(m) z^{-m},$$

which can be realized using the basic FIR filter

$$F(z) = \sum_{m=1}^{M} c(m) z^{-m}$$

as shown in Figure 1(a). Also, as it can be seen in Figure 1(b), the basic filter $F(z)$ can be decomposed as follows

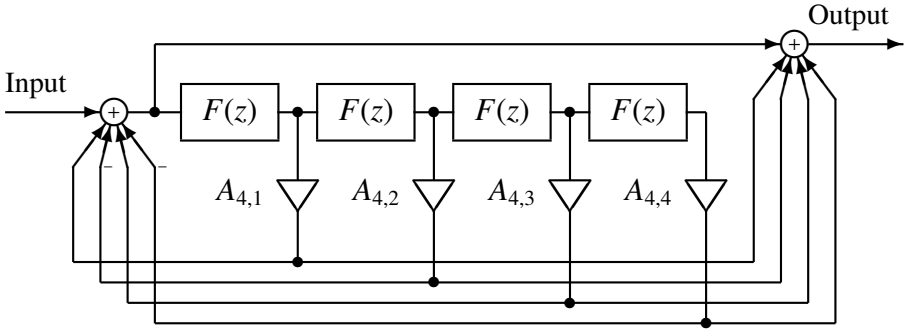
$$F(z) = F_1(z) + F_2(z)$$

where

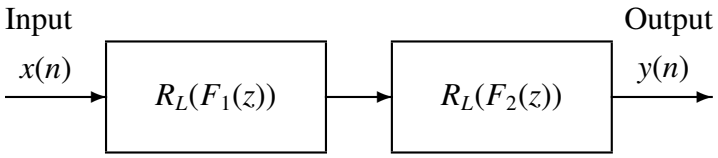
$$F_1(z) = c(1) z^{-1}$$

$$F_2(z) = \sum_{m=2}^{M} c(m) z^{-m}$$

By doing this decomposition, the accuracy of the approximation is improved. Also, the values of the coefficients $A_{4,l}$ are given in table 1



(a)



(b)

Figure 1: (a) $R_L(F(z)) \simeq D(z) \quad L = 4$
(b) 2 level cascade realization
$R_L(F_1(z)) \cdot R_L(F_2(z)) \simeq D(z)$

Table 1: The values for the coefficients $A_{L,l}$

l	$A_{4,l}$	$A_{5,l}$
1	4.999273×10^{-1}	4.999391×10^{-1}
2	1.067005×10^{-1}	1.107098×10^{-1}
3	1.170221×10^{-2}	1.369984×10^{-2}
4	5.656279×10^{-4}	9.564853×10^{-4}
5		3.041721×10^{-5}

OPTIONS

–m	*M*	order of cepstrum	[25]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–P	*Pa*	order of the Padé approximation *Pa* should be 4 or 5	[4]
–k		filtering without gain	[FALSE]
–v		inverse filter	[FALSE]

–v	transpose filter	[FALSE]

EXAMPLE

In this example, the excitation is generated from the pitch data read in float format from *data.pitch*, passed through an LMA filter obtained from cepstrum file *data.cep*, and the synthesized speech is written to *data.syn*.

```
excite < data.pitch | lmadf data.cep > data.syn
```

SEE ALSO

uels, acep, poledf, ltcdf, glsadf, mlsadf, mglsadf

NAME

lpc – LPC analysis using Levinson-Durbin method

SYNOPSIS

lpc [**–l** *L*] [**–m** *M*] [**–f** *F*] [***infile***]

DESCRIPTION

lpc calculates linear prediction coefficients (LPC) from *L*-length framed windowed data from *infile* (or standard input), sending the result to standard output.

For each *L*-length input vector

$$x(0), x(1), \ldots, x(L-1),$$

the autocorrelation function is calculated (see acorr), then the gain K and the linear prediction coefficients

$$K, a(1), \ldots, a(M)$$

are calculated using the Levinson-Durbin algorithm (see levdur).

Input and output data are in float format.

OPTIONS

–l	*L*	frame length	[256]
–m	*M*	order of LPC	[25]
–f	*F*	mimimum value of the determinant of the normal matrix	[0.000001]

EXAMPLE

In this example, the 20-th order linear prediction analysis is applied to input read from *data.f* in float format, and the linear prediction coefficients are written to *data.lpc*:

```
frame < data.f | window | lpc -m 20 > data.lpc
```

SEE ALSO

acorr, levdur, lpc2par, par2lpc, lpc2c, lpc2lsp, lsp2lpc ltcdf, lspdf

NAME

lpc2c – transform LPC to cepstrum

SYNOPSIS

lpc2c **[–m** M_1 **] [–M** M_2 **] [** ***infile*** **]**

DESCRIPTION

lpc2c calculates LPC cepstral coefficients from linear prediction (LPC) coefficients from *infile* (or standard input), sending the result to standard output. That is, when the input sequence is

$$\sigma, a(1), a(2), \ldots, a(p)$$

where

$$H(z) = \frac{\sigma}{A(z)} = \frac{\sigma}{1 + \sum_{k=1}^{P} a(k) z^{-k}}$$

then the LPC cepstral coefficients are evaluated as follows.

$$c(n) = \begin{cases} \ln(h), & n = 0 \\ -a(n) = -\sum_{k=1}^{n-1} \frac{k}{n} c(k) a(n-k), & 1 \le n \le P \\ -\sum_{k=n-P}^{n-1} \frac{k}{n} c(k) a(n-k), & n > P \end{cases}$$

And the sequence of cepstral coefficients

$$c(0), c(1), \ldots, c(M)$$

is given as output. Input and output data are in float format.

OPTIONS

–m	M_1	order of LPC	[25]
–M	M_2	order of cepstrum	[25]

EXAMPLE

In the example below, a 10-th order LPC analysis is undertaken after passing the speech data *data.f* in float format through a window, 15-th order LPC cepstral coefficients are calculated, and the result is written to *data.cep*.

```
frame < data.f | window | lpc -m 10 |\
lpc2c -m 10 -M 15 > data.cep
```

SEE ALSO

lpc, gc2gc, mgc2mgc, freqt

NAME

lpc2lsp – transform LPC to LSP

SYNOPSIS

lpc2lsp [**–m** *M*] [**–s** *S*] [**–k**] [**–l**] [**–o** *O*] [**–n** *N*] [**–p** *P*] [**–q** *Q*] [**–d** *D*] [***infile***]

DESCRIPTION

lpc2lsp calculates line spectral pair (LSP) coefficients from M-th order linear prediction (LPC) coefficients from *infile* (or standard input), sending the result to standard output.

Although the gain K is included in the LPC input vectors as follows

$$K, a(1), \ldots, a(M)$$

K is not used in the calculation of the LSP coefficients.

The M-th order polynomial linear prediction equation $A(z)$ is

$$A_M(z) = 1 + \sum_{m=1}^{M} a(m) z^{-m}$$

The PARCOR coefficients satisfy the following equations.

$$A_m(z) = A_{m-1}(z) - k(m) B_{m-1}(z)$$
$$B_m(z) = z^{-1}(B_{m-1}(z) - k(m) A_{m-1}(z))$$

Also, the initial conditions are set as follows,

$$A_0(z) = 1$$
$$B_0(z) = z^{-1}. \tag{1}$$

When the linear prediction polynomial equation of M-th order $A_M(z)$ are given, and the evaluation of $A_{M+1}(z)$ is obtained with the value of $k(M+1)$ set to 1 or -1, then $P(z)$ and $Q(z)$ are defined as follow.

$$P(z) = A_M(z) - B_M(z)$$
$$Q(z) = A_M(z) + B_M(z)$$

Making $k(M+1)$ equal to ± 1 means that, regarding PARCOR coefficients, the boundary condition for the glottis of the fixed vocal tract model satisfies a perfect reflection characteristic. Also, $A_M(z)$ can be written as

$$A_M(z) = \frac{P(z) + Q(z)}{2}.$$

Also, to make sure the roots of $A_M(z) = 0$ will all be inside the unit circle, i.e. to make sure $A_M(z)$ is stable, the following conditions must be met.

- All of the roots of $P(z) = 0$ and $Q(z) = 0$ are on the unit circle line.
- the roots of $P(z) = 0$ and $Q(z) = 0$ should be above the unit circle line and intercalate.

If we assume that M is an even number, then $P(z)$ and $Q(z)$ can be factorized as follows.

$$P(z) = (1 - z^{-1}) \prod_{i=2,4,\ldots,M} (1 - 2z^{-1}\cos\omega_i + z^{-2})$$

$$Q(z) = (1 + z^{-1}) \prod_{i=1,3,\ldots,M-1} (1 - 2z^{-1}\cos\omega_i + z^{-2})$$

Also, the values of ω_i will satisfy the following ordering condition.

$$0 < \omega_1 < \omega_2 < \cdots < \omega_{M-1} < \omega_M < \pi$$

If M is an odd number, a solution can be found in a similar way.

The coefficients ω_i obtained through factorization are called LSP coefficients.

OPTIONS

–m	*M*	order of LPC	[25]
–s	*S*	sampling frequency (kHz)	[10.0]
–k		output gain	[TRUE]
–l		output log gain instead of linear gain	[FALSE]
–o	*O*	output format	[0]

0 normalized frequency $(0 \ldots \pi)$
1 normalized frequency $(0 \ldots 0.5)$
2 frequency (kHz)
3 frequency (Hz)

Usually, the options below do not need to be assigned.

–n	*N*	split number of unit circle	[128]
–p	*P*	maximum number of interpolation	[4]
–d	*D*	end condition of interpolation	[1e-06]

EXAMPLE

In the following example, speech data is read in float format from *data.f*, 10-th order LPC coefficients are calculated, and the LSP coefficients are evaluated and written to *data.lsp*:

```
frame < data.f | window | lpc -m 10 |\
lpc2lsp -m 10 > data.lsp
```

SEE ALSO

lpc, lsp2lpc, lspdf

NAME

lpc2par – transform LPC to PARCOR

SYNOPSIS

lpc2par [–m *M* **] [–g** *G* **] [–c** *C* **] [–s] [** ***infile*** **]**

DESCRIPTION

lpc2par calculates PARCOR coefficients from M-th order linear prediction (LPC) coefficients from *infile* (or standard input), sending the result to standard output.

The LPC input format is

$$K, a(1), \ldots, a(M),$$

and the PARCOR output format is

$$K, k(1), \ldots, k(M).$$

If the –s option is assigned, the stability of the filter is analyzed. If the filter is stable, then 0 is returned. If the filter is not stable, then 1 is returned to the standard output.

Input and output data are in float format.

The transformation from LPC coefficients to PARCOR coefficients is undertaken as follows:

$$k(m) = a^{(m)}(m)$$
$$a^{(m-1)}(i) = \frac{a^{(m)}(i) + a^{(m)}(m)a^{(m)}(m-i)}{1 - k^2(m)},$$

where $1 \le i \le m-1$, $m = p, p-1, \ldots, 1$. The initial condition is

$$a^{(M)}(m) = a(m), \qquad 1 \le m \le M.$$

If we use the –g option, then the input contains normalized generalized cepstral coefficients with power parameter γ and the output contains the corresponding PARCOR coefficients. In other words, the input is

$$K, c'_\gamma(1), \ldots, c'_\gamma(M)$$

and the initial condition is

$$a^{(M)}(m) = \gamma c'_\gamma(M), \qquad 1 \le m \le M.$$

Also with respect to the stability analysis, the PARCOR coefficients are checked through the following equation.

$$-1 < k(m) < 1$$

If this condition satisfy then the filter is stable.

OPTIONS

–m	*M*	order of LPC	[25]
–g	*G*	gamma of generalized cepstrum $\gamma = G$	[0]
–c	*C*	gamma of generalized cepstrum $\gamma = -1/(\text{int})C$ C must be $C \geq 1$	
–s		check stable or unstable	[FALSE]

EXAMPLE

In the example below, a linear prediction analysis is done in the input file *data.f* in float format, the LPC coefficients are then transformed into PARCOR coefficients, and the output is written to *data.rc*:

```
frame < data.f | window | lpc | lpc2par > data.rc
```

SEE ALSO

acorr, levdur, lpc, par2lpc, ltcdf

NAME

lsp2lpc – transform LSP to LPC

SYNOPSIS

lsp2lpc [–m *M* **] [–s** *S* **] [–k] [–l] [–i** *I* **] [** ***infile*** **]**

DESCRIPTION

lsp2lpc calculates linear prediction (LPC) coefficients from M-th order line spectral pair (LSP) coefficients from *infile* (or standard input), sending the result to standard output.

The LSP input input format is

$$[K], l(1), \ldots, l(M),$$

and the LPC output format is

$$K, a(1), \ldots, a(M).$$

By default, *lsp2lpc* assumes that the LSP input vectors include the gain K, and it passes that gain value through to the LPC output vectors. However, if the –k option is present, *lsp2lpc* assumes that K is not present in the LSP input vectors, and it sets K to 1.0 in the LPC output vectors.

OPTIONS

–m	*M*	order of LPC	[25]
–s	*S*	sampling frequency (kHz)	[10.0]
–k		input & output gain	[TRUE]
–l		regard input as log gain and output linear gain	[FALSE]
–i	*I*	input format	[0]

0 normalized frequency $(0 \ldots \pi)$
1 normalized frequency $(0 \ldots 0.5)$
2 frequency (kHz)
3 frequency (Hz)

EXAMPLE

In the example below, 10-th order LSP coefficients in float format are read from file *data.lsp*, the linear prediction coefficients are evaluated, and written to *data.lpc*:

```
lsp2lpc -m 10 < data.lsp > data.lpc
```

SEE ALSO

lpc, lpc2lsp

NAME

lspcheck – check stability and rearrange LSP

SYNOPSIS

lspcheck **[–m** *M* **] [–s** *S* **] [–k] [–i** *I* **] [–o** *O* **] [–r** *R***] [** ***infile*** **]**

DESCRIPTION

lspcheck tests the stability of the filter corresponding to the line spectral pair (LSP) coefficients from *infile* (or standard input), sending the result to standard output.

By default, the output is the same as the input. When the –c option is given, the output is LSP coefficients that have been rearranged so the filter is stable. If an frame is unstable, an ASCII report of the number of the frame is sent to standard error.

OPTIONS

–m	*M*	order of LPC	[25]
–s	*S*	sampling frequency (kHz)	[10.0]
–k		input & output gain	[TRUE]
–i	*I*	input format	[0]
–o	*O*	output format	[*I*]
		0 normalized frequency $(0 \ldots \pi)$	
		1 normalized frequency $(0 \ldots 0.5)$	
		2 frequency (kHz)	
		3 frequency (Hz)	
–c		rearrange LSP check the distance between two consecutive LSPs and extend the distance (if it is smaller than $R \times \pi/M$)	[N/A]
–r	*R*	threthold of rearrangement of LSP *s.t.* $0 \leq R \leq 1$	[0.0]

EXAMPLE

In the following example, 10-th order LSP coefficients are read from *data.lsp* in float format, stability is checked, the unstable coefficients are rearranged so that they become stable, and the distance between two consecutive LSPs are extended to $\pi/1000$ if it is smaller than $\pi/1000$, and the rearranged LSP coefficients are written to *data.lspr*:

```
lspcheck -m 10 -c -r 0.01 < data.lsp > data.lspr
```

SEE ALSO

lpc, lpc2lsp, lsp2lpc

NAME

lspdf – LSP speech synthesis digital filter

SYNOPSIS

lspdf **[–m** *M* **] [–p** *P* **] [–i** *I* **] [–s** *S* **] [–o** *O* **] [–k] [–l]** ***lspfile*** **[** ***infile*** **]**

DESCRIPTION

lspdf derives an LSP digital filter from the line spectral pair (LSP) coefficients in *lspfile* and uses it to filter an excitation sequence from *infile* (or standard input) and synthesize speech data, sending the result to standard output.

Both input and output files are in float format.

OPTIONS

–m	*M*	order of coefficients	[25]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–k		filtering without gain	[FALSE]
–l		regard input as log gain	[FALSE]

EXAMPLE

In the example below, excitation is generated from the pitch information given in *data.pitch* in float format. This excitation is passed through the LSP synthesis filter constructed from the LSP file *data.lsp*, and the synthesized speech is written to *data.syn*:

```
excite < data.pitch | lspdf data.lsp > data.syn
```

SEE ALSO

lsp, lpc2lsp

NAME

ltcdf – all-pole lattice digital filter for speech synthesis

SYNOPSIS

ltcdf [–m *M* **] [–p** *P* **] [–i** *I* **] [–k]** ***rcfile*** **[** ***infile*** **]**

DESCRIPTION

ltcdf derives an all-pole lattice digital filter from PARCOR coefficients in *rcfile* and uses it to filter an excitation sequence from *infile* (or standard input) and synthesize speech data, sending the result to standard output.

Both input and output files are in float format.

OPTIONS

–m	*M*	order of coefficients	[25]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–k		filtering without gain	[FALSE]

EXAMPLE

In the example below, excitation is generated from the pitch information given in *data.pitch* in float format. This excitation is passed through the lattice filter constructed from the LPC file *data.rc*, and the synthesized speech is written to *data.syn*:

```
excite < data.pitch | ltcdf data.k > data.syn
```

SEE ALSO

lpc, acorr, levdur, lpc2par, par2lpc, poledf, zerodf, lspdf

NAME

mc2b – transform mel-cepstrum to MLSA digital filter coefficients

SYNOPSIS

mc2b **[–a** *A* **] [–m** *M* **] [** ***infile*** **]**

DESCRIPTION

mc2b calculates MLSA filter coefficients $b(m)$ from mel-cepstral coefficients $c_\alpha(m)$ from *infile* (or standard input), sending the result to standard output.

Both input and output files are in float format.

The coefficients are given as follows:

$$b(m) = \begin{cases} c_\alpha(M), & m = M \\ c_\alpha(m) - \alpha b(m+1), & 0 \le m < M \end{cases}$$

These coefficients $b(m)$ can be directly used in the implementation of a MLSA filter. *mc2b* implements the inverse transformation undertaken by the command b2mc.

OPTIONS

–a	*A*	all-pass constant α	[0.35]
–m	*M*	order of mel-cepstrum	[25]

EXAMPLE

In the example below, speech data is read in float format from *data.f*, a 12-th order mel-cepstral analysis is undertaken, these mel-cepstral coefficients are transformed into MLSA filter coefficients, and then the coefficients $b(m)$ are written to *data.b*:

```
frame < data.f | window | mcep -m 12 |\
mc2b -m 12 > data.b
```

SEE ALSO

mlsadf, mglsadf, b2mc, mcep, mgcep, amcep

NAME

mcep – mel cepstral analysis(10; 12)

SYNOPSIS

mcep **[–a** *A* **] [–m** *M* **] [–l** *L* **] [–q** *Q* **] [–i** *I* **] [–j** *J* **] [–d** *D* **] [–e** *e* **] [–E** *E* **] [–f** *F* **] [** ***infile*** **]**

DESCRIPTION

mcep uses mel-cepstral analysis to calculate mel-cepstral coefficients $c_\alpha(m)$ from *L*-length framed windowed data from *infile* (or standard input), sending the result to standard output.

Input and output data are in float format.

In the mel-cepstral analysis, the spectrum of the speech signal is modeled by *M*-th order mel-cepstral coefficients $c_\alpha(m)$ as follows.

$$H(z) = \exp \sum_{m=0}^{M} c_\alpha(m) \tilde{z}^{-m}$$

The command "mcep" applies a cost function based on the unbiased log spectrum estimation method. The variable $\tilde{z}^{-1}$ can be expressed as the following first order all-pass function

$$\tilde{z}^{-1} = \frac{z^{-1} - \alpha}{1 - \alpha z^{-1}}.$$

The phase characteristic is given by the variable α. For a sampling rate of 16 kHz, α is set to 0.42. For a sampling rate 10 kHz, α is set to 0.35. For a sampling rate 8 kHz, α is set to 0.31. By making these choices for α, the mel-scale becomes a good approximation to the human sensitivity to the loudness of speech.

The Newton-Raphson method is used to minimize the cost function when evaluating mel-cepstral coefficients.

OPTIONS

–a	*A*	all-pass constant α	[0.35]
–m	*M*	order of mel cepstrum	[25]
–l	*L*	frame length	[256]
–q	*Q*	input data style	[0]

$Q = 0$ windowed data sequence
$Q = 1$ $20 \times \log |f(w)|$
$Q = 2$ $\ln |f(w)|$
$Q = 3$ $|f(w)|$
$Q = 4$ $|f(w)|^2$

Usually, the options below do not need to be assigned.

–i	*I*	minimum iteration of Newton-Raphson method	[2]
–j	*J*	maximum iteration of Newton-Raphson method	[30]
–d	*D*	end condition of Newton-Raphson	[0.001]
–e	*e*	small value added to periodgram	[0.0]
–E	*E*	floor in db calculated per frame	[N/A]
–f	*F*	minimum value of the determinant of the normal matrix	[0.000001]

EXAMPLE

In the example below, speech data is read in float format from *data.f* and analyzed. Then, mel-cepstral coefficients are written to *data.mcep*:

```
frame < data.f | window | mcep > data.mcep

frame < data.f | window | fftr -A -H | mcep -q 3 > data.mcep
```

SEE ALSO

uels, gcep, mgcep, mlsadf

NAME

merge – data merge

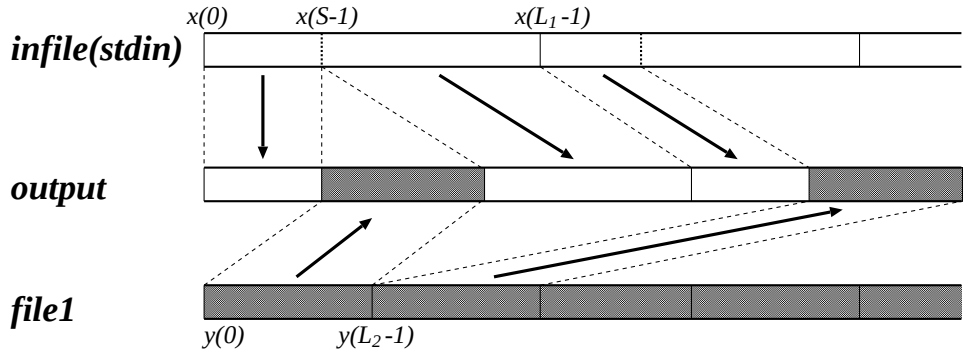
SYNOPSIS

merge **[–s** S **] [–l** L_1 **] [–n** N_1 **] [–L** L_2 **] [–N** N_2 **]**
[–o] [+*type*]*file1* [*infile*]

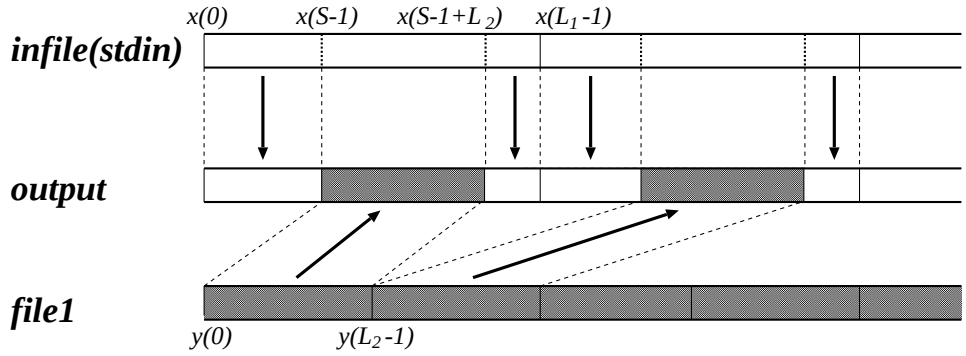
DESCRIPTION

merge merges, on a frame-by-frame basis, data from *file1* into the data from *infile* (or standard input), sending the result to standard output, as described below.

Insert mode



Overwrite mode



OPTIONS

–s	S	insert point	[0]
–l	L_1	frame length of input data	[25]
–n	N_1	order of input data	[$L_1 - 1$]
–L	L_2	frame length of insert data	[10]
–N	N_2	order of insert data	[$L_2 - 1$]

–o	overwrite mode	[FALSE]
+t	input data format	[f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)

EXAMPLE

The following example inserts blocks of 2 samples from *data.f2* in short format into *data.f1*, also in short format. The frame length of the file *data.f1* is 3, and the blocks from *data.f2* will be inserted from the 3rd sample of every frame. The result is written to *data.merge*.

```
merge +f -s 2 -l 3 -L 2 +s data.f2 < data.f1 > data.merge
```

For example, if the *data.f1* file is given by

$$1, 1, 1, 2, 2, 2, \ldots$$

, and the *data.f2* file is given by

$$2, 3, 5, 6, \ldots$$

then the output *data.merge* will be

$$1, 1, 2, 3, 1, \ 2, 2, 5, 6, 2, \ldots$$

The next example overwrites blocks of 2 samples from *data.f2* in long format into *data.f1*, also in long format, the frame length of the file *data.f1* is 4, and the blocks from *data.f2* will be inserted from the 2nd sample of every frame. The result is *data.merge*.

```
merge +f -s 2 -l 4 -L 2 +l -o data.f2 < data.f1 > data.merge
```

For example, if the *data.f1* file is given by

$$1, 1, 1, 1, 2, 2, 2, 2, \ldots$$

, and the *data.f2* file is given by

$$3, 4, 5, 6, \ldots$$

then the output *data.merge* will be

$$1, 3, 4, 1, \ 2, 5, 6, 2, \ldots$$

SEE ALSO

bcp

NAME

mfcc – mel-frequency cepstral analysis

SYNOPSIS

mfcc **[–a** *A* **] [–e** *E* **] [–l** L_1 **] [–L** L_2 **] [–s or –f** *F* **] [–m** *M* **]**
[–n *N* **] [–s** *S* **] [–w** *W***] [–d] [– E] [–0][** ***infile*** **]**

DESCRIPTION

mfcc uses mel-frequency cepstral analysis to calculate mel-frequency cepstrum from L_1-length framed data from *infile* (or standard input), sending the result to standard output.Since *mfcc* can apply a window function to input data in the function, it is not necessary to use windowed data as input. The input time domain sequence of length L_1 is of the form:

$$x(0), x(1), \ldots, x(L_1 - 1)$$

Also, note that the input and output data are in float format, and that the output data cannot be used for speech synthesis through the MLSA filter.

OPTIONS

–a	*A*	preemphasise coefficient	[0.97]
–c	*C*	liftering coefficient	[22]
–e	*E*	flooring value for calculating $\log(x)$ in filterbank analysis if $x < E$ then return $x = E$	[1.0]
–l	L_1	frame length of input	[256]
–L	L_2	frame length for fft. default value 2^n satisfies $L_1 < 2^n$	[2^n]
–m	*M*	order of mfcc	[12]
–n	*N*	order of channel for mel-filter bank	[20]
–s	*S*	sampling frequency (kHz)	[16.0]
–w	*W*	type of window	[0]
		0 Hamming 1 Do not use a window function	
–d		use dft (without using fft) for dct	[FALSE]
–E		output energy	[FALSE]
–0		output 0'th static coefficient	[FALSE]

if the -E or -0 option is given, energy E or 0'th static coefficient $C0$ is outputted as follows.

$$mc(0), mc(1), \ldots, mc(m-1), E(C0)$$

Also, if both -E and -0 option are given, the output is as follows.

$$mc(0), mc(1), \ldots, mc(m-1), C0, E$$

EXAMPLE

In the example below, speech data in float format is read from *data.f*. Here, we specify the frame length, frame shift and sampling frequency as 40ms, 10ms and 16kHz, respectivelly. The 12 order mel-frequency cepstral coefficients, together with the energy component, are outputted to *data.mfc*.

```
frame -l 640 -p 160  data.f |\
mfcc -l 640 -m 12 -s 16 -E > data.mfc
```

Also, in case we want to calculate the coefficients the same way as in HTK, following the conditions:

```
SOURCEFORMAT = NOHEAD
SOURCEKIND = WAVEFORM
SOURCERATE = 625      # Sampling rate (1 / 16000 * 10^7)
TARGETKIND = MFCC_D_A_E
TARGETRATE = 100000   # Frame shift (ns)
WINDOWSIZE = 400000   # Frame length (ns)
DELTAWINDOW = 1       # Delta widndow size
ACCWINDOW = 1         # Accelaration widndow size
ENORMALISE = FALSE
```

We have to use the following command in SPTK. Below, because of the difference of the calcuration method of regression coefficients between SPTK and HTK, differencial coefficients are specified directly using -d option in *delta* command.

```
frame -l 640 -p 160  data.f |\
mfcc -l 640 -m 12 -s 16 -E > data.mfc
delta -m 12 -d -0.5 0 0.5 |\
-d 0.25 0 -0.5 0 0.25 data.mfc > data.mfc.diff
```

Here, because of the difference in the calculation method of regression coefficients between SPTK and HTK, differencial coefficients are specified directly using the –d option in *delta* dommand. The correspondence between the option of SPTK's command option and the HTK's configuration for extracting mel-frequency cepstrum is shown in Table 2. Please, refer to the HTKBook for more information on extracting mel-frequency cepstrum with HTK.

SEE ALSO

frame, gcep, mcep, mgcep, spec

Table 2: Configuration for extracting MFCC

Settings	SPTK	HTK
pre-emphasis coefficient	-a (at *mfcc* command)	PREEMCOEF
liftering coefficient	-c (at *mfcc* command)	CEPLIFTER
small value for calculating log()	-e (at *mfcc* command)	N/A
sampling rate	-s (at *mfcc* command)	SOURCERATE
frame shift	-p (at *frame* command)	TARGETRATE
frame length of input	-l (at *frame* command) -l (at *mfcc* command)	WINDOWSIZE
frame length for fft	-L (at *mfcc* command)	N/A (automatically calculated)
order of cepstrum	-m (at *mfcc* command)	NUMCEPS
order of channel for mel-filter bank	-n (at *mfcc* command)	NUMCHANS
use hamming window	-w (at *mfcc* command)	USEHAMMING
use dft	-d (at *mfcc* command)	N/A
output energy	-E (at *mfcc* command)	TARGETKIND
output 0'th static coefficient	-0 (at *mfcc* command)	TARGETKIND
delta window size	-r (at *delta* command)	DELTAWINDOW
acceleration window size	-r (at *delta* command)	ACCWINDOW
Normalize log energy	N/A	ENORMALISE

NAME

mgc2mgc – frequency and generalized cepstral transformation

SYNOPSIS

mgc2mgc **[–m** M_1 **]** **[–a** A_1 **]** **[–g** G_1 **]** **[–c** C_1 **]** **[–n]** **[–u]**
[–M M_2 **]** **[–A** A_2 **]** **[–G** G_2 **]** **[–C** C_2 **]** **[–N]** **[–U]** **[** ***infile*** **]**

DESCRIPTION

mgc2mgc transforms mel-generalized cepstral coefficients $c_{\alpha_1,\gamma_1}(0), \ldots, c_{\alpha_1,\gamma_1}(M_1)$ from *infile* (or standard input) into a different set of mel-generalized cepstral coefficients $c_{\alpha_2,\gamma_2}(0), \ldots, c_{\alpha_2,\gamma_2}(M_2)$ sending the result to standard output.

α characterizes the frequency-warping transform, while γ characterizes the generalized log magnitude transform.

Input and output data are in float format.

First, a frequency transformation ($\alpha_1 \rightarrow \alpha_2$) is undertaken in the input mel-generalized cepstral coefficients $c_{\alpha_1,\gamma_1}(m)$, and $c_{\alpha_2,\gamma_1}(m)$ is calculated as follows.

$$\alpha = (\alpha_2 - \alpha_1)/(1 - \alpha_1\alpha_2)$$

$$c^{(i)}_{\alpha_2,\gamma_1}(m) = \begin{cases} c_{\alpha_1,\gamma_1}(-i) + \alpha\, c^{(i-1)}_{\alpha_2,\gamma_1}(0), & m = 0 \\ (1-\alpha^2)\, c^{(i-1)}_{\alpha_2,\gamma_1}(0) + \alpha\, c^{(i-1)}_{\alpha_2,\gamma_1}(1), & m = 1 \\ c^{(i-1)}_{\alpha_2,\gamma_1}(m-1) + \alpha \left(c^{(i-1)}_{\alpha_2,\gamma_1}(m) - c^{(i)}_{\alpha_2,\gamma_1}(m-1) \right), & m = 2, \ldots, M_2 \end{cases}$$
$$i = -M_1, \ldots, -1, 0$$

Then the gain is normalized and $c'_{\alpha_2,\gamma_1}(m)$ is evaluated.

$$K_{\alpha_2} = s^{-1}_{\gamma_1}\left(c^{(0)}_{\alpha_2,\gamma_1}(0)\right),$$
$$c'_{\alpha_2,\gamma_1}(m) = c^{(0)}_{\alpha_2,\gamma_1}(m) / \left(1 + \gamma_1\, c^{(0)}_{\alpha_2,\gamma_1}(0)\right), \qquad m = 1, 2, \ldots, M_2$$

Afterwards, $c'_{\alpha_2,\gamma_1}(m)$ is transformed into $c'_{\alpha_2,\gamma_2}(m)$ through a generalized log transformation ($\gamma_1 \rightarrow \gamma_2$).

$$c'_{\alpha_2,\gamma_2}(m) = c'_{\alpha_2,\gamma_1}(m) + \sum_{k=1}^{m-1} \frac{k}{m} \left\{ \gamma_2\, c_{\alpha_2,\gamma_1}(k)\, c'_{\alpha_2,\gamma_2}(m-k) - \gamma_1\, c_{\alpha_2,\gamma_2}(k)\, c'_{\alpha_2,\gamma_1}(m-k) \right\},$$
$$m = 1, 2, \ldots, M_2$$

Finally, the gain is inversely normalized and $c_{\alpha_2,\gamma_2}(m)$ is calculated.

$$c_{\alpha_2,\gamma_2}(0) = s_{\gamma_2}\left(K_{\alpha_2}\right),$$
$$c_{\alpha_2,\gamma_2}(m) = c'_{\alpha_2,\gamma_2}(m) \left(1 + \gamma_2\, c_{\alpha_2,\gamma_2}(0)\right), \qquad m = 1, 2, \ldots, M_2$$

In case we represent input and output with γ, if the coefficients $c_{\alpha,\gamma}(m)$ are not normalized, then the following representation is assumed

$$1 + \gamma c_{\alpha,\gamma}(0), \gamma c_{\alpha,\gamma}(1), \ldots, \gamma c_{\alpha,\gamma}(M),$$

if they are normalized, then the following representation is assumed

$$K_\alpha, \gamma c'_{\alpha,\gamma}(1), \ldots, \gamma c'_{\alpha,\gamma}(M).$$

OPTIONS

–m	M_1	order of mel-generalized cepstrum (input)	[25]
–a	A_1	alpha of mel-generalized cepstrum (input)	[0]
–g	G_1	gamma of mel-generalized cepstrum (input) $\gamma_1 = G_1$	[0]
–c	C_1	gamma of mel-generalized cepstrum (input) $\gamma_1 = -1/(\text{int})C_1$ C_1 must be $C_1 \geq 1$	
–n		regard input as normalized mel-generalized cepstrum	[FALSE]
–u		regard input as multiplied by gamma	[FALSE]
–M	M_2	order of mel-generalized cepstrum (output)	[25]
–A	A_2	alpha of mel-generalized cepstrum (output)	[0]
–G	G_2	gamma of mel-generalized cepstrum (output) $\gamma_2 = G_2$	[1]
–C	C_2	gamma of mel-generalized cepstrum (output) $\gamma_2 = -1/(\text{int})G_2$ C_2 must be $C_2 \geq 1$	
–N		regard output as normalized mel-generalized cepstrum	[FALSE]
–U		regard input as multiplied by gamma	[FALSE]

EXAMPLE

In the example below, 12-th order LPC coefficients are read in float format from *data.lpc*, and 30-th order mel-cepstral coefficients are calculated and written to *data.mcep*:

```
mgc2mgc -m 12 -a 0 -g -1 -M 30 -A 0.31 -G 0
                        < data.lpc > data.mcep
```

SEE ALSO

uels, gcep, mcep, mgcep, gc2gc, freqt, lpc2c

NAME

mgc2mgclsp – transform MGC to MGC-LSP

SYNOPSIS

mgc2mgclsp **[–a** *A*] **[–g** *G*] **[–m** *M*] **[–o** *O*] **[–s** *S*] **[–k**] **[–l**] **[** ***infile*** **]**

DESCRIPTION

mgc2mgc transforms mel-generalized cepstral coefficients $c_{\alpha,\gamma}(0), \ldots, c_{\alpha,\gamma}(M)$ from *infile* (or standard input) into line spectral pair coefficients (MGC-LSP) $K, l(1), \ldots, l(M)$ sending the result to standard output.

α characterizes the frequency-warping transform, while γ characterizes the generalized log magnitude transform and K is the gain.

mgc2mgclsp does not check for stability of the MGC-LSP. One should use the command *lspcheck* to check the stability of the MGC-LSP.

OPTIONS

–a	*A*	alpha of mel-generalized cepstrum	[0.35]
–g	G_1	gamma of mel-generalized cepstrum $\gamma = G$	[-1]
–c	C_1	gamma of mel-generalized cepstrum (input) $\gamma = -1/(\text{int})C$ C must be $C \geq 1$	
–m	*M*	order of mel-generalized cepstrum	[25]
–o	*O*	output format	[0]
		0 normalized frequency $(0 \ldots \pi)$ 1 normalized frequency $(0 \ldots 0.5)$ 2 frequency (kHz) 3 frequency (Hz)	
–s	*S*	sampling frequency (kHz)	[10]
–k		do not output gain	[FALSE]
–l		output log gain instead of linear gain	[FALSE]

EXAMPLE

In the following example, speech data is read in float format from *data.f*, analyzed with $\alpha = 0.35, \gamma = -1$ and the MGC-LSP coefficients are evaluated and written to *data.mgclsp*:

```
frame < data.f | window | mgcep -a 0.35 -g -1 |\
mgc2mgclsp -a 0.35 -g -1 > data.mgclsp
```

Also, MGC-LSP stability can be checked by using the following:

```
frame < data.f | window | mgcep -a 0.35 -g -1 |\
mgc2mgclsp -a 0.35 -g -1 | lspcheck -r 0.01 > data.mgclsp
```

SEE ALSO

lpc, lsp2lpc, lspcheck, mgc2mgc, mgcep

NAME

mgc2sp – transform mel-generalized cepstrum to spectrum

SYNOPSIS

mgc2sp **[–a** *A* **] [–g** *G* **] [–c** *C* **] [–m** *M* **] [–n] [–u] [–l** *L* **] [–p]**
[–o *O* **] [** ***infile*** **]**

DESCRIPTION

mgc2sp calculates the log magnitude spectrum from mel-generalized cepstral coefficients $c_{\alpha,\gamma}(m)$ from *infile* (or standard input), sending the result to standard output.

Input and output data are in float format.

The mel-generalized cepstral coefficients $c_{\alpha,\gamma}(m)$ are transformed into cepstral coefficients (refer to mgc2mgc) and then the log magnitude spectrum is calculated (refer to spec).

When the input data is normalized by the gain, it can be expressed as follows.

$$\begin{aligned} K_\alpha &= s_\gamma^{-1}\left(c^{(0)}_{\alpha,\gamma}(0)\right), \\ c'_{\alpha,\gamma}(m) &= c^{(0)}_{\alpha,\gamma}(m) / \left(1 + \gamma\, c^{(0)}_{\alpha,\gamma}(0)\right), \qquad m = 1, 2, \ldots, M \end{aligned}$$

Supposing the input data is represented by γ for non-normalized coefficients $c_{\alpha,\gamma}(m)$, the following representation is assumed

$$1 + \gamma c_{\alpha,\gamma}(0), \gamma c_{\alpha,\gamma}(1), \ldots, \gamma c_{\alpha,\gamma}(M)$$

and the following representation is assumed for normalized coefficients

$$K_\alpha, \gamma c'_{\alpha,\gamma}(1), \ldots, \gamma c'_{\alpha,\gamma}(M)$$

OPTIONS

–a	*A*	alpha α	[0]
–g	*G*	power parameter γ of mel-generalized cepstrum $\gamma = G$	[0]
–c	*C*	power parameter γ of mel-generalized cepstrum $\gamma = -1/(\text{int})C$ C must be $C \geq 1$	
–m	*M*	order of mel-generalized cepstrum	[25]
–n		regard input as normalized cepstrum	[FALSE]
–u		regard input as multiplied by γ	[FALSE]
–l	*L*	FFT length	[256]

–p output phase [FALSE]
–o *O* output format [0]
if the –p option is assigned, scale of output spectrum can be assigned.

$O = 0$ $20 \times \log |H(z)|$
$O = 1$ $\ln |H(z)|$
$O = 2$ $|H(z)|$
$O = 3$ $|H(z)|^2$

if the –p option is not assigned, unit of output phase can be assigned.

$O = 0$ $\arg |H(z)| \div \pi$ $[\pi\ rad.]$
$O = 1$ $\arg |H(z)|$ $[rad.]$
$O = 2$ $\arg |H(z)| \times 180 \div \pi$ $[deg.]$

EXAMPLE

In the following example, mel-generalized cepstral coefficients in float format are read from *data.mgcep* ($M = 12, \alpha = 0.35, \gamma = -0.5$) and the log magnitude spectrum is evaluated and plotted:

```
mgc2sp -m 12 -a 0.35 -c 2 < data.mgcep | glogsp | xgr
```

SEE ALSO

c2sp, mgc2mgc, gc2gc, freqt, gnorm, lpc2c

NAME

mgcep – mel-generalized cepstral analysis(13; 14)

SYNOPSIS

mgcep **[–a** *A* **]** **[–g** *G* **]** **[–c** *C* **]** **[–m** *M* **]** **[–l** *L* **]** **[–q** *Q* **]** **[–o** *O* **]**
[–i *I* **]** **[–j** *J* **]** **[–d** *D* **]** **[–p** *P* **]** **[–e** *e* **]** **[–E** *E* **]** **[–f** *F* **]** **[** ***infile*** **]**

DESCRIPTION

mgcep uses mel-generalized cepstral analysis to calculate mel-generalized cepstral coefficients from *L*-length framed windowed input data from *infile* (or standard input), sending the result to standard output. There are several different output formats, controlled by the –o option.

Considering an input signal of length L, the time sequence is presented by

$$x(0), x(1), \ldots, x(L-1)$$

Input and output data are in float format.

In the mel-generalized cepstral analysis, the spectrum of the speech signal is modeled by M-th order mel-generalized cepstral coefficients $c_{\alpha,\gamma}(m)$ as expressed below:

$$\begin{aligned} H(z) &= s_\gamma^{-1}\left(\sum_{m=0}^{M} c_{\alpha,\gamma}(m) z^{-m}\right) \\ &= \begin{cases} \left(1+\gamma\sum_{m=1}^{M} c_{\alpha,\gamma}(m)\tilde{z}^{-m}\right)^{1/\gamma}, & -1 \le \gamma < 0 \\ \exp\sum_{m=1}^{M} c_{\alpha,\gamma}(m)\tilde{z}^{-m}, & \gamma = 0 \end{cases} \end{aligned}$$

For this command "mgcep", a cost function based on the unbiased estimation log spectrum method is applied. The variable $\tilde{z}^{-1}$ can be expressed as the following first order all-pass function

$$\tilde{z}^{-1} = \frac{z^{-1}-\alpha}{1-\alpha z^{-1}}$$

The phase characteristic is represented by the variable α. For a sampling rate 10kHz, α is made equal to 0.35. For a sampling rate 8kHz, α is made equal to 0.31. By setting α to these values, the mel-scale becomes a good approximation to the human sensitivity to the loudness of speech.

The Newton-Raphson method is used to minimize the cost function when evaluating mel-cepstral coefficients.

The mel-generalized cepstral analysis includes several other methods to analyze speech, depending on the values of α and γ (refer to figure 1).

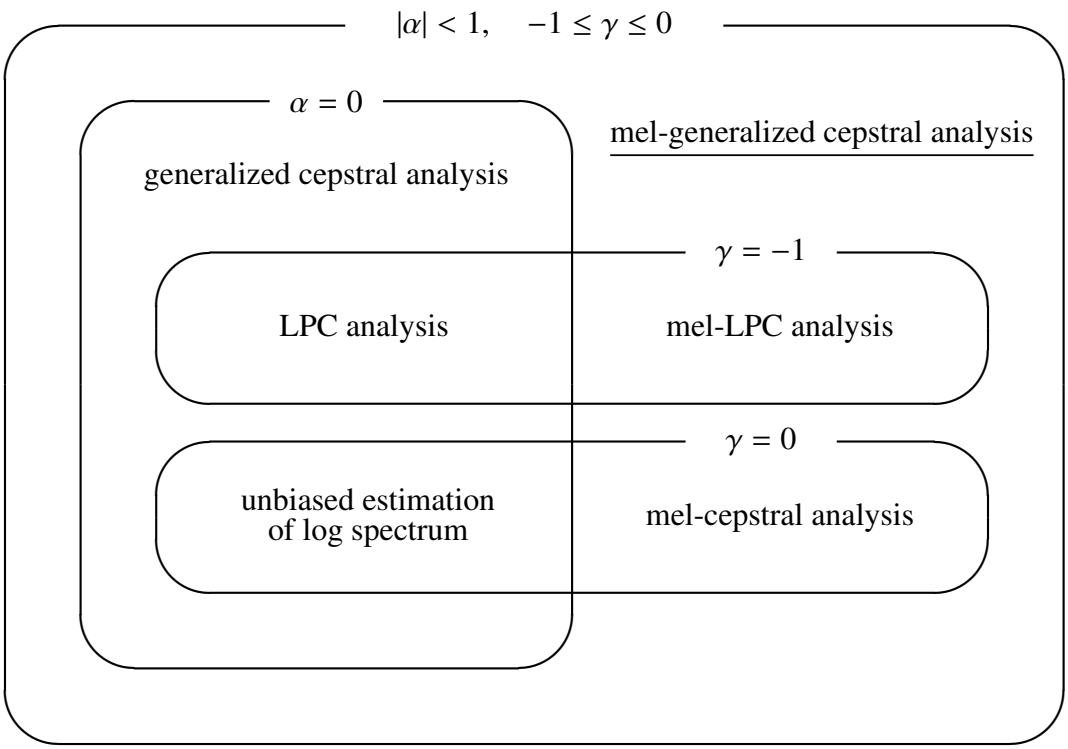


Figure 1: mel-generalized cepstral analysis and other method relations

OPTIONS

–a	*A*	alpha α	[0.35]
–g	*G*	power parameter of generalized cepstrum γ $\gamma = G$	[0]
–c	*C*	power parameter of generalized cepstrum γ $\gamma = -1/(\text{int})C$ C must be $C \geq 1$	
–m	*M*	order of mel-generalized cepstrum	[25]
–l	*L*	frame length power of 2	[256]
–q	*Q*	input data style	[0]

$Q = 0$ windowed data sequence
$Q = 1$ $20 \times \log |f(w)|$
$Q = 2$ $\ln |f(w)|$
$Q = 3$ $|f(w)|$
$Q = 4$ $|f(w)|^2$

–o *O* output format [0]

$O = 0$ $c_{\alpha,\gamma}(0), c_{\alpha,\gamma}(1), \ldots, c_{\alpha,\gamma}(M)$
$O = 1$ $b_{\gamma}(0), b_{\gamma}(1), \ldots, b_{\gamma}(M)$
$O = 2$ $K_{\alpha}, c'_{\alpha,\gamma}(1), \ldots, c'_{\alpha,\gamma}(M)$
$O = 3$ $K, b'_{\gamma}(1), \ldots, b'_{\gamma}(M)$
$O = 4$ $K_{\alpha}, \gamma\, c'_{\alpha,\gamma}(1), \ldots, \gamma\, c'_{\alpha,\gamma}(M)$
$O = 5$ $K, \gamma\, b'_{\gamma}(1), \ldots, \gamma\, b'_{\gamma}(M)$

Usually, the options below do not need to be assigned.

–i	*I*	minimum iteration of Newton-Raphson method	[2]
–j	*J*	maximum iteration of Newton-Raphson method	[30]
–d	*D*	end condition of Newton-Raphson method	[0.001]
–p	*P*	order of recursions	[$L-1$]
–e	*e*	small value added to periodogram	[0]
–E	*E*	floor in db calculated per frame	[N/A]
–f	*F*	mimimum value of the determinant of the normal matrix	[0.000001]

EXAMPLE

In the following example, speech data is read in float format from *data.f* and analyzed with $\gamma = 0$, $\alpha = 0$ (which correspond to UELS method for log spectrum estimation) and the resulting cepstral coefficients are written *data.cep*:

```
frame < data.f | window | mgcep > data.cep
```

In a similar way, mel-cepstral coefficients can be obtained by

```
frame < data.f | window | mgcep -a 0.35 > data.mcep
```

And linear prediction coefficients can be obtained by

```
frame < data.f | window | mgcep -g -1 -o 5 > data.lpc
```

In this case, the linear prediction coefficients are represented as

$$K, a(1), a(2), \ldots, a(M)$$

In the following example, speech data in float format is read from *data.f*, and analyzed with $\gamma = 0$, $\alpha = 0$ (which correspond to UELS method for log spectrum estimation). The resulting cepstral coefficients are written to *data.cep*:

```
frame < data.f | window | \
fftr -A -H | mgcep -q 3 > data.cep
```

SEE ALSO

uels, gcep, mcep, freqt, gc2gc, mgc2mgc, gnorm, mglsadf

NAME

mgclsp2mgc – transform MGC-LSP to MGC

SYNOPSIS

mgclsp2mgc [**–a** *A*] [**–g** *G*] [**–m** *M*] [**–i** *I*] [**–s** *S*] [**–l**] [***infile***]

DESCRIPTION

mgclsp2mgc transforms M-th order line spectral pair coefficients (MGC-LSPs)

$$K, l(1), \ldots, l(M)$$

read from *infile* (or standard input) into mel-generalized cepstrum coefficients

$$c_{\alpha,\gamma}(0), \ldots, c_{\alpha,\gamma}(M), a$$

sending the result to standard output.

α characterizes the frequency-warping transform, while γ characterizes the generalized log magnitude transform and K represents the gain.

Also, *mgclsp2mgc* does not check the stability of MGC-LSPs. If it is necessary to use the *lspcheck* command for checking the stability of the input MGC-LSPs and then generating the mel-generalized cepstrum coefficients.

OPTIONS

–a	*A*	alpha of mel-generalized cepstrum	[0.35]
–g	G_1	gamma of mel-generalized cepstrum $\gamma = G$	[-1]
–c	C_1	gamma of mel-generalized cepstrum (input) $\gamma = -1/(\text{int})C$ C must be $C \geq 1$	
–m	*M*	order of mel-generalized cepstrum	[25]
–i	*I*	input format	[0]
		0 normalized frequency $(0 \ldots \pi)$ 1 normalized frequency $(0 \ldots 0.5)$ 2 frequency (kHz) 3 frequency (Hz)	
–s	*S*	sampling frequency (kHz)	[10]
–l		regard input as log gain and output linear gain	[FALSE]

EXAMPLE

In the following example, MGC-LSP is read in float format from *data.mgclsp*, and analyzed with $\alpha = 0.35, \gamma = -1$. The mel-generalized cepstrum coefficients are evaluated and written to *data.mgc*:

```
mgclsp2mgc -a 0.35 -g -1 data.mgclsp > data.mgc
```

Also, stability of MGC-LSP's can be checked using the following command:

```
lspcheck -r 0.01 data.mgclsp | \
mgclsp2mgc -a 0.35 -g -1 > data.mgc
```

SEE ALSO

lpc, lsp2lpc, lspcheck, mgc2mgc, mgcep

NAME

mglsadf – MGLSA digital filter for speech synthesis(21; 22)

SYNOPSIS

mglsadf **[–m** *M* **] [–a** *A* **] [–c** *C* **] [–p** *P* **] [–i** *I* **] [–v] [–t] [–k] [–P** *Pa* **]**
mgcfile **[** ***infile*** **]**

DESCRIPTION

mglsadf derives a Mel-Generalized Log Spectral Approximation digital filter from mel-generalized cepstral coefficients $c_{\alpha,\gamma}(m)$ in *mgcfile* and uses it to filter an excitation sequence from *infile* (or standard input) to synthesize speech data, sending the result to standard output.

Input and output data are in float format.

The transfer function $H(z)$ related to the synthesis filter is obtained from the M-th order mel-generalized cepstral coefficients $c_{\alpha,\gamma}(m)$ as expressed below:

$$H(z) = s_\gamma^{-1}\left(\sum_{m=0}^{M} c_{\alpha,\gamma}(m)\tilde{z}^{-m}\right) \tag{1}$$

$$= \begin{cases} \left(1 + \gamma \sum_{m=0}^{M} c_{\alpha,\gamma}(m)\tilde{z}^{-m}\right)^{1/\gamma}, & 0 < \gamma \leq -1 \\ \exp \sum_{m=0}^{M} c_{\alpha,\gamma}(m)\tilde{z}^{-m}, & \gamma = 0 \end{cases}$$

where

$$\tilde{z}^{-1} = \frac{z^{-1} - \alpha}{1 - \alpha z^{-1}}$$

The transfer function $H(z)$ can be rewritten as

$$H(z) = s_\gamma^{-1}\left(\sum_{m=0}^{M} b'_\gamma(m)\Phi_m(z)\right)$$

$$= K \cdot D(z) \tag{2}$$

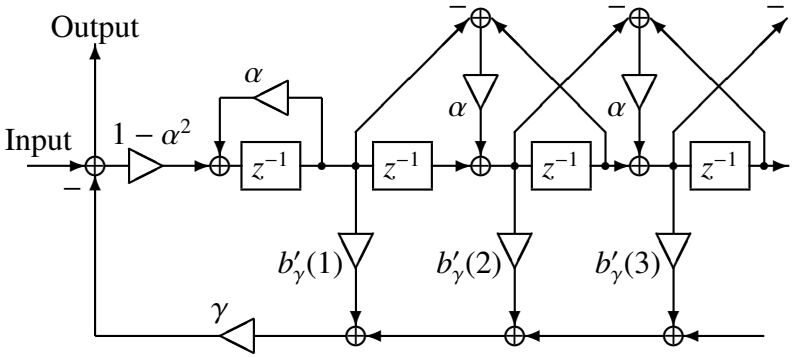
where

$$\Phi_m(z) = \begin{cases} 1, & m = 0 \\ \dfrac{(1-\alpha^2)z^{-1}}{1 - \alpha z^{-1}}\tilde{z}^{-(m-1)}, & m \geq 1 \end{cases}$$

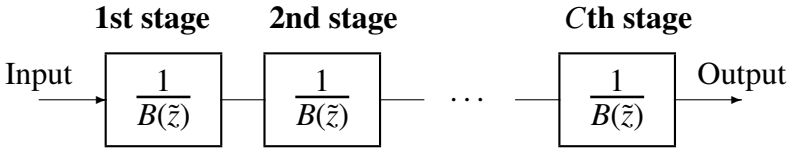
and

$$K = s_\gamma^{-1}(b_\gamma(0))$$

$$D(z) = s_\gamma^{-1}\left(\sum_{m=1}^{M} b_\gamma(m)\Phi_m(z)\right)$$



(a) Structure of filter $1/B(z)$



(b) C level cascaded filter $1/B(z)$

Figure 1: Realization synthesis filter $D(z)$

Also, the coefficients $b'_\gamma(m)$ are obtained from the coefficients $c_{\alpha,\gamma}(m)$ by applying normalization (refer to gnorm), and a linear transformation (refer to mc2b and b2mc). Here we consider only cases where the power parameter is represented by $\gamma = -1/C$, where C is a natural number. In this case the filter $D(z)$ is constructed as shown in figure (b), where each filter of the C level cascaded filter is constructed as shown in figure (a), and can be expressed as

$$\frac{1}{B(\tilde{z})} = \frac{1}{1 + \gamma \sum_{m=1}^{M} b'_\gamma(m)\Phi_m(z)}$$

OPTIONS

–m	*M*	order of mel-generalized cepstrum	[25]
–a	*A*	alpha	[0.35]
–c	*C*	power parameter $\gamma = -1/C$ of generalized cepstrum if $C == 0$, the MLSA filter is used	[1]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–v		inverse filter	[FALSE]
–t		transpose filter	[FALSE]
–k		filtering without gain	[FALSE]

The option below only works if $C == 0$.

–P	*Pa*	order of the Padé approximation *Pa* should be 4 or 5	[4]

EXAMPLE

In the following example, the excitation is constructed from pitch data read in float format from *data.pitch*, and passed through an MGLSA filter built from the mel-generalized cepstrum in *data.mgcep*. The synthesized speech is then written to *data.syn*:

```
excite < data.pitch | mglsadf data.mgcep > data.syn
```

SEE ALSO

mgcep, poledf, zerodf, ltcdf, lmadf, mlsadf, glsadf

NAME

minmax – find minimum and maximum values

SYNOPSIS

minmax **[–l** *L* **] [–n** *N* **] [–b** *B* **] [–o** *O* **] [–d] [** ***infile*** **]**

DESCRIPTION

minmax determines the *B* (default 1) minimum and maximum values, on a frame-by-frame basis, of the data from *infile* (or standard input), sending the result to standard output. If the frame length *L* is 1, each input number is considered to be both the minimum and maximum value for its length-1 frame.

The input format is float by default. If the –d option is not given, the output format will also be float, consisting of the minimum and maximum values. If the –d option is given, the output format will be ASCII, showing the positions within the frame where the minimum and maximum values occurred, as follows:

$$value : position_0, position_1, \ldots$$

Also, when specifying –o 0, –o 1, and –o 2, *minmax* output minimum and maximum values, only minimum values, and only maximum values, respectively.

OPTIONS

–l	*L*	length of vector	[1]
–n	*N*	order of vector	[L-1]
–b	*B*	find n-best values	[1]
–o	*O*	output format	[0]
		$O = 0$ minimum and maximum	
		$O = 1$ minimum	
		$O = 2$ maximum	
–d		output data number	[FALSE]

EXAMPLE

If, for example, the input data in *data.f* in float format is given as

$$1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 10$$

then the output of the following command

```
minmax data.f -l 6 > data.m
```

is written to *data.m* as

$$1, 5, 6, 10$$

Also, if the following command is applied

```
minmax -n 2 -d data.f
```

then the result will be

```
1:0
2:2
3:0
5:2
6:0
8:2
9:0,1
10:2
```

NAME

mlpg – obtains parameter sequence from PDF sequence(23)

SYNOPSIS

mlpg [**–l** L] [**–m** M] [**–d** (fn | d_0 [d_1 ...])] [**–r** N_R W_1 [W_2]]
[**–i** I] [**–s** S] [***infile***]

DESCRIPTION

mlpg calculates the maximum likelihood parameters from the means and diagonal covariances of Gaussian distributions from *infile* (or standard input), and sends the result to standard output. The input format is

$$\ldots, \mu_t(0), \ldots, \mu_t(M), \mu_t^{(1)}(0), \ldots, \mu_t^{(1)}(M), \ldots, \mu_t^{(N)}(M),$$
$$\sigma_t^2(0), \ldots, \sigma_t^2(M), {\sigma^{(1)}}_t^2(0), \ldots, {\sigma^{(1)}}_t^2(M), \ldots, {\sigma^{(N)}}_t^2(M), \ldots$$

Input and output data are in float format.

The speech parameter vector $\boldsymbol{o}_t$ for every frame t is composed of the static feature vector $\boldsymbol{c}_t$, where

$$\boldsymbol{c}_t = [c_t(0), c_t(1), \ldots, c_t(M)]^\top$$

and the dynamic feature vector $\Delta^{(1)}\boldsymbol{c}_t, \ldots, \Delta^{(N)}\boldsymbol{c}_t$. Thus, the speech parameter vector can be expressed as:

$$\boldsymbol{o}_t = [\boldsymbol{c}_t', \Delta^{(1)}\boldsymbol{c}_t', \ldots, \Delta^{(N)}\boldsymbol{c}_t']^\top.$$

The dynamic feature vector $\Delta^{(n)}\boldsymbol{c}_t$ is obtained from the static feature vector as follows.

$$\Delta^{(n)}\boldsymbol{c}_t = \sum_{\tau=-L^{(n)}}^{L^{(n)}} w^{(n)}(\tau)\boldsymbol{c}_{t+\tau}$$

where n represents the order of dynamic feature vector. (e.g. $n = 2$ for Δ^2) The mlpg command reads the probability density functions sequence

$$((\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), (\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2), \ldots, (\boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T)),$$

where

$$\boldsymbol{\mu}_t = \left[\boldsymbol{\mu}_t'^{(0)}, \boldsymbol{\mu}_t'^{(1)}, \ldots, \boldsymbol{\mu}_t'^{(N)}\right]^\top$$
$$\boldsymbol{\Sigma}_t = \mathrm{diag}\left[\boldsymbol{\Sigma}_t^{(0)}, \boldsymbol{\Sigma}_t^{(1)}, \ldots, \boldsymbol{\Sigma}_t^{(1)}\right]$$

and evaluates the maximum likelihood parameter sequence $(\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T)$. The output is the static feature vector sequence $\boldsymbol{c}_t = (\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_T)$. In the example above, $\boldsymbol{\mu}^{(0)}, \boldsymbol{\Sigma}^{(0)}$ represent the static feature vector mean and covariance matrix, respectively, and $\boldsymbol{\mu}^{(n)}, \boldsymbol{\Sigma}^{(n)}$ represent the n-th order dynamic feature vector mean and covariance matrix, respectively.

OPTIONS

–m M order of vector [25]

–l L length of vector [$M + 1$]

–d ($fn \mid d_0\ [d_1\ \ldots]$) fn is the file name of the parameters $w^{(n)}(\tau)$ used when evaluating the dynamic feature vector. It is assumed that the number of coefficients to the left and to the right have the same length. If this is not true, then zeros are added to the short side. For example, if the coefficients are [N/A]

$$w(-1), w(0), w(1), w(2), w(3)$$

then zeros are added to the left as follows.

$$0, 0, w(-1), w(0), w(1), w(2), w(3)$$

Instead of entering the filename fn, the coefficients(which compose the file fn) can be directly input in the command line. When the order of the dynamic feature vector is higher than one, the sets of coefficients can be input one after the other as shown on the last example below. This option cannot be used with the –r option.

–r $N_R\ W_1\ [W_2]$ This option is used when N_R-th order dynamic parameters are used and the weighting coefficients $w^{(n)}(\tau)$ are evaluated by regression. N_R can be set to 1 or 2. The variables W_1 and W_2 represent the widths of the first and second order regression coefficients, respectively. The first order regression coefficients for $\Delta \boldsymbol{c}_t$ at frame t are evaluated as follows. [N/A]

$$\Delta \boldsymbol{c}_t = \frac{\sum_{\tau=-W_1}^{W_1} \tau \boldsymbol{c}_{t+\tau}}{\sum_{\tau=-W_1}^{W_1} \tau^2}$$

For the second order regression coefficients, $a_2 = \sum_{\tau=-W_2}^{W_2} \tau^4$, $a_1 = \sum_{\tau=-W_2}^{W_2} \tau^2$, $a_0 = \sum_{\tau=-W_2}^{W_2} 1$ and

$$\Delta^2 \boldsymbol{c}_t = \frac{\sum_{\tau=-W_2}^{W_2} (a_0 \tau^2 - a_1) \boldsymbol{c}_{t+\tau}}{2(a_2 a_0 - a_1^2)}$$

This option can not be used with the –d option.

–i *I* type of input PDFs [0]

$$\begin{array}{lll} I = 0 & \boldsymbol{\mu}, & \boldsymbol{\Sigma} \\ I = 1 & \boldsymbol{\mu}, & \boldsymbol{\Sigma}^{-1} \\ I = 2 & \boldsymbol{\mu}\boldsymbol{\Sigma}^{-1}, & \boldsymbol{\Sigma}^{-1} \end{array}$$

–s *S* range of influenced frames [30]

EXAMPLE

In the example below, the number of parameters is 15, the width of the window for first or second order dynamic feature evaluation is 1, and the parameter sequence is evaluated from the probability density function:

```
mlpg -m 15 -r 2 1 1 data.pdf > data.par
```

or

```
echo "-0.5 0 0.5" | x2x +af > delta
echo "0.25 -0.5 0.25" | x2x +af > accel
mlpg -m 15 -d delta -d accel data.pdf > data.par
```

NAME

mlsacheck – check stability of MLSA filter

SYNOPSIS

mlsacheck [–m *M* **] [–a** *A* **] [–r] [–R] [–P** *Pa* **] [** ***infile*** **]**

DESCRIPTION

mlsacheck tests the stability of the Mel Log Spectral Approximation (MLSA) digital filter of the mel-cepstrum coefficients in *infile* (or standard input), sending the result to standard output.

Both input and output are in float format.

As described in mlsadf, the transfer function $H(z)$ is expressed as

$$\begin{aligned} H(z) &= \exp \sum_{m=0}^{M} b(m)\Phi_m(z) \\ &= K \cdot D(z) \end{aligned}$$

where

$$\Phi_m(z) = \begin{cases} 1, & m = 0 \\ \dfrac{(1-\alpha^2)z^{-1}}{1-\alpha z^{-1}}\tilde{z}^{-(m-1)}, & m \geq 1 \end{cases}$$

and

$$\begin{aligned} \tilde{z}^{-1} &= \frac{z^{-1}-\alpha}{1-\alpha z^{-1}}, \\ K &= \exp b(0), \\ D(z) &= \exp \sum_{m=1}^{M} b(m)\Phi_m(z). \end{aligned}$$

To construct the exponential transfer function $H(z)$, Padé approximation is used to approximate complex exponential function $\exp\ w$ by a following rational function:

$$\exp\ w \simeq R_L(w) = \frac{1 + \sum_{l=1}^{L} A_{L,l} w^l}{1 + \sum_{l=1}^{L} A_{L,l}(-w)^l}$$

Then $D(z)$ is approximated by

$$D(z) = \exp(F(z)) \simeq R_L(F(z))$$

where

$$F(z) = \sum_{m=0}^{M} b(m)\tilde{z}^{-m}.$$

The stability of the MLSA synthesis filter is related to the accuracy of the approximation. When $|F(e^{j\omega})| < r = 4.5$ and $L = 4$ for $R_L(w)$, the log approximation error does not exceed 0.24 dB. The corresponding synthesis filter $R_L(F(z)) \simeq \exp(F(z)) = D(z)$ is stable when $|F(e^{j\omega})| < r_{max} = 6.2$. Also, the log approximation error does not exceed 0.2735 dB when $r = 6.0$ and $L = 5$. The corresponding synthesis filter is stable when $r_{max} = 7.65$.

In spite of whether specifying –c option or not, *mlsacheck* tests the stability and sends an ASCII report of the number of unstable frame to standard error. When specifying –c option, *mlsacheck* modifies the filter coefficients if unstable frame is found. When specifying –r option, the stable condition can be selected as follows: When '–r 0', *mlsacheck* keeps the log approximation not exceeding 0.24 dB ($Pa = 4$) or 0.2735 dB ($Pa = 5$), where *Pa* is the order of Padé approximation. When '–r 1', *mlsacheck* keeps the MLSA filter stable although the accuracy of log approximation is lost.

OPTIONS

–m	*M*	order of mel-cepstrum	[25]
–a	*A*	all-pass constant α	[0.35]
–l	*L*	FFT length	[256]
–c		modify MLSA filter coefficients of unstable frames	[N/A]
–r	*R*	stable condition for MLSA filter	[0]
		$R = 0$ keep log approximation error not exceeding 0.24 dB ($Pa = 4$) or 0.2735 dB ($Pa = 5$)	
		$R = 1$ keep MLSA filter stable	
–P	*Pa*	order of the Padé approximation *Pa* should be 4 or 5	[4]

EXAMPLE

In the following example, 25-th order mel-cepstrum coefficients are read from *data.mcep* in float format, then the stability of MLSA filter is checked, and the results are written to *data.mlsachk*.

```
mlsacheck -m 25 -c data.mcep > data.mlsachk
```

SEE ALSO

mcep, amcep, poledf, zerodf, ltcdf, lmadf, glsadf, mglsadf

NAME

mlsadf – MLSA digital filter for speech synthesis(19; 20; 12)

SYNOPSIS

mlsadf **[–m** *M* **] [–a** *A* **] [–p** *P* **] [–i** *I* **] [–b] [–P** *Pa* **] [–v] [–t] [–k]**
***mcfile* [*infile*]**

DESCRIPTION

mlsadf derives a Mel Log Spectral Approximation digital filter from mel-cepstral coefficients $c_\alpha(0), c_\alpha(1), \ldots, c_\alpha(M)$ in *mcfile* and uses it to filter an excitation sequence from *infile* (or standard input) and synthesize speech data, sending the result to standard output.

Input and output data are in float format.

The exponential transfer function $H(z)$ related to the MLSA synthesis filter is obtained from the M-th order mel-cepstral coefficients $c_\alpha(m)$ as follows.

$$H(z) = \exp \sum_{m=0}^{M} c_\alpha(m) \tilde{z}^{-m}$$

where

$$\tilde{z}^{-1} = \frac{z^{-1} - \alpha}{1 - \alpha z^{-1}}.$$

The highly accurate approximation method of the above transfer function is explained below. First, the transfer function $H(z)$ is expressed as

$$\begin{aligned} H(z) &= \exp \sum_{m=0}^{M} b(m) \Phi_m(z) \\ &= K \cdot D(z) \end{aligned}$$

where,

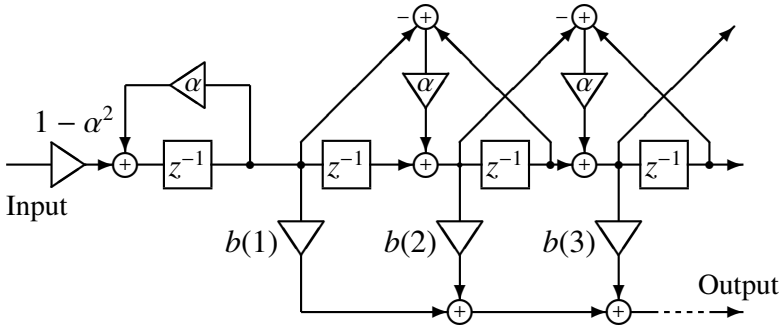
$$\Phi_m(z) = \begin{cases} 1, & m = 0 \\ \dfrac{(1-\alpha^2)z^{-1}}{1 - \alpha z^{-1}} \tilde{z}^{-(m-1)}, & m \geq 1 \end{cases}$$

and

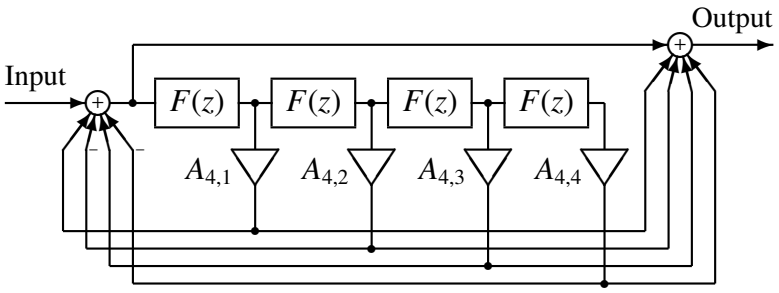
$$K = \exp b(0)$$

$$D(z) = \exp \sum_{m=1}^{M} b(m) \Phi_m(z)$$

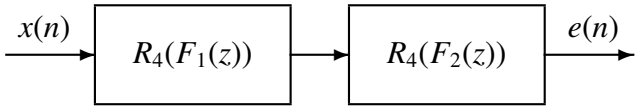
Therefore, the coefficients $b(m)$ can be obtained through a linear transformation of $c_\alpha(m)$ (refer to mc2b and b2mc).



(a) Basic filter $F(z)$



(b) $R_L(F(z)) \simeq D(z) \quad L = 4$



(c) Two-stage cascade structure
$R_4(F_1(z)) \cdot R_4(F_2(z)) \simeq D(z)$

Figure 1: Realization of exponential transfer function $1/D(z)$

The filter $D(z)$ can be constructed as shown in figure 1(b), where basic filter (figure 1(a)) is the following IIR filter.

$$F(z) = \sum_{m=1}^{M} b(m)\Phi_m(z)$$

If we want to improve the accuracy of the approximation, we can decompose the basic filter as shown in figure 1(c),

$$F(z) = F_1(z) + F_2(z)$$

where

$$F_1(z) = b(1)z^{-1}$$
$$F_2(z) = \sum_{m=2}^{M} b(m)\Phi_m(z)$$

Also, the coefficients $A_{4,l}$ in figure 1(b) have same value as the LMA filter (refer to lmadf).

OPTIONS

–m	*M*	order of mel-cepstrum	[25]
–a	*A*	all-pass constant α	[0.35]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–b		output filter coefficient $b(m)$ (coefficients which are linear transformed from mel-cepstrum)	[FALSE]
–P	*Pa*	order of the Padé approximation *Pa* should be 4 or 5	[4]
–k		filtering without gain	[FALSE]
–v		inverse filter	[FALSE]
–t		transpose filter	[FALSE]

EXAMPLE

In the following example, the excitation is constructed from pitch data read in float format from *data.pitch*, passed through an MLSA filter built from the mel-cepstrum in *data.mcep*, and the synthesized speech is written to *data.syn*:

```
excite < data.pitch | mlsadf data.mcep > data.syn
```

SEE ALSO

mcep, amcep, poledf, zerodf, ltcdf, lmadf, glsadf, mglsadf

NAME

msvq – multi stage vector quantization

SYNOPSIS

msvq **[–l** *L* **] [–n** *N* **][–s** *S* ***cbfile*** **] [–q] [** ***infile*** **]**

DESCRIPTION

msvq encodes the data from *infile* (or standard input) using multi-stage vector quantization with codebooks specified by multiple –s options, sending the result to standard output.

Input data is in float format and output data is in int format.

OPTIONS

–l	*L*	length of vector	[26]
–n	*N*	order of vector	[$L-1$]
–s	*S cbfile*	codebook	[N/A N/A]
		S codebook size	
		cbfile codebook file	
–q		output quantized vector	[FALSE]

EXAMPLE

In the example below, a two level vq is undertaken in input *data.f* file. the codebook sizes of *cbfile1* and *cbfile2* are 256 and the output is written to *data.vq*:

```
msvq -s 256 cbfile1 -s 256 cbfile2 < data.f > data.vq
```

SEE ALSO

imsvq, vq, ivq, lbg

NAME

nan – data check

SYNOPSIS

nan [infile]

DESCRIPTION

nan checks whether input data contains NaN (Not a Number) or Infinity, showing the positions where these values occurred.

EXAMPLE

This example reads input data *data.f* in float format and checks it:

```
nan data.f
```

NAME

norm0 – normalize coefficients

SYNOPSIS

norm0 [–m *M* **] [** ***infile*** **]**

DESCRIPTION

norm0 normalizes vectors from *infile* (or standard input) by dividing vector components by the zero-order component, sending the result to standard output.

For the input sequence

$$x(0), x(1), \ldots, x(M),$$

the normalized output sequence is

$$1/x(0), x(1)/x(0), \ldots, x(M)/x(0).$$

Input and output data are in float format.

OPTIONS

–m *M* order of input data [25]

EXAMPLE

Speech data is read from *data.f* in float format, the 15-th order autocorrelation coefficients are evaluated and normalized, and the results is written to *data.nacorr*:

```
frame < data.f | window | acorr -m 15 |\
norm0 -m 15 > data.nacorr
```

SEE ALSO

linear_intpl

NAME

nrand – generate normal distributed random value

SYNOPSIS

nrand **[–l** *L* **] [–s** *S* **] [–m** *M* **] [–v** *V* **] [–d** *D* **]**

DESCRIPTION

nrand generates a sequence of normally-distributed random values, sending the result to standard output.

Output data is in float format.

OPTIONS

–l	*L*	output length In the case $L \leq 0$ then random values will be generated indefinitely.	[256]
–s	*S*	seed for nrand	[1]
–m	*M*	mean of normal distribution	[0.0]
–v	*V*	variance of normal distribution	[1.0]
–d	*D*	standard deviation of normal distribution	[1.0]

EXAMPLE

Normal distributed random values of length 100 are generated and written to *data.rnd*:

```
nrand -l 100 -s 3 > data.rnd
```

NAME

par2lpc – transform PARCOR to LPC

SYNOPSIS

par2lpc [–m *M* **] [** ***infile*** **]**

DESCRIPTION

par2lpc calculates linear prediction (LPC) coefficients from M-th order PARCOR coefficients from *infile* (or standard input), sending the result to standard output.

The PARCOR input format is

$$K, k(1), \ldots, k(M),$$

and the LPC output format is

$$K, a(1), \ldots, a(M).$$

Input and output data are in float format.

The Durbin algorithm is used for the transformation of PARCOR coefficients into linear prediction coefficients as follows;

$$\begin{aligned} a^{(m)}(m) &= k(m) \\ a^{(m)}(i) &= a^{(m-1)}(i) + k(m)a^{(m-1)}(m-i), \qquad 1 \le i \le m \end{aligned}$$

where $m = 1, 2, \ldots, p$. The initial condition is

$$a^{(M)}(m) = a(m), \qquad 1 \le m \le M.$$

OPTIONS

–m *M* order of LPC [25]

EXAMPLE

PARCOR coefficients are read in float format from *data.rc* and converted into the corresponding linear prediction coefficients. The output is written to *data.lpc*:

```
par2lpc < data.rc > data.lpc
```

SEE ALSO

acorr, levdur, lpc, lpc2par

NAME

pca – principal component analysis

SYNOPSIS

pca [**–l** *L*] [**–n** *N*] [**–i** *I*] [**–e** *e*] [**–v**] [**–V** *fn*] [***infile***]

DESCRIPTION

pca applies principal component analysis in the data from *infile* (or standard input) using the Jacobi method, and sends the result to standard output. *pca* can also calculate contribution ratio with the eigen values.

In *infile*, the input training data set consists of *L*-dimension vectors of the form:

$$\boldsymbol{x}(0), \boldsymbol{x}(1), \boldsymbol{x}(2), \boldsymbol{x}(3), \cdots \quad \text{where} \quad \boldsymbol{x}(i) = (x_i(1), x_i(2), \cdots, x_i(L))$$

Input and output data are in float format.

OPTIONS

–l	*L*	dimension of vector	[3]
–n	*N*	number of output principal components	[2]
–i	*I*	limit of iterations of the Jacobi method	[10000]
–e	*e*	threshold of convergence of the Jacobi method	[0.000001]
–v		output eigenvectors and mean vector of the training data	[FALSE]
–V	*fn*	output eigenvalues and contribution rate (output filename = fn)	[FALSE]

EXAMPLE

In the example below, the eigenvectors and the eigenvalues are calculated from *data.f* which contains three-dimensional training vectors. The mean vectors and eigenvectors are sent to *pca.dat*, and the eigenvalues are sent to *eigen.dat*.

```
pca data.f -n 2 -l 3 -v -V eigen.dat > pca.dat
```

Note that in the *pca.dat*, the mean vector is written in front of the eigenvectors. In the *eigen.dat*, the eigenvalues and their contribution ratio are bound by the same principal component and ordered according to the magnitude of the eigen values.

SEE ALSO

pcas

NAME

pcas – calculate principal component scores

SYNOPSIS

pcas **[–l** *L* **] [–n** *N*] ***pcafile*** **[** ***infile*** **]**

DESCRIPTION

pcas calculates principal component scores from the data in *infile* (or standard input) , and sends the result to standard output.

The input data set must be composed of an L-dimension, mean vector $\boldsymbol{m}$ and eigenvectors $\boldsymbol{e}(i)$ as in:

$$\boldsymbol{m}, \boldsymbol{e}(0), \boldsymbol{e}(1), \boldsymbol{e}(2), \cdots$$

$$\textit{where} \quad \boldsymbol{m} = (m(1), m(2), \cdots, m(L)) \quad \textit{and} \quad \boldsymbol{e}(i) = (e_i(1), e_i(2), \cdots, e_i(L))$$

Input and output data are in float format.

OPTIONS

–l	*L*	dimensionality of vector	[3]
–n	*N*	output number of principal components	[2]

EXAMPLE

In the example below, the principal component scores are calculated from *test.dat* and sent to *score.dat*. Here, *pca.dat* is a file that contains the mean and eigenvectors.

```
pcas pca.dat -l 3 -n 2 < test.dat > score.dat
```

In *pca.dat*, the mean vector must be written before the eigenvectors.

SEE ALSO

pca

NAME

phase – transform real sequence to phase

SYNOPSIS

phase **[–l** *L* **]** **[–p** ***pfile*** **]** **[–z** ***zfile*** **]** **[–m** *M* **]** **[–n** *N* **]** **[** ***infile*** **]**

DESCRIPTION

phase calculates the phase of the spectrum of a real sequence from *infile* (or standard input), and sends the result to standard output. Assume that the input sequence is

$$x(0), x(1), \ldots, x(L-1)$$

and the FFT is

$$\begin{aligned} X_k &= X(e^{j\omega})\Big|_{\omega = \frac{2\pi k}{L}} \\ &= \sum_{m=0}^{L-1} x(m)e^{-j\omega m}\Big|_{\omega = \frac{2\pi k}{L}} \quad , \qquad k = 0, 1, \ldots, L-1 \end{aligned}$$

Then the output is given by

$$Y_k = \arg X_k, \qquad k = 0, 1, \ldots, L/2$$

In this case the phase is written in continuous form. The output data angular frequency varies from $0 \sim \pi$. Input and output data are in float format.

If the **–p, –z** options are assigned then the phase of the corresponding filter related to the assigned coefficients is calculated [1].

OPTIONS

–l	*L*	frame length power of 2	[256]
–p	*pfile*	numerator coefficients file The *pfile* should follow this structure in float format: $K, a(1), \ldots, a(M)$	[NULL]
–z	*zfile*	denominator coefficients file The *zfile* should follow this structure in float format: $b(0), b(1), \ldots, b(N)$ The contents of *pfile* and *zfile* should be in a similar form to that used in the *dfs* command. When only the **–p** option is assigned then the denominator is made equal to 1. When only the **–z** option is assigned, the numerator and the gain K are both set to 1. If neither **–p** nor **–z** are assigned, data is read from the standard input.	[NULL]

[1] In this case the phase is not evaluated from the filter impulse response, but from the difference between the numerator and denominator phases

–m	*M*	order of polynomial denominator If the number of input data values is less $M + 1$, then M is set to the number of input data values -1. On the other hand, There is no need to assign a values to M if one doesn't want the data to be analyzed is blocks of $M + 1$ size.	$[L - 1]$
–n	*N*	order of polynomial numerator Likewise the –m option, if the number of input data values is less then $N + 1$, then N is set to the number of input data values -1. On the other hand, There is no need to assign a values to N if one doesn't want the data to be analyzed is blocks of $N + 1$ size.	$[L - 1]$
–u		unwrapping	[TRUE]

EXAMPLE

In the example below, the phase characteristic of a digital filter with coefficients assigned by the files *data.p, data.z* in float format can be displayed by:

```
phase -p data.p -z data.z | fdrw | xgr
```

If the filter defined by *data.p*, *data.z* is stable then the following command will give a similar result:

```
impulse | dfs -p data.p -z data.z | phase | fdrw | xgr
```

SEE ALSO

spec, fft, fftr, dfs

BUGS

If the sample interval between FFT points is large (the value assigned by the –l option is small), or if the phase characteristic includes steep angles (i.e. zeros and/or poles are close to the unit circle in the z domain), it might happen that the phase is not properly drawn in continuous form.

NAME

pitch – pitch extraction

SYNOPSIS

pitch [**–a** *A*] [**–s** *S*] [**–p** *P*] [**–T** *T*] [**–t** *t*] [**–L** *Lo*] [**–H** *Hi*] [**–o** *O*][***infile***]

DESCRIPTION

pitch extracts the pitch values from *infile* (or standard input), sending the result to standard output. The RAPT (24) and SWIPE' (25) algorithm are adopted for pitch extraction. They can be specified by –a option. The output format (pitch, F0 or log(F0)) can be specified by –o option.

Both input and output files are in float format.

OPTIONS

–a	*A*	algorithm used for extraction of pitch	[0]
		A = 0 RAPT	
		A = 1 SWIPE'	
–s	*S*	sampling frequency (kHz)	[16.0]
–p	*P*	frame shift	[80]
–T	*T*	voiced/unvoiced threshold (used only for RAPT algorithm)	[0.0]
–t	*t*	voiced/unvoiced threshold (used only for SWIPE' algorithm)	[0.3]
–L	*Lo*	minimum fundamental frequency to search for (Hz)	[60.0]
–H	*Hi*	maximum fundamental frequency to search for (Hz)	[240.0]
–o	*O*	output format	[0]
		O = 0 pitch	
		O = 1 F0	
		O = 2 log(F0)	

EXAMPLE

In the example below, speech data in float format is read from *data.f* and the pitch data is extracted via SWIPE' algorithm under the condition that sampling frequency is 16kHz, the frame shift is 80 point, and the minimum and maximum fundamental frequency are 80 and 165 Hz, respectively. Then, the output is written to *data.pitch*:

```
pitch -a 1 -s 16 -p 80 -L 80 -H 165 data.f > data.pitch
```

SEE ALSO

excite

NAME

poledf – all pole digital filter for speech synthesis

SYNOPSIS

poledf **[–m** *M* **] [–p** *P* **] [–i** *I* **] [–t] [–k]** ***afile*** **[** ***infile*** **]**

DESCRIPTION

poledf derives an all pole standard form digital filter from the linear prediction (LPC) coefficients $K, a(1), \ldots, a(M)$ in *afile* and uses it to filter an excitation sequence from *infile* (or standard input) to synthesize speech data, sending the result to standard output.

Input and output data are in float format.

The transfer function $H(z)$ of an all pole standard form filter is

$$H(z) = \frac{K}{1 + \sum_{m=1}^{M} a(m) z^{-m}}$$

OPTIONS

–m	*M*	order of coefficients	[25]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–t		transpose filter	[FALSE]
–k		filtering without gain	[FALSE]

EXAMPLE

In the example below, the excitation is generated from the pitch information read from *data.pitch* in float format. It is then passed through the standard form synthesis filter built from the linear prediction coefficients file *data.lpc*, and the synthesized speech is output to *data.syn*:

```
excite < data.pitch | poledf data.lpc > data.syn
```

SEE ALSO

lpc, acorr, ltcdf, lmadf, zerodf

NAME

psgr – XY-plotter simulator for EPSF

SYNOPSIS

psgr **[–t *title*]** **[–s** *S* **]** **[–c** *C* **]** **[–x** *X* **]** **[–y** *Y* **]** **[–p P]** **[–r** *R* **]** **[–b]**
[–T *T* **]** **[–B** *B* **]** **[–L** *L* **]** **[–R** *R* **]** **[–P]** **[** ***infile*** **]**

DESCRIPTION

psgr converts FP5301 plotter commands from *infile* (or standard input) to PostScript (EPSF or PS), sending the result to standard output.

OPTIONS

–t	*title*	title of figure	[NULL]
–s	*S*	shrink	[1.0]
–c	*C*	number of copy	[1]
–x	*X*	x offset (mm)	[0]
–y	*Y*	y offset (mm)	[0]
–p	*P*	paper (Letter, A0, A1, A2, A3, A4, A5, B0, B1, B2, B3, B4, B5)	[FALSE]
–l		landscape	[FALSE]
–r	*R*	resolution (dpi)	[600]
–b		bold font mode	[FALSE]
–T	*T*	top margin (mm)	[0]
–B	*B*	bottom margin (mm)	[0]
–L	*L*	left margin (mm)	[0]
–R	*R*	right margin (mm)	[0]
–P		output Postscript code	[FALSE]

EXAMPLE

This example/command creates the figure file *data.fig* and sends it to a printer.

```
fig data.fig | psgr | lpr
```

BUGS

- It may happen that a part of the Y axis label is not properly output. This problem can be solved by altering the margins.
- When the size of the figure is modified, and included in a TEXfile, it may not be displayed correctly. To solve this problem, please use TEXoptions for including pictures and adjusting sizes.

SEE ALSO

fig, fdrw, xgr

NAME

ramp – generate ramp sequence

SYNOPSIS

ramp [**–l** *L*] [**–n** *N*] [**–s** *S*] [**–e** *E*] [**–t** *T*]

DESCRIPTION

ramp generates ramp sequences of length L, sending the result to standard output. The output is as follows.

$$\underbrace{S, S+T, S+2T, \ldots, S+(L-1)T}_{L}$$

Output format is in float format. In the case the last value is assigned the generated sequence is,

$$\underbrace{S, S+T, S+2T, \ldots, E}_{(E-S)/T}$$

If the –l , –e and –n options are used at the same time, only the last option is taken into account.

OPTIONS

–l	*L*	length of ramp sequence If $L \leq 0$ ramp values will be generated indefinitely.	[256]
–n	*N*	order of ramp sequence	[L-1]
–s	*S*	start value	[0]
–e	*E*	end value	[N/A]
–t	*T*	step size	[1]

EXAMPLE

The command below outputs the following sequence:

$$y(n) = \exp(-n)$$

```
ramp | sopr -m -1 -E | dmp +f
```

SEE ALSO

impulse, step, train, sin

NAME

raw2wav – raw to wav (RIFF)

SYNOPSIS

raw2wav [–swab][–s *S* **] [–d** *D* **] [–n] [–N] [+*type*] [*infile*]**

DESCRIPTION

raw2wav converts file format from raw to wav.

OPTIONS

–swab		change endian	[FALSE]
–s	*S*	sampling frequency	[16000]
–d	*D*	destination directory	[N/A]
–n		normalization with the maximum value if max >= 32767	[FALSE]
–N		normalization	[FALSE]
+*type*1		input data type	[s]
+*type*2		output data type	[s]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)

EXAMPLE

In the following command, the file *file.raw*, in raw format is converted to the wav format file *data.wav* and saved to the same directory of the input file. Here, the –s option specifies the sampling frequency of the input file. One can also specify a different directory for the output file by using the –d option.

```
raw2wav -s 8000 data.raw
```

SEE ALSO

swab, minmax

NAME

reverse – reverse the order of data in each block

SYNOPSIS

reverse **[–l** *L* **] [–n** *N* **] [** ***infile*** **]**

DESCRIPTION

reverse reverses the order of data within *L*-length blocks of input data from *infile* (or standard input), and sends the result to standard output. The default value for *L* is the entire file. If *L* is given but the file length is not a multiple of *L*, leftover values are discarded as shown in the example below.

OPTIONS

–l	*L*	length of block	[EOF]
–n	*N*	order of block	[EOF-1]

EXAMPLE

Let's assume that the following data is read from *data.in* file in float format.

$$\underbrace{0.0,\ 1.0,\ 2.0},\ \underbrace{3.0,\ 4.0,\ 5.0},\ \underbrace{6.0,\ 7.0,\ 8.0},\ 9.0$$

The command

```
reverse -l 3 data.in > data.out
```

will write the following output to *data.out*.

$$\underbrace{2.0,\ 1.0,\ 0.0},\ \underbrace{5.0,\ 4.0,\ 3.0},\ \underbrace{8.0,\ 7.0,\ 6.0}$$

NAME

rmse – calculation of root mean squared error

SYNOPSIS

rmse [–l *L* **]** ***file1*** **[** ***infile*** **]**

DESCRIPTION

rmse calculates RMSE (Root Mean Square Error) of input data sequences from *infile* (or standard input) and *file1*, sending the results to standard output.

If two files are given, the L-length time series

$$\underbrace{x_1(0), x_1(1), \ldots, x_1(L-1)}, \underbrace{x_2(0), x_2(1), \ldots}$$

and

$$\underbrace{y_1(0), y_1(1), \ldots, y_1(L-1)}, \underbrace{y_2(0), y_2(1), \ldots}$$

are read, and the RMSE of these two series are calculated and output. The RMSE is given by:

$$\mathrm{RMSE}_j = \sqrt{\sum_{m=0}^{L-1} (x_j(m) - y_j(m))^2 / L}$$

Input and output data are in float format.

OPTIONS

–l *L* data length to calculate RMSE. [0]
If $L = 0$, RMSE of whole input data is output.

EXAMPLE

This example calculates the RMSE of input data files *data.f1* and *data.f2*, and outputs its maximum and minimum values:

```
rmse -l 26 data.f1 data.f2 | minmax | dmp +f
```

SEE ALSO

histogram, minmax

NAME

root_pol – calculate roots of a polynomial equation

SYNOPSIS

root_pol **[–m** *M* **] [–n** *N* **] [–e** *E* **] [–i] [–s] [–r] [** ***infile*** **]**

DESCRIPTION

root_pol finds root values of a polynomial equation from *infile* (or standard input), and sends the result to standard output.

For a given input file, the coefficients

$$a_0, a_1, \ldots, a_n$$

of an n-th order polynomial equation of the form:

$$P(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n,$$

are first read from the file and then the roots of the polynomial are calculated by the Durand-Kerner-Aberth method.

If roots of $P(x)$ are z_i, the result is sent to standard output in complex form as

$$\begin{array}{cc} \mathrm{Re}[z_0], & \mathrm{Im}[z_0] \\ \mathrm{Re}[z_1], & \mathrm{Im}[z_1] \\ \vdots & \\ \mathrm{Re}[z_{n-1}], & \mathrm{Im}[z_{n-1}] \end{array}$$

or polar form as

$$\begin{array}{cc} |z_0|, & \arg[z_0] \\ |z_1|, & \arg[z_1] \\ \vdots & \\ |z_{n-1}|, & \arg[z_{n-1}] \end{array}$$

Both input and output data are in float format.

OPTIONS

–m	*M*	order of polynomial equation	[32]
–n	*N*	maximum iteration to search roots	[1000]
–e	*E*	error margin for roots ε	[10^{-14}]
–i		set $a_0 = 1$	[FALSE]
–s		reverse order of coefficients	[FALSE]
–r		output results in polar form	[complex form]

EXAMPLE

The following command calculates roots of the polynomial equation specified in the file *data.z*. The results are output in polar form:

```
root_pol -r < data.z | x2x +a 2
```

NAME

sin – generate sinusoidal sequence

SYNOPSIS

sin **[–l** *L* **] [–p** *P* **] [–m** *M* **]**

DESCRIPTION

sin generates a discrete sin wave sequence of period P, length L and magnitude M of the form,

$$x(n) = M \cdot \sin\left(\frac{2\pi}{P} \cdot n\right),$$

and sends the result to standard output.

Both input and output data are in float format.

OPTIONS

–l	*L*	length	[256]
		If $L \leq 0$, sin values will be generated indefinitely.	
–p	*P*	period	[10.0]
–m	*M*	magnitude	[1.0]

EXAMPLE

In the following example, a sin wave sequence is parsed through a Blackman window and the results are displayed the results on the screen:

```
sin -p 12.3 | window | fdrw | xgr
```

SEE ALSO

impulse, step, train, ramp

NAME

smcep – mel-cepstral analysis using 2nd order all-pass filter(15; 16)

SYNOPSIS

smcep **[–a** *A* **]** **[–t** *t* **]** **[–T** *T* **]** **[–s** *s* **]** **[–m** *M* **]** **[–l** *L* **]** **[–q** *Q* **]**
[–i *I* **]** **[–j** *J* **]** **[–d** *D* **]** **[–e** *e* **]** **[–E** *E* **]** **[–f** *F* **]** **[** ***infile*** **]**

DESCRIPTION

smcep calculates the mel-cepstral coefficients from L-length framed windowed input data from *infile* (or standard input), sending the result to standard output. The analysis uses a second-order all-pass function raised to the $1/2$ power $1/2$:

$$A(z) = \left(\frac{z^{-2} - 2\alpha \cos\theta z^{-1} + \alpha^2}{1 - 2\alpha \cos\theta z^{-1} + \alpha^2 z^{-2}} \right)^{\frac{1}{2}},$$

$$\tilde{z}^{-1} = \frac{z^{-1} - \alpha}{1 - \alpha z^{-1}}.$$

Input and output data are in float format.

In the mel-cepstral analysis using a 2nd-order all pass function, the speech spectrum is modeled as m-th order cepstral coefficients $c(m)$ as follows.

$$H(z) = \exp \sum_{m=0}^{M} c(m)\, B_m(e^{j\omega})$$

where

$$\mathrm{Re}\left[B_m(e^{j\omega}) \right] = \frac{A^m(e^{j\omega}) + A^m(e^{-j\omega})}{2}$$

The Newton-Raphson method is applied to calculate the mel-cepstral coefficients through the minimization of the cost function.

OPTIONS

–a	A	all-pass constant α	[0.35]
–t	t	emphasized frequency $\theta * \pi$ (rad)	[0]
–T	T	emphasized frequency (Hz)	[0]
–s	s	sampling frequency (kHz)	[10]
–m	M	order of mel cepstrum	[25]
–l	L_1	frame length	[256]
–L	L_2	ifft size for making matrices	[1024]
–q	Q	input data style	[0]

$Q = 0$ windowed data sequence
$Q = 1$ $20 \times \log |f(w)|$
$Q = 2$ $\ln |f(w)|$
$Q = 3$ $|f(w)|$
$Q = 4$ $|f(w)|^2$

Usually, the options below do not need to be assigned.

–i	I	minimum iteration of Newton-Raphson method	[2]
–j	J	maximum iteration of Newton-Raphson method	[30]
–d	D	end condition of Newton-Raphson	[0.001]
–e	e	small value added to periodogram	[0]
–E	E	floor in db calculated per frame	[N/A]
–f	F	mimimum value of the determinant of the normal matrix	[0.000001]

EXAMPLE

In the example below, speech data is read in float format from *data.f*, analyzed, and resulting mel-cepstral coefficients are written to *data.mcep*:

```
frame < data.f | window | smcep > data.mcep
```

SEE ALSO

uels, gcep, mcep, mgcep, mlsadf

NAME

snr – evaluate SNR and segmental SNR

SYNOPSIS

snr **[–l** *L* **] [–o** *O* **]** ***file1*** **[** ***infile*** **]**

DESCRIPTION

srn calculates the SNR (Signal to Noise Ratio) and the SNR_{seg} (segmental SNR) between corresponding *L*-length frames of *file1* and *infile* (or standard input), sending the result to standard output. The output format is specified by the –o option.

The SNR and SNR_{seg} are calculated through the following equations.

$$\text{SNR} = 10 \log \frac{\sum_n \{x(n)\}^2}{\sum_n \{e(n)\}^2} \quad [\text{dB}]$$

$$\text{SNR}_{\text{seg}} = \frac{1}{N_i} \sum_{i=1}^{N_i} \text{SNR}_i \quad [\text{dB}]$$

where

$$e(n) = x_1(n) - x_2(n)$$

The number of frames is represented by N_i. For signals with small amplitudes, such as consonant sounds, the segmental SNR represents a better subjective measure than the SNR.

OPTIONS

–l *L* frame length [256]
–o *O* output data format [0]

0 SNR and SNRseg
1 SNR and SNRseg in detail
2 SNR
3 SNRseg

if 0 or 1 are assigned
the output data is written in ASCII format.
if 2 or 3 are assigned
the output data is written in float format

EXAMPLE

The following command reads the input files *data.f1* and *data.f2*, evaluates the SNR and segmental SNR, and sends the results to the standard output:

```
snr data.f1 data.f2
```

SEE ALSO

histogram, average, rmse

NAME

sopr – execute scalar operations

SYNOPSIS

sopr **[–a** *A* **] [–s** *S* **] [–m** *M* **] [–d** *D* **] [–f** *F***] [–c C] [–magic** *magic* **]**
[–MAGIC *MAGIC* **] [–ABS] [–INV] [–P] [–R] [–SQRT] [–LN]**
[–LOG2] [–LOG10] [–EXP] [–POW2] [–POW10] [–FIX] [–UNIT]
[–CLIP] [–SIN] [–COS] [–TAN] [–ATAN] [–r m*n* **] [–w m***n* **] [** ***infile*** **]**

DESCRIPTION

sopr performs a sequence of scalar operations on float data from *infile* (or standard input), sending the float output data to standard output.

The sequence of operations is specified by command line options and is performed in the given order.

OPTIONS

–a	*A*	addition $y = x + A$	[FALSE]
–s	*S*	subtraction $y = x - S$	[FALSE]
–m	*M*	multiplication $y = x * M$	[FALSE]
–d	*D*	division $y = x/D$	[FALSE]
–f	*F*	flooring $y = F$ if $x < F$	[FALSE]
–c	*C*	ceiling $y = C$ if $x > C$	[FALSE]
–magic	*magic*	remove magic number	[FALSE]
–MAGIC	*MAGIC*	replace magic number by *MAGIC* if -magic option is not given, return error. if -magic or -MAGIC option is given multiple times, also return error.	[FALSE]

If the argument of the above operation option given is "*dB*", "*cent*" or "*octave*" then the values $20/\log_e 10$, $1200/\log_e 2$ or $1/\log_e 2$ are assigned, respectively. Likewise, if "*pi*" is written after the operation option, then its value will be used. Expression such as "*ln2*", "*exp10*", "*sqrt30*" can also be used as arguments.

–ABS	absolute $y = \|x\|$	[FALSE]
–INV	inverse $y = 1/x$	[FALSE]
–P	square $y = x^2$	[FALSE]
–R	square root $y = \sqrt{x}$	[FALSE]
–SQRT	square root $y = \sqrt{x}$	[FALSE]
–LN	logarithm $y = \log x$	[FALSE]
–LOG2	logarithm $y = \log_2 x$	[FALSE]
–LOG10	logarithm $y = \log_{10} x$	[FALSE]

–EXP		exponential $y = \exp x$	[FALSE]
–POW2		power of 2 $y = 2^x$	[FALSE]
–POW10		power of 10 $y = 10^x$	[FALSE]
–FIX		round $(int)x$	[FALSE]
–UNIT		unit step $u(x)$	[FALSE]
–CLIP		clipping $x * u(x)$	[FALSE]
–SIN		sin $y = \sin(x)$	[FALSE]
–COS		cos $y = \cos(x)$	[FALSE]
–TAN		tan $y = \tan(x)$	[FALSE]
–ATAN		atan $y = \text{atan}(x)$	[FALSE]
–r	m*n*	read from memory register m*n* ($n = 0..9$)	
–w	m*n*	write from memory register m*n* ($n = 0..9$)	

EXAMPLE

In the following example, a ramp function $(0, 1, 2, \ldots)$ is multiplied by 2 $(0, 2, 4, \ldots)$ and then 1 is added $(1, 3, 5, \ldots)$:

```
ramp | sopr -m 2 -a 1 | dmp +f
```

The output file *data.avrg* contains the mean taken from data in files *data.f1* and *data.f2* read in float format:

```
vopr -a data.f1 data.f2 | sopr -d 2 > data.avrg
```

In the following examples, data is read in float format from *data.f*, and the results in dB are written to the output file:

```
sopr data.f -LN -m dB | dmp +f

sopr data.f -LOG10 -m 20 | dmp +f
```

In the following, the results in cent are written to the output file:

```
sopr data.f -LN -m cent | dmp +f

sopr data.f -LOG2 -m 1200 | dmp +f
```

The following example replace the number 0 by 1.0. While the -Magic option is not given, skip any operations at the magic number.

```
sopr data.f -magic 0 -m 4.0 -INV -MAGIC 1.0 | dmp +f
```

If we want to evaluate the following equation,

$$y = (1 + 3x + 4x^2)/(1 + 2x + 5x^2)$$

then memory registers can be used as follows.

```
sopr data.f -w m0 -m 5 -a 2 -m m0 -a 1 -w m1 \
       -r m0 -m 4 -a 3 -m m0 -a 1 -d m1 | dmp +f
```

In the example above, m0 and m1 are memory registers. Registers from m0 to m9 can be used. The –w option is used to write into a memory register, while the –r option is used to read from a register.

SEE ALSO

vopr, vsum

NAME

spec – transform real sequence to log spectrum

SYNOPSIS

spec **[–l** *L* **]** **[–m** *M* **]** **[–n** *N* **]** **[–z** ***zfile*** **]** **[–p** ***pfile*** **]**
[–e *e* **]** **[–E** *E* **]** **[–o** *O* **]** **[** ***infile*** **]**

DESCRIPTION

spec computes the log spectrum magnitude of framed windowed input data from *infile* (or standard input), and sends the result to standard output.

Alternatively, given the poles (–p *pfile* option) and zeroes (–z *zfile* option) of a digital filter, *spec* computes the frequency response of that filter.

The output format is specified by the –y option.

If the input sequence is given by

$$x(0), x(1), \ldots, x(L-1)$$

and the FFT algorithm is used to evaluate

$$\begin{aligned} X_k &= X(e^{j\omega})\Big|_{\omega = \frac{2\pi k}{L}} \\ &= \sum_{m=0}^{L-1} x(m)e^{-j\omega m}\Big|_{\omega = \frac{2\pi k}{L}} \quad , \qquad k = 0, 1, \ldots, L-1 \end{aligned}$$

then if the **–y** option is applied, the output will be

$$Y_k = 20 \log_{10} |X_k|, \qquad k = 0, 1, \ldots, L/2$$

The output data corresponds to angular frequencies varying from $0 \sim \pi$. Input and output data are in float format.

If the **–p** and **–z** options are assigned then the phase of the corresponding filter related to the assigned coefficients is calculated [2].

OPTIONS

–l	*L*	FFT window length *L* must be power of 2	[256]
–m	*M*	order of MA part In the case where the number of input data values is less then $M + 1$, then M is made equal to the number of input data values -1. You don't need to assign a value to M in case there is no need to for the data to be analyzed in blocks of size $M + 1$.	[0]

[2] In this case the phase is not evaluated from the filter impulse response, the phase is evaluated from the difference between the numerator and denominator phases

–n	*N*	order of AR part Similarly to the –m option, in the case where the number of input data values is less then $N + 1$, then N is made equal to the number of input data values -1. You don't need to assign a value to N in case there is no need to for the data to be analyzed in blocks of size $N + 1$.	[0]
–z	*zfile*	MA coefficients filename The *zfile* should contain the following structure in float format: $b(0), b(1), \ldots, b(N)$	[NULL]
–p	*pfile*	AR coefficients filename The *pfile* should contain the following structure in float format: $K, a(1), \ldots, a(M)$	[NULL]
–e	*e*	small value for calculating log()	[0.0]
–E	*E*	floor in db calculated per frame	[N/A]
–o	*O*	output format	[0]

$O = 0$	$20 \times \log \lvert X_k \rvert$	$k = 0, 1, \ldots, L/2$
$O = 1$	$\ln \lvert X_k \rvert$	$k = 0, 1, \ldots, L/2$
$O = 2$	$\lvert X_k \rvert$	$k = 0, 1, \ldots, L/2$
$O = 3$	$\lvert X_k \rvert^2$	$k = 0, 1, \ldots, L/2$

The contents of *pfile* and *zfile* should be in a similar form to that used in the *dfs* command. When only the **–p** option is assigned, the denominator is set to 1. When only the **–z** option is assigned, the numerator and the gain K are set to 1. If neither **–p** nor **–z** are assigned, data is read from the standard input.

EXAMPLE

In the example below, a pulse train excitation is passed through digital filter and Blackman window. The log spectrum magnitude is, thus, evaluated and plotted on the screen:

```
train -p 50 | dfs -a 1 0.9 | window | spec | fdrw | xgr
```

This example evaluates the frequency response of a digital filter with coefficients specified in *data.p and data.z* in float format:

```
spec -p data.p -z data.z | fdrw | xgr
```

A similar result can be obtained with the following command, for a stable filter:

```
impulse | dfs -p data.p -z data.z | spec | fdrw | xgr
```

SEE ALSO

phase, fft, fftr, dfs

NAME

step – generate step sequence

SYNOPSIS

step [**–l** *L*] [**–n** *N*] [**–v** *V*]

DESCRIPTION

step generates a step sequence of length *L*, sending the result to standard output.

The output is in float format, as follows.

$$\underbrace{V, V, V, \ldots, V}_{L}$$

OPTIONS

–l	*L*	length In the case where $L \leq 0$, step values will be generated indefinitely.	[256]
–n	*N*	order	[255]
–v	*V*	step value	[1.0]

EXAMPLE

In the following example, the unit step sequence is passed through a digital filter and sent to the standard output:

```
step | dfs -a 1 -0.8 | dmp +f
```

SEE ALSO

impulse, train, ramp, sin

NAME

swab – swap bytes

SYNOPSIS

swab **[–S** S_1 **] [–s** S_2 **] [–E** E_1 **] [–e** E_2 **] [+***type* **] [*infile*]**

DESCRIPTION

swab changes the byte order (from big-endian to little-endian or vice versa) of the input data from *infile* (or standard input), and sends the result to standard output.

The range of input data that is changed can be restricted with the –S, –E or –s, –e options.

The +*type* option specifies the input and output data formats.

OPTIONS

–S	S_1	start address	[0]
–s	S_2	start offset number	[0]
–E	E_1	end address	[EOF]
–e	E_2	end offset number	[0]
+*type*		Input and output data format	[s]

s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)

EXAMPLE

In the example below, the byte order of the file *data.f* in float format is changed and written to *data.swab*:

```
swab +f data.f > data.swab
```

NAME

symmetrize – symmetrize the sequence of data

SYNOPSIS

symmetrize **[–l** L **] [–o** o **] [** ***infile*** **]**

DESCRIPTION

symmetrize symmetrizes the sequence of $L/2$-length of input data from *infile* (or standard input) and sends the result to standard output. The value of L must be even number. The output format is specified by the -o option. If the file length is not a multiple of $L/2$, leftover values are discarded as shown in the example below.

$$\text{Input sequence} \quad x(0),\ x(1),\ \ldots,\ x(L/2-1)$$

OPTIONS

–l	L	frame length	[256]
–o	o	output format	[0]

$o = 0$ $x(0),\ x(1),\ \ldots,\ x(L/2-1),\ x(L/2-2),\ \ldots,\ x(2),\ x(1)$
$o = 1$ $x(L/2-1),\ x(L/2-2),\ \ldots,\ x(1),\ x(0),\ x(1),\ \ldots,\ x(L/2-1)$
$o = 2$ $x(L/2-1)/2,\ x(L/2-2),\ \ldots,\ x(1),\ x(0),\ x(1),\ \ldots,\ x(L/2-1)/2$

EXAMPLE

Let's assume that the following data is read from *data.in* file in float format.

$$\underbrace{0.0,\ 1.0,\ 2.0, 3.0}, \ 4.0$$

The command

```
symmetrize -l 8 -o 1 data.in > data.out
```

will write the following output to *data.out*.

$$\underbrace{3.0,\ 2.0,\ 1.0,\ 0.0,\ 1.0,\ 2.0,\ 3.0}$$

NAME

train – generate pulse sequence

SYNOPSIS

train **[–l** *L* **] [–p** *P* **]**

DESCRIPTION

train generates a normalized pulse train sequence or a sequence with values ±1, and sends the result to standard output. Output data is in float format.

OPTIONS

–l *L* sequence length [256]

–p *P* frame period ($P \geq 1.0$) [0.0]
if $P = 0.0$ a sequence with values ±1 is generated.

–n *N* type of normalization [1]
If $x(n)$ is the impulse sequence, then:

0 no-normalization

1 normalization as $\sum_{n=0}^{L-1} x^2(n) = 1$

2 normalization as $\sum_{n=0}^{L-1} x(n) = 1$

EXAMPLE

The following example displays the spectrum of the signal obtained from passing a train pulse sequence through a digital filter:

```
train | dfs -b 1 0.9 | window | spec | fdrw | xgr
```

SEE ALSO

impulse, sin, step, ramp

NAME

transpose – transpose a matrix

SYNOPSIS

transpose **[–m** *m* **] [–n** *n* **] [** ***infile*** **]**

DESCRIPTION

transpose assumes the input data from *infile* (or standard input) as $m \times n$ matrix and transposes the matrix to $n \times m$ matrix. Then, sends the result to standard output. You have to define the number of rows and columns and if the file length is not a multiple of $m \times n$, leftover values are discarded as shown in the example below.

Input sequence

$$\begin{array}{ccccccc} x(0,0) & , & x(0,1) & , & \dots & , & x(0,n-1) \quad , \\ x(1,0) & , & x(1,1) & , & \dots & , & x(1,n-1) \quad , \\ \vdots & & \vdots & & & & \vdots \\ x(m-1,0) & , & x(m-1,1) & , & \dots & , & x(m-1,n-1) \end{array}$$

Output sequence

$$\begin{array}{ccccccc} x(0,0) & , & x(1,0) & , & \dots & , & x(m-1,0) \quad , \\ x(0,1) & , & x(1,1) & , & \dots & , & x(m-1,1) \quad , \\ \vdots & & \vdots & & & & \vdots \\ x(0,n-1) & , & x(1,n-1) & , & \dots & , & x(m-1,n-1) \end{array}$$

OPTIONS

–m	*m*	number of rows	[N/A]
–n	*n*	number of columns	[N/A]

EXAMPLE

Let's assume that the following data is read from *data.in* file in float format.

$$\underbrace{0.0,\ 1.0,\ 2.0,}\ \underbrace{3.0,\ 4.0,\ 5.0,}\ 6.0$$

The command

```
transpose -m 2 -n 3 data.in > data.out
```

will write the following output to *data.out*.

$$\underbrace{0.0,\ 3.0,}\ \underbrace{1.0,\ 4.0,}\ \underbrace{2.0,\ 5.0}$$

NAME

uels – unbiased estimation of log spectrum(2; 3)

SYNOPSIS

uels **[–m** *M* **] [–l** *L* **] [–q** *Q* **] [–i** *I* **] [–j** *J* **] [–d** *D* **] [–e** *e* **] [–E** *E* **] [** ***infile*** **]**

DESCRIPTION

uels uses the unbiased estimation of log spectrum method to calculate cepstral coefficients $c(m)$ from L-length framed windowed input data from infile (or standard input), sending the result to standard output.

Input and output data are in float format.

Until the proposition of the unbiased estimation of log spectrum method, the conventional methods had two main problems. The importance of smoothing the log spectrum was not clear and it could not be guaranteed that the bias of the estimated value would be sufficiently small.

The evaluation procedure to obtain the unbiased estimation log spectrum values is similar to other improved methods to calculate cepstral coefficients. The main difference is that in UELS method a non-linear smoothing is used to guarantee that the estimation will be unbiased.

OPTIONS

–m	*M*	order of cepstrum	[25]
–l	*L*	frame length	[256]
–q	*Q*	input data style	[0]

$Q = 0$ windowed data sequence
$Q = 1$ $20 \times \log |f(w)|$
$Q = 2$ $\ln |f(w)|$
$Q = 3$ $|f(w)|$
$Q = 4$ $|f(w)|^2$

Usually, the options below do not need to be assigned.

–i	*I*	minimum iteration	[2]
–j	*J*	maximum iteration	[30]
–d	*D*	end condition	[0.001]
–e	*e*	small value added to periodogram	[0.0]
–E	*E*	floor in db calculated per frame	[N/A]

EXAMPLE

The example below reads data in float format, evaluates 15-th order log spectrum through UELS method, and sends spectrum coefficients to *data.cep*:

```
frame < data.f | window | uels -m 15 > data.cep
```

SEE ALSO

gcep, mcep, mgcep, lmadf

NAME

ulaw – μ-law compress/decompress

SYNOPSIS

ulaw **[–v** *V* **] [–u** *U* **] [–c] [–d] [** ***infile*** **]**

DESCRIPTION

ulaw converts data between 8-bit μ-law and 16-bit linear formats. The input data is infile (or standard input), and the output is sent to standard output.

If the input is $x(n)$, the output is $y(n)$, the largest value of input data is V, the compression coefficients vector is U, then the compression will be performed using made through the following equation.

$$y(n) = sgn(x(n))V\frac{\log(1 + U\frac{|x(n)|}{V})}{\log(1 + U)}$$

Likewise, the decompression can be performed by applying the following:

$$y(n) = sgn(x(n))V\frac{(1 + u)^{|x(n)|/V} - 1}{U}$$

OPTIONS

–v	*V*	maximum value of input	[32768]
–u	*U*	compression ratio	[256]
–c		coder mode	[TRUE]
–d		decoder mode	[FALSE]

EXAMPLE

In the following, 16-bit data read from *data.s* is compressed to 8-bit ulaw format, and output to *data.ulaw*

```
x2x +sf data.s | ulaw | sopr -d 256 | x2x +fc -r > data.ulaw
```

NAME

us – up-sampling

SYNOPSIS

us **[–s** *S* **] [–c** ***file*** **] [–u** *U* **] [–d** *D* **] [** ***infile*** **]**

DESCRIPTION

us up-samples data from *infile* (or standard input), sending the result to standard output.

The format of input and output data is float. The following filter coefficients can be used.

$S = 23F$	$SPTK/share/SPTK/lpfcoef.2to3f
$S = 23S$	$SPTK/share/SPTK/lpfcoef.2to3s
$S = 34$	$SPTK/share/SPTK/lpfcoef.3to4
$S = 45$	$SPTK/share/SPTK/lpfcoef.4to5
$S = 57$	$SPTK/share/SPTK/lpfcoef.5to7
$S = 58$	$SPTK/share/SPTK/lpfcoef.5to8

($SPTK is the directory where toolkit was installed.)

The ratio between up-sampling and down-sampling can be modified by the **–u** and **–d** options respectively. If you want to specify filter coefficients, **–c** should also be specified.

Filter coefficients are in ASCII format.

For up-sampling from 10 or 12 to 16kHz, the *us16* command can be used. For up/down-sampling between 8, 10, 12 or and 11.025, 22.05 or 44.1 kHz, the *uscd* command can be used. The *ds* command may also be used for down-sampling.

OPTIONS

–s	*S*	conversion type	[58]
		$S = 23F$ up-sampling by 2 : 3	
		$S = 23S$ up-sampling by 2 : 3	
		$S = 34$ up-sampling by 3 : 4	
		$S = 45$ up-sampling by 4 : 5	
		$S = 57$ up-sampling by 5 : 7	
		$S = 58$ up-sampling by 5 : 8	
–c	*file*	filename of low pass filter coefficients	[Default]
–u	*U*	up-sampling ratio	[N/A]
–d	*D*	down-sampling ratio	[N/A]

EXAMPLE

In this example, the speech data in the input file *data.16*, which was sampled at 16 kHz in short int format, is converted to an 44.1 kHz sampling rate:

```
x2x +sf data.16 | us -s 23F | us -s 23S | us -s 57 | \
us -c /usr/local/SPTK/lib/lpfcoef.5to7 -u 7 -d 8 | \
x2x +fs > data.44
```

Note: $\frac{44100}{16000} = \frac{3 \times 3 \times 7 \times 7 \times 100}{2 \times 2 \times 5 \times 8 \times 100}$

SEE ALSO

ds, uscd, us16

NAME

us16 – up-sampling from 10 or 12 kHz to 16 kHz

SYNOPSIS

us16 **[–s** *S* **] [** ***infile*** **] [** ***outfile*** **]**

us16 **[–s** *S* **]** ***infile1*** **. . . [** ***infileN*****] *outdir***

DESCRIPTION

us16 upsamples data from 10 kHz or 12 kHz to 16 kHz. If the arguments *infile* and *outfile* are not given, standard input and standard output are used. If several input files are given, the last argument is considered as a directory name and multiple output files are created in that directory, with names similar to the input file names but with file extensions changed to ".16".

OPTIONS

–s *S* input sampling frequency 10—12 kHz [10]

EXAMPLE

In the example below, speech data sampled at 10 kHz is read from *data.10*, upsampled to 16 kHz, and the results are written to data.16:

```
us16 -s 10 < data.10 > data.16
```

SEE ALSO

ds, us, uscd

NAME

uscd – up/down-sampling from 8, 10, 12, or 16 kHz to 11.025, 22.05, or 44.1 kHz

SYNOPSIS

uscd **[–s** *S S* **] [** ***infile*** **] [** ***outfile*** **]**

uscd **[–s** *S S* **]** ***infile1*** *...* **[** ***infileN*****]** ***outdir***

DESCRIPTION

uscd converts the sample rate from one of 8, 10, 12, or 16 kHz to one of 11.025, 22.04, or 44.1 kHz. If *infile* and *outfile* arguments are not given, standard input and output are used. If the last argument given names a directory, each of the preceding argument files is re-sampled. The results are stored in multiple files in that directory, with base names the same as the input file base names, but with extensions indicating the new sample rate.

OPTIONS

–s	*S* 1	input sampling frequency (one of 8, 10, 12 or 16)	[10]
–S	*S* 2	output sampling frequency (one of 11.025, 22.05, or 44.1) *S* 2 can be abbreviated as 11, 22, or 44. If the last command line argument is a directory name, the suffix for the output files is either ".11", ".22", or ".44."	[11.025]

EXAMPLE

In the example below, speech data sampled at 16 kHz is read from *data.16*, upsampled to 22.05 kHz, and the results are written to data.22:

```
uscd -s 16 22.05 < data.16 > data.22
```

SEE ALSO

ds, us, us16

NAME

vopr – execute vector operations

SYNOPSIS

vopr [–l *L* **] [–n** *N* **] [–i] [–a] [–s] [–m] [–d] [–ATAN2] [–AM] [–GM]**
[–gt] [–ge] [–lt] [–le] [–eq] [–ne] [***file1*** **] [** ***infile*** **]**

DESCRIPTION

This command performs vector operations in input files. In other words

file1 first vector file (if it is not assigned then stdin)

infile second vector file (if it is not assigned then stdin)

the first file gives the operation vectors **a** and the second file gives the operation vectors **b**. The assigned operation is undertaken and the results are sent to the standard output.

Input and output data are in float format.

The undertaken action depends on the number of assigned files as well as the vector lengths as exemplified in the following.

If two files are assigned (when only one file is assigned, it is assumed that it corresponds to *infile*) then, depending on the vector sizes, the following actions are taken.

when $L = 1$

file1 (stdin)	a_1	a_2	...	a_i	...
infile	b_1	b_2	...	b_i	...
Output (stdout)	y_1	y_2	...	y_i	...

One data from one file corresponds to one data on the other file.

when $L \geq 2$

file1 (stdin)	$a_{11},\ldots,a_{1L}$	$a_{21},\ldots,a_{2L}$	$a_{31},\ldots,a_{3L}$	$a_{41},\ldots$
infile	$b_1,\ldots,b_L$			
Output (stdout)	$y_{11},\ldots,y_{1L}$	$y_{21},\ldots,y_{2L}$	$y_{31},\ldots,y_{3L}$	$y_{41},\ldots$

In this case, the operation vector is read only once from *infile*, and the operations are recursively performed.

When the information related to **a** and **b** is contained in a single file, (if only one file is assigned, or if no file assignment is made), the –i option should be used and the action does not depend on the vector length.

when $L \geq 1$

file (stdin)	$a_{11},\ldots,a_{1L}$	$b_{11},\ldots,b_{1L}$	$a_{21},\ldots,a_{2L}$	$b_{21},\ldots,b_{2L}$	
Output (stdout)	$y_{11},\ldots,y_{1L}$		$y_{21},\ldots,y_{2L}$		

Input vectors are read from a single file.

OPTIONS

–l	*L*	length of vector	[1]
–n	*N*	order of vector	[L-1]
–i		when a single file file is specified, the file contains a and b.	[FALSE]
–a		addition $y_i = a_i + b_i$	[FALSE]
–s		subtraction $y_i = a_i - b_i$	[FALSE]
–m		multiplication $y_i = a_i * b_i$	[FALSE]
–d		division $y_i = a_i / b_i$	[FALSE]
–ATAN2		atan2 $y_i = \text{atan}\,2(b_i, a_i)$	[FALSE]
–AM		arithmetic mean $y_i = (a_i + b_i)/2$	[FALSE]
–GM		geometric mean $y_i = \sqrt{a_i * b_i}$	[FALSE]
–c		choose smaller value	[FALSE]
–f		choose larger value	[FALSE]
–gt		decide “greater than”	[FALSE]
–ge		decide “greater than or equal”	[FALSE]
–lt		decide “less than”	[FALSE]
–le		decide “less than or equal”	[FALSE]
–eq		decide “equal to”	[FALSE]
–ne		decide “not equal to”	[FALSE]

EXAMPLE

The output file *data.c* contains addition of vectors in float format read from *data.a* and *data.b*:

```
vopr -a data.a data.b > data.c
```

In the following example, a sin wave is passed through a window with length 256 and coefficients given from *data.w*:

```
sin -p 30 -l 1000 | vopr data.w -l 256 -m | fdrw | xgr
```

Similar results as from the above example can be obtained using the following: Here, it is considered that the contents of *data.w* correspond to a Blackman window:

```
sin -p 30 -l 1000 | window | fdrw | xgr
```

For other examples, suppose *data.a* contains

$$1, 2, 3, 4, 5, 6, 7$$

in float format and *data.b* contains

$$3, 2, 1, 0, 5, 6, 7$$

in float format. In the following example, smaller scalar values can be taken from *data.a* and *data.b*, and the result is sent to *data.c* in float format.

```
vopr -c data.b < data.a > data.c
```

The output file *data.c* contains

$$1, 2, 1, 0, 5, 6, 7.$$

When executing following command line,

```
vopr -ge data.b < data.a > data.c
```

the output file *data.c* contains:

$$0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0$$

On the other hand, when executing following command line,

```
vopr -gt data.b < data.a > data.c
```

the output file *data.c* contains:

$$0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0$$

Moreover, when executing following command line,

```
vopr -eq data.b < data.a > data.c
```

the output file *data.c* contains:

$$0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0$$

SEE ALSO

sopr, vsum

NAME

vq – vector quantization

SYNOPSIS

vq [–l *L* **] [–n** *N* **] [–q]** ***cbfile*** **[*infile*]**

DESCRIPTION

vq uses vector quantization to compress vectors from *infile* (or standard input) according to the codebook *cbfile*, sending either codebook indexes or quantized vectors to standard output.

For each length L input vector

$$x(0), x(1), \ldots, x(L-1),$$

vq finds the codebook vector $\boldsymbol{c}_i$ that minimizes the Euclidean distance

$$d_i = \frac{1}{L}\sum_{m=0}^{L-1}(x(m) - c_i(m))^2.$$

Input data is in float format. If the –q option is given, the output is the code vector $[c_i(0), c_i(1), \cdots, c_i(L-1)]$ in float format. If the –q option is not given, the output is the codebook index i in int format.

OPTIONS

–l	*L*	length of vector	[26]
–n	*N*	order of vector	[L-1]
–q		output quantized vector	[FALSE]

EXAMPLE

In this example, a sequence of length 25 is read from *data.f* in float format. it is quantized using codebook *cbfile*, and the results are written to *data.vq*:

```
vq -q cbfile < data.f > data.vq
```

SEE ALSO

ivq, msvq, imsvq, lbg

NAME

vstat – vector statistics calculation

SYNOPSIS

vstat **[–l** *L* **] [–n** *N* **] [–t** *T* **] [–c** *C* **] [–d] [–o** *O* **] [** ***infile*** **]**

DESCRIPTION

vstat calculates the mean and covariance of groups of vectors from *infile* (or standard input), sending the result to standard output.

For each group of T input vectors of length L, *vstat* calculates the mean vector of length L and the $L \times L$ covariance matrix. In other words, if the input data is:

$$\overbrace{\overbrace{x_1(1),\ldots,x_1(L)}^{L},\overbrace{x_2(1),\ldots,x_2(L)}^{L},\ldots,\overbrace{x_N(1),\ldots,x_N(L)}^{L},\ldots}^{T\times L}$$

then the output will be given by:

$$\overbrace{\overbrace{\mu(1),\ldots,\mu(L)}^{L},\overbrace{\overbrace{\sigma(11),\ldots,\sigma(1L)}^{L},\ldots\overbrace{\sigma(L1),\ldots,\sigma(LL)}^{L},\ldots}^{L\times L}}$$

and the values of $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ can be obtained through the following:

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{k=1}^{N}\boldsymbol{x}$$

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{k=1}^{N}\boldsymbol{x}\boldsymbol{x}' - \boldsymbol{\mu}\boldsymbol{\mu}'$$

If the –d option is given, the length L diagonal of the covariance matrix is outputted instead of the entire $L \times L$ matrix.

If the –o 3 option is specified, *vstat* also calculates the confidence interval of the mean via Student's t-distribution for each dimension, i.e. for each dimension, the confidence interval can be estimated at the confidence level α (%) satisfying the following condition:

$$t(\alpha,\phi) \geq \left|\frac{\mu(i) - m(i)}{\sqrt{\hat{\sigma(i)}^2/L}}\right|, \quad i = 1,2,\ldots,L$$

where $t(\alpha,\phi)$ is the upper $0.5(100-\alpha)$-th percentile of the t-distribution with ϕ degrees of freedom, $m(i)$ is the population mean, $\hat{\sigma(i)}^2$ is the unbiased variance. The confidence

level α can be specified by the –c option. The upper and lower bounds $u(i)$ and $l(i)$ can be written as

$$u(i) = \mu(i) + t(\alpha, L-1)\sqrt{\frac{\hat{\sigma(i)}^2}{L}},$$

$$l(i) = \mu(i) - t(\alpha, L-1)\sqrt{\frac{\hat{\sigma(i)}^2}{L}}.$$

The order of the output is as follows.

$$\overbrace{\mu(1), \ldots, \mu(L)}^{L}, \overbrace{u(1), \ldots, u(L)}^{L}, \overbrace{l(1), \ldots, l(L)}^{L}$$

If the –o 4 option is specified, *vstat* outputs the median of input vectors of length L. If the number of vectors is even number, *vstat* outputs the arithmetic mean of two vectors of center.

Also, input and output data are in float format.

OPTIONS

–l	*L*	length of vector	[1]
–n	*N*	order of vector	[L-1]
–t	*T*	number of vector	[N/A]
–o	*O*	output format	[0]

$O = 0$	mean & covariance
$O = 1$	mean
$O = 2$	covariance
$O = 3$	mean & upper / lower bound of confidence interval via Student's t-distribution
$O = 4$	median

–c	*C*	confidence level of confidence interval (%)	[95.00]
–d		diagonal covariance	[FALSE]
–i		output inverse covariance instead of covariance	[FALSE]
–r		output correlation instead of covariance	[FALSE]

EXAMPLE

The output file *data.stat* contains the mean and covariance matrix taken from the whole data in *data.f* read in float format.

```
vstat data.f > data.stat
```

In the example below, the mean of 15-th order coefficients vector is taken for every group of 3 frames and sent to *data.av*:

```
vstat -l 15 -t 3 -o 1 data.f > data.av
```

The output file *data.stat* contains the mean and upper / lower bound of the confidence interval (90%) calculated via Student's t-distribution.

```
vstat -C 90.0 -o 3 data.f > data.stat
```

SEE ALSO

average, vsum

NAME

vsum – summation of vector

SYNOPSIS

vsum **[–l** *L* **]** **[–n** *N* **]** **[** ***infile*** **]**

DESCRIPTION

vsum calculates the vector sum of groups of N input vectors of length L from *infile* (or standard input), sending the result to standard output. That is, if the input data is given by

$$\overbrace{\overbrace{a_1(1),\ldots,a_1(L)}^{L},\overbrace{a_2(1),\ldots,a_2(L)}^{L},\ldots,\overbrace{a_N(1),\ldots,a_N(L)}^{L},\ldots}^{N\cdot L}$$

then the output is

$$\overbrace{s(1),\ldots,s(L)}^{L},\ldots$$

,where $s(n)$ can be written as

$$s(n)=\sum_{k=1}^{N}a_k(n)$$

Input and output data are in float format.

OPTIONS

–l	*L*	order of vector	[1]
–n	*N*	number of vector	[EOD]

EXAMPLE

The output file *data.sum* contains the summation of the whole data in file *data.f* read in float format:

```
vsum data.f > data.sum
```

In this example, the norm of 10-th order vectors are evaluated and written to *data.n*:

```
sopr data.f -P | vsum -n 10 | sopr -R > data.n
```

In the next example, 15-th order coefficients vectors are read from *data.f*, the average for every 3 frames is evaluated, and output to *data.av*:

```
vsum -l 15 -n 3 data.f | sopr -d 3 > data.av
```

SEE ALSO

sopr

NAME

wav2raw – wav (RIFF) to raw

SYNOPSIS

wav2raw [–swab] [–d *D* **] [–n] [–N] [–L] [–R] [*+type*] [*infile*]**

DESCRIPTION

wav2raw converts file format from wav to raw.

OPTIONS

–swab		change endian	[FALSE]
–d	*D*	destination directory	[N/A]
–n		normalization with the maximum value according to bit/sample of the wav file if max >= 255 (8bit), 32767 (16bit), 8388067 (24bit) or 2147483647 (32bit)	[FALSE]
–N		normalization with the maximum value	[FALSE]
–L	*L*	convert left sound from stereo wav file	[FALSE]
–R	*R*	convert right sound from stereo wav file	[FALSE]
+type		output data type	[f]

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
f	float (4 bytes)	d	double (8 bytes)
a	ascii		

EXAMPLE

In the following example, the file *data.wav* is converted to *data.raw* and normalized with the maximum value. The output will be saved in the same directory as *data.wav* unless the *-d* option is given:

```
wav2raw -N data.wav
```

SEE ALSO

raw2wav, swab

NAME

window – data windowing

SYNOPSIS

window [**–l** L_1] [**–L** L_2] [**–n** N] [**–w** W] [***infile***]

DESCRIPTION

window multiplies, on an element-by-element basis, length L input vectors from *infile* (or standard input) by a specified windowing function, sending the result to standard output.

For the input data

$$x(0), x(1), \ldots, x(L_1 - 1)$$

and the windowing function

$$w(0), w(1), \ldots, w(L_1 - 1),$$

the output is calculated as follows:

$$x(0) \cdot w(0),\ x(1) \cdot w(1),\ \ldots,\ x(L_1 - 1) \cdot w(L_1 - 1).$$

If L_2 is greater then L_1, then 0s are added to the output as follows.

$$\underbrace{x(0) \cdot w(0),\ x(1) \cdot w(1),\ \ldots,\ x(L_1 - 1) \cdot w(L_1 - 1), 0, \ldots, 0}_{L_2}$$

Input and output data are in float format.

OPTIONS

–l L_1 frame length of input ($L \leq 2048$) [256]
–L L_2 frame length of output [L_1]
–n N type of normalization [1]

- 0 no normalization
- 1 normalization as $\sum_{n=0}^{L-1} w^2(n) = 1$
- 2 normalization as $\sum_{n=0}^{L-1} w(n) = 1$

–w W type of window [0]

- 0 Blackman
- 1 Hamming
- 2 Hanning
- 3 Bartlett
- 4 trapezoid
- 5 rectangular

EXAMPLE

This example prints in the screen a sin wave function with period 20 after windowing it with a Blackman window:

```
sin -p 20 | window | fdrw | xgr
```

This example passes the excitation generated through a train pulse by a digital filter, applies a Blackman windowing function to it, evaluates the log magnitude spectrum through 512 points FFT, and plots the results on the screen:

```
train -p 50 | dfs -a 1 0.9 | window -l 50 -L 512 |\
spec -l 512 | fdrw | xgr
```

SEE ALSO

fftr, spec

NAME

x2x – data type transformation

SYNOPSIS

x2x **[+***type*1 **] [+***type*2 **] [** *%format* **] [+a***N* **] [–r]**

DESCRIPTION

x2x converts data from standard input to a different data type, sending the result to standard output.

The input and output data type are specified by command line options as described below.

OPTIONS

+*type*1	input data type	[f]
+*type*2	output data type	[type1]

both options *type*1, *type*2 can be assigned. one of the options below.

c	char (1 byte)	C	unsigned char (1 byte)
s	short (2 bytes)	S	unsigned short (2 bytes)
i3	int (3 bytes)	I3	unsigned int (3 bytes)
i	int (4 bytes)	I	unsigned int (4 bytes)
l	long (4 bytes)	L	unsigned long (4 bytes)
le	long long (8 bytes)	LE	unsigned long long (8 bytes)
f	float (4 bytes)	d	double (8 bytes)
de	long double (12 bytes)	a	ASCII
aN	ASCII specifying the column number *N*		

data type is converted from $t_1(type_1)$ to $t_2(type_2)$. if t_2 is not assigned then no operation is performed, and the output file is equal to the input file.

–r	specify rounding off when a real number is substituted for an integer	[FALSE]
–o	clip by minimum and maximum of output data type if input data is over the range of output data type. if the -o option is not given, when the data type lengths are different, the process will be aborted.	[FALSE]
+**a***%format*	specify output format similar to 'printf()', only if *type*2 is ASCII.	[*%g*]

EXAMPLE

The following example converts data in ASCII format read from *data.asc* into float format, and writes the output to *data.f*:

```
x2x +af < data.asc > data.f
```

This example reads data in float format from *data.f*, converts it to ASCII format, and sends the output to the screen:

```
x2x +fa < data.f
```

For example, if the contents of *data.f* in float format are

$$1, 2, 3, 4, 5, 6, 7$$

then the following output is printed to the screen.

```
1
2
3
4
5
6
7
```

If for the same data in the example above, the number of columns is assigned:

```
x2x +fa3 < data.f
```

the output will be:

```
1       2       3
4       5       6
7
```

The output uses the printf command %e format:

```
x2x +fa%9.4e < data.f
```

In this example the total number of characters for each number is 11, and the number of decimal points assigned to 4.

```
1.0000e+000
2.0000e+000
    ⋮
7.0000e+000
```

SEE ALSO

dmp

NAME

xgr – XY-plotter simulator for X-window system

SYNOPSIS

xgr **[–s *S*] [–l] [–rv] [–m] [–bg *BG*] [–hl *HL*] [–bd *BD*]**
[–ms *MS*] [–g *G*] [–d *D*] [–t *T*] [*infile*]

DESCRIPTION

xgr plots a graph from a sequence of FP5301 plotter commands, displaying the output on the screen in a new X window.

When the X window is created, the keyboard focus is initially assigned to that new window, which responds to a limited set of user interactions:

- Changing the window size truncates or expands the area in which the graph is displayed, but the graph remains the same size (i.e. it is not rescaled to fit the new window size).
- If the graph is larger than the window, the position within the window can be changed with "vi" cursor movement commands:

  h: left scroll
  j: down scroll
  k: up scroll
  l: right scroll

- To delete the window, type one of the following: "q","Ctrl-c","Ctrl-d"

OPTIONS

–s	*S*	shrink	[3.38667]
–l		landscape	[FALSE]
–rv		reverse mode	[FALSE]
–m		monochrome display mode	[FALSE]
–bg	*BG*	background color	[white]
–hl	*HL*	highlight color	[blue]
–bd	*BD*	border color	[blue]
–ms	*MS*	mouse color	[red]
–g	*G*	geometry	[NULL]
–d	*D*	display	[NULL]
–t	*T*	window title	[xgr]

EXAMPLE

The following example uses fdrw to draw a graph based on data read from *data.f*, and sends the output to a X-Window environment:

```
fdrw < data.f | xgr
```

BUGS

- If the display server does not contain backing store function, then the hidden part of virtual screen is erased.
- To reduce the waiting time to display graphs, an image of virtual screen is copied to the memory. If the size assigned by the –g option is too small or if during the time the graph is being plotted another window is put above the virtual screen, a part of the virtual screen needs to be erased. The –s option is suggested whenever the size of the virtual screen should be reduced.

SEE ALSO

fig, fdrw

NAME

zcross – zero cross

SYNOPSIS

zcross [–l *L* **] [–n] [** ***infile*** **]**

DESCRIPTION

zcross determines the number of zero crossings within each length *L* input vector, sending the result to standard output as one float number for each input vector.

Input and output data are in float format.

OPTIONS

–l	*L*	frame length if $L \leq 0$ then no data output.	[256]
–n		normalized by frame length	[FALSE]

EXAMPLE

Data in float format is read from *data.f*, a zero crossing rate is computed, and the results are written to *data.zc*:

```
zcross < data.f > data.zc
```

SEE ALSO

frame, spec

NAME

zerodf – all zero digital filter for speech synthesis

SYNOPSIS

zerodf [–m *M* **] [–p** *P* **] [–i** *I* **] [–t] [–k]** ***bfile*** **[** ***infile*** **]**

DESCRIPTION

zerodf derives a standard-form FIR (all-zero) digital filter from the coefficients $b(0), b(1), \ldots, b(M)$ in *bfile* and uses it to filter an excitation sequence from *infile* (or standard input) to synthesize speech data, sending the result to standard output.

Input and output data are in float format.

The transfer function $H(z)$ of an FIR filter in standard form is

$$H(z) = \sum_{m=0}^{M} b(m) z^{-m}$$

OPTIONS

–m	*M*	order of coefficients	[25]
–p	*P*	frame period	[100]
–i	*I*	interpolation period	[1]
–t		transpose filter	[FALSE]
–k		filtering without gain	[FALSE]

EXAMPLE

In the following example, Excitation is generated from pitch information read in float format from *data.pitch*. It is then passed through a FIR filter with coefficients read from *data.b*, and the synthesized speech is written to *data.syn*:

```
excite < data.pitch  | zerodf data.b > data.syn
```

SEE ALSO

poledf, lmadf

REFERENCES

[1] S. Imai and Y. Abe, "Spectral envelope extraction by improved cepstral method," *Journal of IEICE*, Vol.J62-A, No.4, pp.217–223, Apr. 1987. (*in Japanese*)

[2] S. Imai and C. Furuichi, "Unbiased estimation of log spectrum," *Journal of IEICE*, Vol.J70-A, No.3, pp.471–480, Mar. 1987. (*in Japanese*)

[3] S. Imai and C. Furuichi, "Unbiased estimator of log spectrum and its application to speech signal processing," *Signal Processing IV: Theory and Applications*, Vol.1, pp.203–206, Elsevier, North-Holland, 1988.

[4] K. Tokuda, T. Kobayashi, S. Shiomoto, and S. Imai, "Adaptive cepstral analysis — Adaptive filtering based on cepstral representation —," *Journal of IEICE*, Vol.J73-A, No.7, pp.1207–1215, July 1990. (*in Japanese*)

[5] K. Tokuda, T. Kobayashi, and S. Imai, "Adaptive cepstral analysis of speech," *IEEE Trans. Speech and Audio Process.*, Vol.3, No.6, pp.481–488, Nov. 1995.

[6] K. Tokuda, T. Kobayashi, R. Yamamoto, and S. Imai, "Spectral estimation of speech based on generalized cepstral representation," *Journal of IEICE*, Vol.J72-A, No.3, pp.457–465, Mar. 1989. (*in Japanese*)

[7] T. Kobayashi and S. Imai, "Spectral analysis using generalized cepstrum," *IEEE Trans. Acoust., Speech, Signal Process.*, Vol.ASSP-32, No.5, pp.1087–1089, Oct. 1984.

[8] K. Tokuda, T. Kobayashi, and S. Imai, "Generalized cepstral analysis of speech — a unified approach to LPC and cepstral method," *Proc. ICSLP-90*, pp.37–40, Nov. 1990.

[9] T. Fukada, K. Tokuda, T. Kobayashi, and S. Imai, "A study on adaptive generalized cepstral analysis," *IEICE Spring National Convention*, A-150, p.150, Mar. 1990. (*in Japanese*)

[10] K. Tokuda, T. Kobayashi, T. Fukada, H. Saito, and S. Imai, "Spectral estimation of speech based on mel-cepstral representation," *Journal of IEICE*, Vol.J74-A, No.8, pp.1240–1248, Aug. 1991. (*in Japanese*)

[11] K. Tokuda, T. Kobayashi, T. Fukada, and S. Imai, "Adaptive mel-cepstral analysis of speech," *Journal of IEICE*, Vol.J74-A, No.8, pp.1249–1256, Aug. 1991. (*in Japanese*)

[12] T. Fukada, K. Tokuda, T. Kobayashi, and S. Imai, "An adaptive algorithm for mel-cepstral analysis of speech," *Proc. ICASSP-92*, pp.137–140, Mar. 1992.

[13] K. Tokuda, T. Kobayashi, K. Chiba, and S. Imai, "Spectral estimation of speech by mel-generalized cepstral analysis," *Journal of IEICE*, Vol.J75-A, No.7, pp.1124–1134, July 1992. (*in Japanese*)

[14] K. Tokuda, T. Kobayashi, T. Masuko, and S. Imai, "Mel-generalized cepstral analysis — a unified approach to speech spectral estimation," *Proc. ICSLP-94*, pp.1043–1046, Sep. 1994.

[15] T. Wakako, K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura, "Speech spectral estimation based on expansion of log spectrum by arbitrary basis functions,", *Journal of IEICE*, Vol.J82-D-II, No.12, pp.2203–2211, Dec. 1999. (*in Japanese*)

[16] C. Miyajima C, H. Watanabe, K. Tokuda, T. Kitamura, and S. Katagiri, "A new approach to designing a feature extractor in speaker identification based on discriminative feature extraction," *Speech Communication*, Vol.35, No.3, pp.203–218, Oct. 2001.

[17] S. Imai, "Log magnitude approximation (LMA) filter," *Journal of IEICE*, Vol.J63-A, No.12, pp.886–893, Dec. 1987. (*in Japanese*)

[18] T. Chiba, K. Tokuda, T. Kobayashi, and S. Imai, "Speech synthesis based on mel-generalized cepstral representation," *IEICE Spring National Convention*, A-243, p.243, Mar. 1988. (*in Japanese*)

[19] S. Imai, "Cepstral analysis synthesis on the mel frequency scale," *Proc. ICASSP-83*, pp.93–96, Apr. 1983.

[20] S. Imai, K. Sumita, and C. Furuichi, "Mel log spectrum approximation (MLSA) filter for speech synthesis," *Journal of IEICE*, Vol.J66-A, No.2, pp.122–129, Feb. 1983. (*in Japanese*)

[21] T. Kobayashi, S. Imai, and Y. Fukuda, "Mel generalized-log spectrum approximation (MGLSA) filter," *Journal of IEICE*, Vol.J68-A, No.6, pp.610–611, June 1985. (*in Japanese*)

[22] K. Koishida, G. Hirabayashi, K. Tokuda, and T. Kobayashi, "A 16kbit/s wideband CELP-based speech coder using mel-generalized cepstral analysis," *IEICE Trans. Inf. and Syst.*, vol.E83-D, no.4, pp.876–883, Apr. 2000.

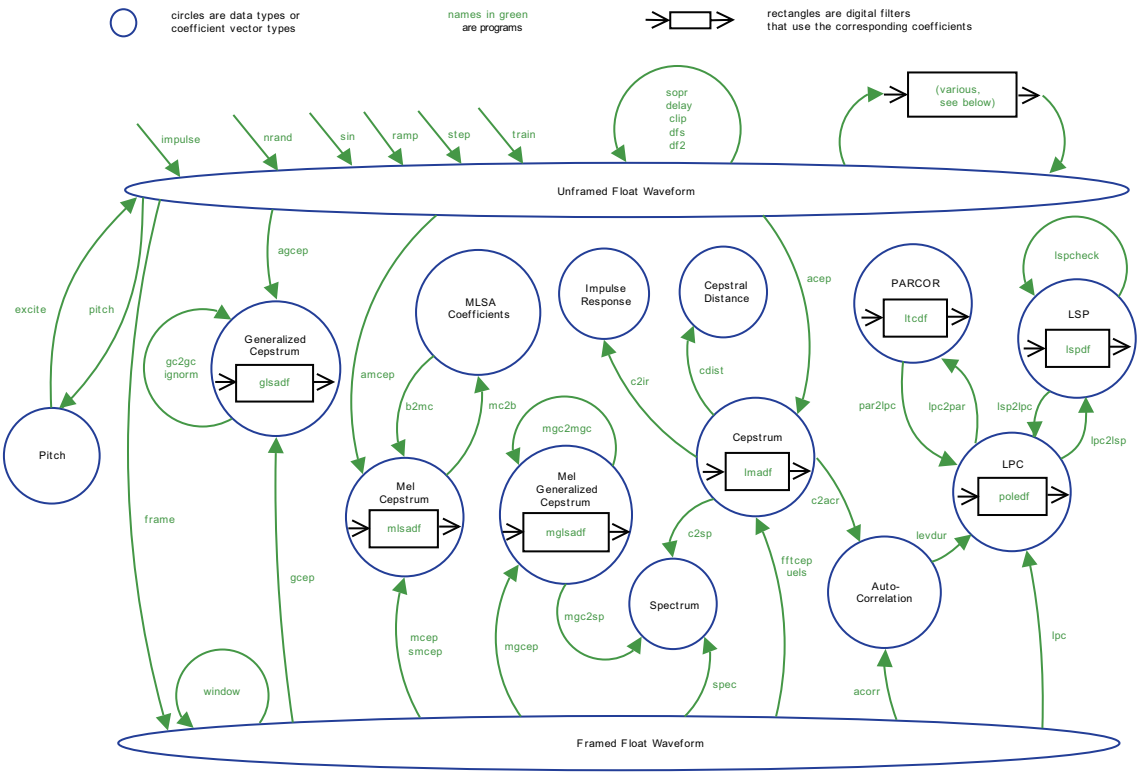
[23] K. Tokuda, T. Masuko, T. Yamada, T. Kobayashi, and S. Imai, "An algorithm for speech parameter generation from continuous mixture HMMs with dynamic features," *Proc. EUROSPEECH-95*, pp.757–760, Sep. 1995.

[24] D. Talkin, "A Robust Algorithm for Pitch Tracking (RAPT)," in *Speech Coding & Synthesis*, W. B. Kleijn and K. K. Pailwal (Eds.), Elsevier, pp.495–518, 1995.

[25] A. Camacho, "SWIPE: A Sawtooth Waveform Inspired Pitch Estimator for Speech And Music," Ph.D. Thesis, University of Florida, 116p., 2007.

Block diagram of SPTK commands

Mitch Bradley kindly provided us the following diagram to help users understand and remember the relationships between the SPTK commands and data representations.



INDEX of TOPICS