ORCHESTRA STATEMENTS

ALLGEMEINES

Jedes Orchestrafile besteht aus einem *Header,* in dem einige globale Parameter festgelegt werden, sowie aus einer oder mehreren Instrumentendefinitionen (*instrument block*), in denen die Synthese und Bearbeitung von Klängen mit Hilfe verschiedener *Anweisungen* beschrieben wird.

Eine Anweisung (*statement*) im Orchestrafile hat im allgemeinen folgendes Format: **opcode**

```
label:   result     opcode     argument1,argument2,... ;comments
```

Das `label` ist optional und dient zum Markieren der Zeile für eine `goto`-Anweisung.
Kommentare werden durch ein Semikolon (`;`) eingeleitet und gelten bis zum Ende der Zeile.
Sie sind optional und können auch alleine in einer Zeile stehen.
Die eigentliche Anweisung wird durch `result`, **`opcode`** und `arguments` gebildet. Eine Anweisung darf sich nicht über mehrere Zeilen erstrecken.
Ein `result` ist der Ergebniswert einer Operation, die durch den **`opcode`** und einen oder mehrere Eingabewerte (`argument`) spezifiziert wird.
`Results` werden in *Variablen* gespeichert. Die `arguments` können in Form von *Konstanten, Variablen,* Score-*Parametern* sowie *Ausdrücken* angegeben werden.
Die verschiedenen Anweisungen werden jeweils zu bestimmten Zeiten ausgeführt:

1.) bei der Initialisierung des Orchestras
2.) zu Beginn jeder Note: init-time *i-time*
3.) zu Beginn jeder Kontrollschleife: performance-time control-rate *k-rate*
4.) für jedes Sample: performance-time audio-rate *a-rate*

Die Initialisierung des Orchestras beinhaltet das Setzen der Samplingrate **`sr`**, der Kontrollrate **`kr`** und der Anzahl der Audiokanäle **`nchnls`** (mono, stereo, quadro). Diese Werte können sich für das jeweilige Orchestrafile nicht mehr ändern.

Die Initialisierung der Instrumente erfolgt mit jedem Aufruf dieses Instruments durch eine Note des Scorefiles neu. Hier werden die die *i-Variablen* gesetzt und *p-Parameter* aus dem Scorefile übergeben. Dieser Zeitpunkt zu Beginn jedes Instrumentenaufrufs wird *init-time*, *init-rate* bzw. *i-time* genannt. **i-time**

Die Berechnung eines Klanges nach den Vorgaben des Instruments und der Notenparameter läuft in 2 Zeitebenen ab: Kontrollrate **`kr`** und Samplingrate **`sr`**. Der gesamte Vorgang der Berechnung heißt *performance-time*.

Die Kontrollschleife, in der sich nur relativ langsam ändernde Werte *control-signals* (Hüllkurven, LFO's etc.) berechnet werden, wird genau **`kr`**-mal je Sekunde aufgerufen. Die jetzt berechneten Werte gelten bis zur nächsten Kontrollschleife, d.h. **`1/kr`** Sekunden. Der Zeitpunkt der Neuberechnung der Kontrollwerte bzw. die Berechnungsrate heißt *k-time, control-rate* bzw. *k-rate.* **k-rate**

Die Audiosignale (Samples) werden in jeder Schleife aktualisiert, d.h. **`sr`**-mal je Sekunde, bzw. **`ksmps`** Audiosamples auf einmal. Diese Zeitebene heißt *a-time*, *audio-rate* bzw. *a-rate*. **a-rate**

KONSTANTEN UND VARIABLEN

Konstanten sind Fließkommazahlen, wie z.B. 3, 0.237 oder -180.4562, die ihren Wert nicht ändern.

Variablen speichern Zahlen, deren Wert zu bestimmten Zeiten geändert werden kann.
Über einen Variablenbezeichner kann auf diese Zahlen zugegriffen werden.
Man unterscheidet zwischen *lokalen, globalen* und *reservierten* Variablen.

lokale Variablen

Lokale Variable werden innerhalb eines Instrumentenblocks definiert und gelten nur für diesen Block.
Der Variablenbezeichner besteht aus einem Namen und vorangestelltem **i**, **k** oder **a** bzw. aus einer *pfield number* mit vorangestelltem **p**:

iname - *init variables* werden einmal zu Beginn der Performance (Berechnung eines Klangs bzw. einer Note) initialisiert und verändern sich danach nicht mehr.

kname - *control signals* werden zu Beginn jeder Kontroll-Schleife neu berechnet.

aname - *audio signals* werden für jedes Sample berechnet.

pnumber - *parameter fields* dienen zur Übernahme von Werten aus dem Scorefile. Die **p**-Werte können im Orchestrafile auch nachträglich verändert werden. Sie werden ansonsten wie Init-Variablen behandelt.

globale Variablen

Globale Variablen gelten für alle Instrumente gleichzeitig und können innerhalb und außerhalb von Instrumentenblöcken definiert und geändert werden. Sie dienen hauptsächlich der Kommunikation zwischen Instrumenten und übergeordneten Operationen, wie z.B. Mischen oder Verhallung. Die Bezeichnung erfolgt wie bei lokalen Variablen mit zusätzlich vorangestelltem **g** :

giname, **gk**name, **ga**name.

(Zu *reservierten* Variablen siehe *header statements*).

HEADER

```
sr = n1                                                    sr
kr = n2                                                    kr
ksmps = n3                                                 ksmps
nchnls = n4                                                nchnls
```

Der Header besteht aus Wertzuweisungen an reservierte globale Variable. Er muß am Beginn jedes Orchestrafiles stehen. Der Header kann auch als Instrument 0 aufgefaßt werden.
Alle Werte im Header sind optional.

sr - Samplingfrequenz in Hz. Der voreingestellte Wert ist 44100.

kr - Kontrollrate in Hz. Gewöhnlich ein Bruchteil von **sr**. Voreingestellt ist 4410.

ksmps - Anzahl der Samples pro Kontrollperiode. Der Wert muß **sr/kr** entsprechen!. Voreingestellt ist 10.

nchnls - Anzahl der Audiokanäle (1,2 oder 4). Voreingestellt ist 1 (mono).

Die Werte dieser Variablen dürfen nicht mehr verändert werden.
Vor der ersten Instrumentendefinition können weitere globale Variablen initialisiert werden.
Die Bit-Auflösung des zu erzeugenden Soundfiles (8 bit, 16 bit, 20 bit, float o.a.) kann nicht im Orchestrafile spezifiziert werden. Sie wird statt dessen per Csound-Flag oder durch die jeweilige Csound-Shell (systemabhängig) eingestellt.

INSTRUMENT BLOCK

```
instr   n                                                  instr
.                                                          endin
.       (Anweisungen)
.
endin
```

Diese Statements klammern Anweisungen zu einem Block. Innerhalb eines Blocks darf keine weitere Instrumentendefinition erfolgen.
n muß eine positive ganze Zahl sein.
Es sind beliebig viele Instrumente in beliebiger Reihenfolge im Orchestrafile erlaubt.

ARITHMETISCHE OPERATOREN

`- a`	
`+ a`	
`a && b`	*(logisches AND; keine a-rate)*
`a || b`	*(logisches OR; keine a-rate)*
`a + b`	
`a - b`	
`a * b`	
`a / b`	
`a ^ b`	(*Potenz: a hoch b)*
`a % b`	(*a modulo b)*

a und b können beliebige Ausdrücke sein.

Vorrangregeln:

```
*, /
+, -
&&
||
```

Bei Operatoren gleichen Rangs wird von links nach rechts abgearbeitet.
Durch Klammern ( ) kann eine beliebige Reihenfolge erzwungen werden.

Beipiele:

`a + b * c`	ist äquivalent zu	`a + (b * c)`
`a && b - c || d`	ist äquivalent zu	`(a && (b - c)) || d`
`a - b - c`	ist äquivalent zu	`(a - b) - c a - (b - c) ist a - b + c`

WERTZUWEISUNGEN

=

```
ir = iarg
ks = karg
as = xarg
```

Wertzuweisungen entsprechend der *rate*.

init

```
ks      init      iarg
as      init      iarg
```

Initialisierung von k- und a-Variablen zu Beginn der Performance.

tival

```
ir      tival
```

Weist der i-Variablen der Wert des internen „Legato“-Flags zu.
Der Wert ist 1, wenn die aktuelle Note an eine gehaltene Note (`p3` negativ oder **`ihold`**) gebunden wird. Der Wert ist 0, wenn die vorhergehende Note nicht gehalten wurde.

pset

```
        pset      const1, const2, const3, const4, ...
```

Initialisierung der pfields eines Instruments. Diese Preset-Werte werden nur für MIDI-getriggerte Events benutzt. In allen anderen Fällen gelten die pfield-Werte aus dem Scorefile.

BEDINGTE WERTZUWEISUNGEN

(> ? :)

```
(a > b   ? v1 : v2)
(a < b   ? v1 : v2)
(a >= b  ? v1 : v2)          (größer-gleich)
(a <= b  ? v1 : v2)          (kleiner-gleich)
(a == b  ? v1 : v2)          (gleich)
(a != b  ? v1 : v2)          (ungleich)
```

a,b,v1 und v2 können Ausdrücke sein, a und b dürfen keine *a-rate* Werte sein.

Zuerst werden a und b verglichen. Ist die Vergleichsrelation wahr (*true*) nimmt der Ausdruck den Wert v1 an. Ist die Relation unwahr (*false*), erhält der Ausdruck den Wert von v2.

Beispiele: √

```
ifreq = (p4 > sr/2 ? sr/2 : p4)
```

Der Variablen `ifreq` wird `p4` nur dann zugewiesen, wenn `p4` maximal die halbe Samplingfrequenz ist.

```
asig      oscil       20000,(p4 > sr/2 ? sr/2 : p4),1
```

Die Frequenz des Oszillators kann maximal `sr/2` sein, um Spiegelfrequenzen zu vermeiden.

FUNKTIONEN

int(x)	*(init-, control - rate)*	**int()**
frac(x)	*(init-, control - rate)*	**frac()**
i(x)	*(control - rate)*	**i()**
abs(x)	*(all rates)*	**abs()**
ampdb(x)	*(all rates)*	**exp()**
dbamp(x)	*(init-, control - rate)*	**log() log10()**
exp(x)	*(all rates)*	**sqrt()**
log(x), **log10**(x)	*(all rates)*	**sin() cos() tan()**
sqrt(x)	*(all rates)*	**sinh() ...**
sin(x), **cos**(x), **tan**(x)	*(all rates)*	**sininv() ...**
sinh(x), **cosh**(x),**tanh**(x)	*(all rates)*	**ampdb()**
sininv(x), **cosinv**(x),**taninv**(x)	*(all rates)*	**dbamp()**
rnd(x), **birnd**(x)	*(init-, control - rate)*	**rnd() birnd()**
ftlen(x)	*(init-rate)*	**ftlen()**
ftlptim(x)	*(init-rate)*	**ftlptim()**
ftsr(x)	*(init-rate)*	**ftsr() nsamp()**
nsamp(x)	*(init-rate)*	

x kann ein beliebiger Ausdruck sein.

i(x) konvertiert einen Kontrollwert (k-rate) in einen Initialwert (i-rate).

int(x) ermittelt den ganzzahligen Teil von x.

frac(x) ermittelt den gebrochenen Teil von x.
abs(x) ermittelt den Absolutwert von x.

dbamp(x) ermittelt den dB-Wert der linearen Amplitude x.
ampdb(x) ermittelt die lineare Amplitude des Pegels x.
(60 dB = 1000, 66 dB = 2000, 90 dB = 32000)

exp(x) ermittelt e^x .

log(x) ermittelt den natürlichen Logarithmus (Basis *e*) von x (x darf nur positiv sein).
log10(x) ermittelt den dekadischen Logarithmus (Basisi 10) von x (x darf nur positiv sein).

sqrt(x) ermittelt die Quadratwurzel von x (x darf nicht negativ sein).

sin(x), **cos**(x), **tan**(x) Winkelfunktionen von x (x im Bogenmaß).
sininv(x), **cosinv**(x), **taninv**(x) inverse Winkelfunktionen (arcus) von x.
sinh(x), **cosh**(x), **tanh**(x) hyperbolische Winkelfunktionen von x.

ftlen(x) ermittelt die Größe der Tabelle x in Punkten (Samples).
ftlptim(x) gibt die Loop-Startzeit eines Tabellensamples x in Sekunden zurück.
ftsr(x) ermittelt die Samplingrate eines Samples in der Tabelle x (`GEN01` oder `GEN22`).
nsamp(x) gibt die Anzahl der Samples in der Tabelle x zurück. Nützlich für Tabellengrößen ungleich einer 2er-Potenz (`GEN01` oder `GEN22`).

rnd(x) erzeugt einen Zufallswert zwischen 0 und x.
birnd(x) erzeugt einen Zufallswert zwischen -x und x.

POTENZFUNKTIONS-GENERATOREN

pow berechnet die Potenz mit dem Exponenten `kpow` zur Basis *`x`*`arg.`

`is`	**`pow`**	`iarg,kpow`	**pow**
`ks`	**`pow`**	`karg,kpow,[inorm]`	**pow**
`as`	**`pow`**	`aarg,kpow,[inorm]`	**pow**

`iarg,karg,aarg`
- Basis der Potenzfunktion

`kpow` - Exponent

`inorm` - Normalisierungswert, d.h. Wert, durch den das Ergebnis dividiert wird (optional, default 1)

Beispiel: √

```
k1      line    0,1,4
k2      pow     k1,3,16
```

Potenziert das Signal k1 mit 3 und normalisiert anschließend. Ergebnis sind Werte zwischen 0 und 4 mit kubischem Anstieg.

TONHÖHEN-FREQUENZ-KONVERTER

`octpch``(pch)`	*(init-, control - rate)*	**octpch()**
`pchoct``(oct)`	*(init-, control - rate)*	**pchoct()**
`cpspch``(pch)`	*(init-, control - rate)*	**cpspch()**
`octcps``(cps)`	*(init-, control - rate)*	**octcps()**
`cpsoct``(oct)`	*(all rates)*	**cpsoct()**

Diese Konverter manipulieren Tonhöhen- und Frequenzwerte.
Die Werte können folgende Formen annehmen:

`pch` - Oktave . Tonigkeit (pitch-class)
`oct` - Oktave . Dezimalwert
`cps` - Frequenz in Hz (cycles per second)

Die Angabe der Oktave erfolgt bei den ersten beiden Formen durch eine ganze Zahl,
8.00 bedeutet c', 9.00 bedeutet c'' usw.
Die Tonigkeit wird in Form einer zweistelligen Zahl 00 bis 11 (oder auch größer) angegeben und repräsentiert die Halbtonschritte über c in einer 12-temperierten Skala.
(8.09 = a', 8.21 = a'', 7.07 = a'').
Der Dezimalwert bezeichnet einen Bruchteil (bezogen auf eine 12-temperierte Teilung) einer Oktave. (8.5 = f#', 8.75 = a').
Die Konverter wandeln entsprechend ihrer Bezeichnung um: das zweite Morphem benennt das Quellformat, das erste Morphem benennt das Zielformat.

Beispiele: √

`cpspch``(8.09)` ergibt `440`
`octcps``(440)` ergibt `8.75`
`cpsoct``(8.75)` ergibt `440`

PROGRAMMSTEUERUNG

Die folgenden Anweisungen werden verwendet, um die Abarbeitungsreihenfolge innerhalb eines Instrumentenblocks steuern.

igoto	label	**igoto**
tigoto	label	**tigoto**
kgoto	label	**kgoto**
goto	label	**goto**

Die **goto**-Anweisungen erzwingen einen *unbedingten* Sprung zu der Programmzeile, die mit label eingeleitet wird. Der Sprung erfolgt entweder nur bei der Initialisierung (*i-time*) mit **igoto**, zur *k-time* mit **kgoto** oder beides (**goto**). **tigoto** springt nur dann zum Label, wenn die aktuelle Note an eine gehaltene Note „gebunden“ wird („Legato-Flag“).

if	ia R ib **igoto** label	**if igoto**
if	ka R kb **kgoto** label	**if kgoto**
if	ia R ib **goto** label	**if goto**

Die **if-goto**-Kombination erzeugt einen *bedingten* Sprung zum label. Immer nur dann, wenn zur *i-time* (**if-igoto**), zur *k-time* (**if-kgoto**) oder beides (**if-goto**), die Relation ia R ib bzw. ka R kb logisch wahr ist, erfolgt ein Sprung.

timout	istrt, idur, label	**timout**

Mit **timout** ist ein bedingter Sprung in Abhängigkeit der vergangenen Notenzeit möglich. Ein Sprung zum label erfolgt ab der Zeit istrt und gilt für idur Sekunden.

reinit	label	**reinit**
rigoto	label	**rigoto**
rireturn		**rireturn**

Diese Anweisungen erlauben es, ein Instrument zur *perfomance-time*, also während einer Note, zu re-initialisieren.

reinit - immer wenn diese Anweisung erreicht wird, wird die aktuelle Performance der Note unterbrochen, um eine spezielle Re-Initialisierung zu durchlaufen. Diese beginnt bei label und wird durch **rireturn** oder **endin** beendet.

rireturn - beendet einen Re-Initialisierungsdurchlauf. Diese Anweisung oder **endin** bewirkt die Wiederaufnahme der normalen Performance.

rigoto - ähnlich wie **igoto**, allerdings nur während einer Re-Initialisierung gültig. Diese Anweisung kann verwendet werden, um Programmabschnitte zu umgehen, die nicht neu initialisiert werden sollen.

DAUERNSTEUERUNG

	ihold	**ihold**
	turnoff	**turnoff**
	turnon insno [, itime]	**turnon**

Diese Anweisungen erlauben dem Instrument, die Dauer seiner Noten zu verändern.

ihold - Diese *i-time* - Anweisung macht aus einer Note mit endlicher Dauer (p3 ist positiv) eine gehaltene Note. Die Note wird solange gehalten, bis eine neue Note mit dem gleichen Instrument gespielt wird oder bis eine **turnoff**-Anweisung wirksam wird.

turnoff - Diese *p-time* - Anweisung erlaubt dem Instrument, sich selbst auszuschalten. Dies gilt für gehaltene Noten ebenso wie für Noten normaler Dauer.

turnon - Diese *p-time* - Anweisung erlaubt dem Instrument, nach einer Zeit itime ein anderes Instrument insno aufzurufen.

ZEITMESSUNG

ks	**timek**	**timek**
ks	**times**	**times**
ir	**itimek**	**instimek**
ir	**itimes**	**instimes**
ks	**instimek**	
ks	**instimes**	

timek gibt die Zeit seit Beginn der Performance in *k-rate*-Zyklen an.
times gibt die Zeit seit Beginn der Performance in Sekunden an.
instimes und **instimek** gibt die Zeit seit Beginn der Note für dieses Instrument an.

MIDI-KONVERTER

ival	**notnum**		**notnum**
ival	**veloc**		**veloc**
icps	**cpsmidi**		**cpsmidi**
icps	**cpsmidib**		**cpsmidib**
kcps	**cpsmidib**		
ioct	**octmidi**		**octmidi**
ioct	**octmidib**		**octmidib**
koct	**octmidib**		
ipch	**pchmidi**		**pchmidi**
ipch	**pchmidib**		**pchmidib**
kpch	**pchmidib**		
iamp	**ampmidi**	iscal [,ifn]	**ampmidi**
kaft	**aftouch**	iscal	**aftouch**
kchpr	**chpress**	iscal	**chpress**
kbend	**pchbend**	iscal	**pchbend**
ival	**midictrl**	inum	**midictrl**
kval	**midictrl**	inum	

Die MIDI-Konverter übergeben die Werte des aktuellen MIDI-Events an Variablen des Instruments.

`iscal` - *i-time* - Skalierungsfaktor
`inum` - Nummer des MIDI-Controllers

`notnum, veloc`
- übergibt das MIDI-Byte (0...127) für *note number* oder *velocity* des aktuellen Events.

`cpsmidi, octmidi, pchmidi`
- übergibt *note number* im **`cps`**-, **`oct`**- oder **`pch`**-Format (siehe Tonhöhen-Konverter).

`cpsmidib, octmidib, pchmidib`
- übergibt *note number,* modifiziert durch aktuelles *pitch bend*, im **`cps`**-, **`oct`**- oder **`pch`**-Format. Verfügbar zur *i-time* oder *k-time* (kontinuierlich).

`ampmidi` - übergibt die Anschlagsstärke des aktuellen Events, optional konvertiert durch eine Tabelle `ifn`, als Amplitudenwert zwischen 0 und `iscal`.

`aftouch, chpress, pchbend`
- übergibt *after touch, channel pressure* oder *pitch bend* im Bereich 0 bis `iscal` .

`midictrl` - übergibt den aktuellen Wert (0...127) des angebenen Controllers `inum`.

SIGNALGENERATOREN

line, expon ... erzeugen Werte entsprechend einer gerade Linie (exponentiellen Kurve) bzw. einer aus linearen (exponentiellen) Segmenten zusammengesetzten Kurve.

```
  ks      line      ia, idur, ib                              line
  as      line      ia, idur, ib
  ks      expon     ia, idur, ib                              expon
  as      expon     ia, idur, ib

  ks      linseg    ia, idur1, ib [, idur2, ic [...]]         linseg
  as      linseg    ia, idur1, ib [, idur2, ic [...]]
  ks      expseg    ia, idur1, ib [, idur2, ic [...]]         expseg
  as      expseg    ia, idur1, ib [, idur2, ic [...]]
```

`ia` - Startwert, für Exponentialkurven ≠ 0 !
`ib,ic,...` - Werte nach jeweils `idur` Sekunden,
für exponentiale Kurven sind negative Wert und 0 nicht erlaubt.
`idur` - Dauer der Segmente in Sekunden

Beispiele: √

```
  k1        expon       1,p3,0.01
```

Erzeugt einen exponentiell abfallenden Verlauf von 1.0 bis 0.01 über die Notendauer.

```
  k2        linseg      0,0.2,1000,p3-1,-1000,0.8,0
```

Erzeugt einen linearen Anstieg von 0 in 200 ms auf 1000, danach in p3-1 Sekunden auf -1000, dann wieder auf 0 in 800 ms.

adsr und **xadsr** erzeugen eine klassische 4-Segment-Hüllkurve mit Werten von 0 ...1. **adsr** hat lineare Segmente, während **xadsr** exponentielle Kurven generiert. Die Gesamtzeit der Hüllkurve entspricht p3.

```
  ks      adsr      iatt, idec, islev, irel [, idelay]        adsr
  as      adsr      iatt, idec, islev, irel [, idelay]
  ks      xadsr     iatt, idec, islev, irel [, idelay]        xadsr
  as      xadsr     iatt, idec, islev, irel [, idelay]
```

`iatt` - Dauer der Attackphase
`idec` - Dauer der Decayphase
`islev` - Pegel (0...1) der Sustainphase
`irel` - Dauer der Releasephase
`idelay` - Verzögerungszeit für die gesamte Hüllkurve (default 0)

phasor produziert zyklisch ansteigende Phasenwerte im Bereich [0,1] . **phasor**

```
  ks      phasor    kcps [, iphs]
  as      phasor    xcps [, iphs]
```

`xcps` - Frequenz der Phasenzyklen
`iphs` - Initialphase (optional, default 0)

TABELLEN UND OSZILLATOREN

`table` erlaubt über Indices den freien Zugriff auf eine, mittels einer im *scorefile* beschriebenen Funktion generierte Tabelle (*table lookup*) zur *i-*, *k-* und *a-rate*.

table

```
ir      table     indx, ifn [, ixmode][, ixoff][, iwrap]
ks      table     kndx, ifn [, ixmode][, ixoff][, iwrap]
as      table     andx, ifn [, ixmode][, ixoff][, iwrap]
```

`xndx` - Index (als Sample-Nummer bzw. normalisiert zwischen 0 und 1)
`ifn` - Nummer der Funktionstabelle
`ixmode` - Indexmodus: 0=Sample-Nummern, 1=normalisiert (optional, default 0)
`ixoff` - Offset (optional, default 0)
`iwrap` - wraparound flag: 0=no wrap (ndx<0 -> 0, ndx>max -> max)
1=wraparound (optional, default 0)

Beispiele: √

```
aindx     phasor      300
asig      table       aindx*2048,1
```

`phasor` generiert 300 Phasenzyklen pro Sekunde. Diese Werte aindx werden, nach Multiplikation mit 2048, als Index auf die Samples der Tabelle 1 benutzt. Als Ergebnis werden in der Variable `asig` die Werte eines mit 300 Hz periodischen Signals und der in Tabelle 1 beschriebenen Wellenform übergeben.
Statt dieser 2 Programmzeilen wird allerdings meistens ein Oszillator `oscil` benutzt.

```
ifreq     table       p4,2
asig      oscil       20000,ifreq,1
```

Der *scorefile* Parameter `p4` wird als Index auf Tabelle 2 benutzt, um eine Frequenz `ifreq` aus dieser Tabelle zu lesen. `ifreq` wird in einen nachfolgenden Oszillator für ein Audiosignal `asig` verwendet.

`oscil` liest periodisch Werte aus einer gespeicherten Funktionstabelle. **oscil**

```
ks      oscil     kamp, kcps, ifn [, iphs]
as      oscil     xamp, xcps, ifn [, iphs]
```

`xamp` - Multiplikator für die gelesenen Tabellenwerte (Amplitude)
`xcps` - Auslesefrequenz in Hz
`ifn` - Nummer der Funktionstabelle
`iphs` - Initialphase (optional, default 0)

Beispiel: √

```
k1        oscil       10,5,1
a1        oscil       5000,440+k1,1
```

Die erste Zeile beschreibt ein 5 Hz-Signal mit der Amplitude 10, d.h. Werten zwischen -10 und +10.
In der zweiten Zeile werden diese Werte `k1` zur Frequenzmodulation verwendet.
Es ergibt sich ein Signal `a1` mit 440 ±10 Hz, also ein a' mit Vibrato.

oscil1 liest die gesamteTabelle `ifn` genau einmal in der Zeit `idur`. Während der Zeit idel wird der 1. Tabellenwert ausgegeben. Nach Durchlaufen der Tabelle wird der letzte Wert beibehalten.

```
ks      oscil1    idel,kamp,idur,ifn                    oscil1
```

`idel` - Verzögerungszeit in Sekunden.
`kamp` - Amplitude
`idur` - Dauer des Tabellendurchlaufs
`ifn` - function table number.

osciln liest Werte aus einer Tabelle `ifn` genau `itimes`-mal mit einer Frequenz `icps`. Nach `itimes/icps` Sekunden wird 0 ausgegeben.

```
asig    osciln    kamp,icps,ifn,itimes                  osciln
```

`icps` - Frequency in Hz.
`ifn` - function table number.
`itimes` - Anzahl der Perioden
`kamp` - Amplitude

Beispiel: √

```
asig      osciln      20000,500,1,10
```

Die Funktionstabelle `f1` wird 10 mal abgetastet mit einer Frequenz von 500 Hz.
`asig` erhält nach 0.02 Sekunden den Wert 0.

foscil besteht aus zwei Oszillatoren in FM-Konfiguration: der Output des einen Generators ("Modulator") moduliert die Frequenz des zweiten Generators ("Träger").

```
as      foscil    kamp, kcps, kcar, kmod, kndx, ifn [, iphs]    foscil
```

`xamp` - Multiplikator für die gelesenen Tabellenwerte (Amplitude)
`kcps` - Grundfrequenz in Hz
`kcar` - Faktor für Carrier (Träger)
`kmod` - Faktor für Modulator
`kndx` - Modulationsindex >= 0
`ifn` - Nummer der Funktionstabelle (für reine FM > Sinuswellenform erforderlich!)
`iphs` - Initialphase (optional, default 0)

Beispiel: √

```
ki        expon       1,p3,0.01
asig      foscil      30000*ki,p4,1,1.41,ki*5,1
```

Das exponentiell abfallende Signal `ki` wird bei `foscil` als Amplitudenhüllkurve und als dynamischer Modulationsindex verwendet.
Das Verhältnis mod:car ist inharmonisch (1.41:1.0), die Grundfrequenz kommt aus dem *scorefile*.
Ergebnis ist ein percussiver Klang.

table, oscil, oscil1 und **foscil** sind auch als linear interpolierende Varianten **tablei, oscili, oscil1i, foscili** und z.T. auch mit kubischer Interpolation **table3, oscil3** verfügbar (dabei erhöht sich die Rechenzeit!).

gbuzz produzierte eine aus Harmonischen (Cosinusfunktionen) additive aufgebaute Wellenform. Anzahl, relative Amplitude und erster vorkommender Teilton sind frei bestimmbar.
buzz ist eine spezielle Variante von **gbuzz**, in der alle Harmonischen die relative Amplitude 1 haben. Der erste Teilton entspricht `xcps`.

```
  as      buzz      xamp, xcps, knh, ifn [, iphs]                    buzz
  as      gbuzz     xamp, xcps, knh, klh, ka, ifn [, iphs]           gbuzz
```

`xamp` - Amplitude
`xcps` - Grundfrequenz in Hz
`knh` - Anzahl der Harmonischen
`klh` - tiefste vorkommende Harmonische
`ka` - Faktor für die Amplituden der Harmonischen $A_n = A \cdot ka^n$
`ifn` - Nummer der Tabelle mit Sinuswellenform für **buzz**
bzw. Cosinuswellenform für **gbuzz**
`iphs` - Initialphase (optional, default 0)

Beispiel: √

```
  k1        expon       1,p3,0.1
  asig      gbuzz       20000,k1*1000,20,1,1-k1,1
```

Ein Glissando von 1 kHz bis 100 Hz, wobei die Amplituden der 20 Teiltöne mit fallender Grundfrequenz immer stärker werden.

vco ist ein bandbegrenzter "analoger" Oszillator mit verschiedenen klassischen Wellenformen.

```
  as      vco       kamp, kcps, iwave, kpw, ifn, imaxd               vco
```

`kamp` - Amplitude
`kcps` - Grundfrequenz in Hz
`iwave` - Wellenformnummer: 1=Sägezahn, 2=Rechteck/Pulswelle, 3=Dreick/Sägezahn
`kpw` - Pulsbreite bzw. Dreieck-Sägezahncharakter in % (0...1)
`ifn` - Nummer der Tabelle mit Sinuswellenform
`imaxd` - maximale Verzögerungszeit (muß 1 / tiefste Frequenz sein!)

lfo ist ein Oszillator (für Modulationen) mit verschiedenen Standardwellenformen.

```
  ks      lfo       kamp, kcps [, itype]                             lfo
  as      lfo       kamp, kcps [, itype]
```

`kamp` - Amplitude
`kcps` - Frequenz in Hz
`itype` - Wellenformnummer: (optional, default 0)
0 = Sinus
1 = Dreieck
2 = Rechteck, bipolar (positiv / negativ)
3 = Rechteck, unipolar (nur positiv)
4 = Sägezahn, steigend
5 = Sägezahn fallend.

loscil liest ein Soundsample (mono oder stereo) aus einer Tabelle (GEN01). Die Bezugsfrequenz (Originaltonhöhe) ist ibas. kcps ist die darauf bezogene Ausleserate.

```
as[,as2] loscil    xamp,kcps,ifn[,ibas][,imod1,ibeg1,iend1]
                   [,imod2,ibeg2,iend2]
```

loscil

xamp - Multiplikator für die gelesenen Tabellenwerte (Amplitude)
kcps - Wiedergabefrequenz in Hz
ifn - Nummer der Funktionstabelle
ibas - (optional) Originalfrequenz des Soundsamples.
Der Wert 0 liest die Frequenz aus dem Soundfile (AIFF), falls vorhanden.
imod1 - Play-Modus für Sustain-Loop: 1 = vorwärts, 2 = vorwärts-rückwärts, 0 = ohne,
(-1 liest Modus aus dem Soundfile).
ibeg1,
iend1 - Beginn- und Endpunkt (in Samples) für die Sustain-Loop.
imod2,ibeg2,
iend2 - Modus, Beginn und Ende der Release-Loop.
Die Releasephase wird von einem Notenende im Scorefile ausgelöst.

Beispiele: √

```
a1         loscil       30000,1.5,3,1.0,0
```

Das Soundsample in Tabelle 3 wird im Verhältnis 1.5:1 ohne Loop ausgelesen. Die Tabellenwerte werden mit 30000 multipliziert.

```
ali,are    loscil       1,cpspch(8.09),4,cpspch(8.00),-1
```

Ein Stereosample in Tabelle 4, dessen Originaltonhöhe c' ist, wird als a' abgespielt.
Loop-Punkte und Loop-Modus entsprechen denen des Soundfiles.

lposcil ist eine Variante von loscil, die eine dynamische Veränderung der Loop-Punkte erlaubt.

```
as       lposcil    kamp, kfreqratio, kloop, kend, ifn [,iphs]
```

lposcil

kamp - Multiplikator für die gelesenen Tabellenwerte (Amplitude)
kfreqration - Wiedergabegeschwindigkeit (1 = original, 0.5 = halbe Geschwindigkeit etc.)
kloop - Startpunkt des Loops (in Samples)
kend - Endpunkt des Loops (in Samples)
ifn - Nummer der Funktionstabelle
iphs - Initialphase (in Samples), optional

ZUFALLSGENERATOREN

Die folgenden Generatoren stehen in Form einer Funktion zur Verfügung.

rnd(x)	*(init-, control - rate)*	**rnd()**
birnd(x)	*(init-, control - rate)*	**birnd()**

Wenn x einen *i-rate* - Wert hat, liefern die Funktionen einen *i-rate* - Zufallswert.
Wenn x einen *k-rate* - Wert hat, liefern die Funktionen *k-rate* - Zufallswerte.

rnd(x) liefert Werte zwischen (incl.) 0 und (excl.) x.
birnd(x) liefert Werte zwischen (excl.) -x und (excl.) x.

Beispiel: √

```
a1        oscil       20000,p4,1,rnd(1)
a2        oscil       20000,p4*(1+birnd(0.01)),1,rnd(1)
a3        oscil       20000,p4*(1+birnd(0.01)),1,rnd(1)
asum      =           a1+a2+a3
```

Bei jedem Aufruf des Instruments werden andere Zufallsphasen und geringe Frequenzabweichungen wirksam.

Mit **seed** kann der interne Pseudo-Zufallsgenerator für jedes Instrument separat initialisiert werden.

```
          seed      ival                                    seed
```

rand produziert mit *k*- bzw. *a-rate* einen gleichverteilten Zufallswert zwischen -xamp und +xamp (weißes Rauschen).
randh produziert nur mit xcps neue Werte die für die Dauer einer Periode gehalten werden (bandbegrenztes Rauschen). **randi** ist eine interpolierende Variante von **randh**..

```
ks        rand      xamp [, iseed]                          rand
ks        randh     kamp, kcps [, iseed]                    ranhd
ks        randi     kamp, kcps [, iseed]                    randi
as        rand      xamp [, iseed]
as        randh     xamp, xcps [, iseed]
as        randi     xamp, xcps [, iseed]
```

xamp - Maximalwert
xcps - Rate für die Generierung neuer Werte
iseed - Initialwert zwischen 0 und 1 (optional, default 0.5),
negative Zahlen verhindern gleiche Zufallssequenzen bei jeder neuen Note.

Beispiele: √

```
anoise    rand        32767
```

Produziert weißes Rauschen mit Maximalamplitude.

```
k1        randh       10,0.5,-0.5
k2        randh       1,abs(k1)
```

Alle 2 Sekunden ändert sich die Rate des 2. Zufallsgenerators auf einen zufälligen Wert zwischen 0 und 10.
k2 enthält Zufallswerte zwischen -1 und +1.

Die folgenden Generatoren liefern Zufallswerte, die einer bestimmten Wahrscheinlichkeitsverteilung entsprechen, während **`rand`**, **`randh`** und **`randi`** immer gleichverteilte Werte erzeugen.
Sie arbeiten sowohl mit *i-*, *k-* und *a-rate.*

xs	**unirand**	krange	**unirand**
xs	**linrand**	krange	**linrand**
xs	**trirand**	krange	**trirand**
xs	**exprand**	krange	**exprand**
xs	**bexprnd**	krange	**bexprnd**
xs	**cauchy**	kalpha	**cauchy**
xs	**pcauchy**	kalpha	**pcauchy**
xs	**poisson**	klambda	**poisson**
xs	**gauss**	krange	**gauss**
xs	**weibull**	ksigma, ktau	**weibull**
xs	**betarand**	krange, kalpha, kbeta	**betarand**

`unirand` - Gleichverteilung im Bereich [0 ... `krange`].
`linrand` - Linearverteilung im Bereich [0 ... `krange`].
`trirand` - Dreiecksverteilung im Bereich [`-krange` ... `+krange`].
`exprand` - Exponentialverteilung im Bereich [0 - `krange`].
`bexprnd` - bilaterale Exponentialverteilung im Bereich [`-krange` ... `+krange`].
`cauchy` - Cauchy-Verteilung, `kalpha` steuert die Bandbreite
im positiven und negativen Bereich.
`pcauchy` - Cauchy-Verteilung, wie **`cauchy`**, aber nur positive Werte
`poisson` - Poisson-Verteilung, `klambda` ist der Mittelwert, nur positive Werte.
`gauss` - Gauss-Verteilung im Bereich [`-krange` ... `+krange`].
`weibull` - Weibull-Verteilung. `ksigma` steuert die Bandbreite,
`ktau` bewirkt, falls größer als 1, eine Häufung in der Nähe von `ksigma`,
falls kleiner als 1, eine Bevorzugung kleinerer Werte und
falls gleich 1, eine Exponentialverteilung
`betarand` - Betaverteilung im Bereich [0 ... `krange`].
Wenn `kalpha` kleiner als 1, werden Werte nahe 0 bervorzugt,
wenn `kbeta` kleiner als 1, werden Werte nahe `krange` bevorzugt,
wenn `kalpha` und `kbeta` gleich 1, entsprechen die Werte
einer Gleichverteilung,
sind sie größer als 1, werden die Werte ähnlich einer Gaussverteilung sein.

Beispiele: √

```
a1        trirand       32000         ; Rauschen mit Dreiecksverteilung
k1        exprand       1000          ; k-rate Rauschen mit Exponentialverteilung
i1        betarand      200,.1,.1     ; i-time Zufallswert, Häufung nahe 0 und nahe 200
```

HÜLLKURVEN

linen bewirkt einen linearen Anstieg (*rise*), eine stationäre Phase (idur-irise-idec) und einen linearen Abfall (*decay*),
linenr bewirkt einen linearen Anstieg, eine stationäre Phase bis zum Ende der Note und danach einen exponentiellen Abfall , wobei die Note um die Decay-Zeit verlängert wird (nur bei MIDI).
envlpx stellt eine Hüllkurve aus 3 Segmenten dar: 1) Anstiegscharakteristik entsprechend einer Tabelle, 2) exponentialer quasi-stationärer Zustand, 3) exponentieller Abfall.

```
ks      linen     ksig,irise,idur,idec                          linen
as      linen     xsig,irise,idur,idec
ks      linenr    ksig,irise,idec,iatdec                        linenr
as      linenr    xsig,irise,idec,iatdec
ks      envlpx    ksig,irise,idur,idec,ifn,iatss,iatdec         envlpx
                  [,ixmod]
as      envlpx    xsig,irise,idur,idec,ifn,iatss,iatdec
                  [,ixmod]
```

xsig - Eingangssignal (= Amplitude)
irise - Anstiegszeit in Sekunden
idur - Gesamtdauer in Sekunden
idec - Decay-Zeit in Sekunden
ifn - Tabelle mit Anstiegskurvenform
iatss - Dämpfungsfaktor während des quasi-stationären Zustands,
> 1 bedeutet exponentielle Zunahme, <1 exponentielle Abnahme,
= 1 stabiler Zustand
iatdec - Amplitudenfaktor, der am Ende der Decay-Zeit erreicht werden soll
(meist bei 0.01)
ixmod - Modifikator für iatss, > 0 bewirkt langsamere Zu- oder Abnahme,
< 0 bewirkt schnelle Zu- oder Abnahme (optional, default 0)

Beispiele: √

```
aout      linen      asig,0.1,p3,0.5
```

Das Signal asig bekommt ein Attack von 100 ms und am Ende der Notendauer ein Decay von 0.5 Sekunden.

follow ist ein *envelope follower.*

```
as      follow    asig,idt                                      follow
```

idt - Zeitabstand in Sekunden, der bestimmt, wann der Durchschnittswert der Amplitude von asig ausgegeben wird. idt sollte immer mindestens so groß wie die längste Periodendauer (= tiefste Frequenz) in asig sein.

Die Ausgabe wird zwischen den im Abstand idt gemessenen Werten *nicht* interpoliert, d.h. man erhält in der Regel ein stufiges Signal.

Beispiel: √

```
asig      soundin     „drums“
ah        follow      asig,0.01
ahf       tone        ah,20
a1        oscil       ahf,p4,1
```

Die Hüllkurve eines Soundfiles wird durch ein Tiefpaßfilter geglättet und als Amplitude für einen Oszillator verwendet.

FILTER

port glättet stufige *k-rate*-Signale durch Tiefpaßfilterung. Die Zeitkonstante `ihtim` entspricht dem Kehrwert der Eckfrequenz des Filters.

```
  ks        port        ksig, ihtim [, isig]                    port
```

`ihtim` - Zeitkonstante
`isig` - Initialwert für den internen Zwischenspeicher (optional, default 0)

Beispiel: √

```
  kx        randh       20,2
  kgx       port        kx,0.1
```

`kx` ist ein Zufallssignal im Bereich zwischen -20 und +20, das seinen Wert 2 mal in der Sekunde ändert. port „interpoliert“ zwischen aufeinanderfolgenden Werten durch Integration. Die Zeitkonstante ist 100ms, das entspricht einer Grenzfrequenz von 10 Hz.

tone ist ein rekursives Tiefpaßfilter 1. Ordnung, **atone** der entsprechende Hochpaß.

```
  as        tone        asig, kfreq [, istor]                   tone
  as        atone       asig, kfreq [, istor]                   atone
```

`kfreq` - Grenzfrequenz in Hz
`istor` - Speicherstatus für Filterfeedback, 0 = löschen, 1 = nicht löschen (optional, default 0)

reson ist ein Bandpaß 2. Ordnung, **areson** die entsprechende Bandsperre.

```
  as        reson       asig, kcf, kbw [, iscl, istor]          reson
  as        areson      asig, kcf, kbw [, iscl, istor]          areson
```

`kcf` - Mittenfrequenz in Hz
`kbw` - Bandbreite in Hz
`iscl` - Skalierungsfaktor 1=Spitzenwert 1, 2=Mittelwert 1, 0=keine Skalierung
`istor` - Speicherstatus für Filterfeedback, 0 = löschen, 1 = nicht löschen (optional, default 0)

butterlp, **butterhp**, **butterbp** und **butterbr** sind Butterworthfilter 2. Ordnung: Tiefpaß, Hochpaß, Bandpaß, Bandsperre. Diese Filter haben eine größere Steilheit sowie einen ebeneren Durchlaßbereich als **tone**, **atone**, **reson** und **areson**, kosten aber etwas mehr Rechenzeit.

```
  as        butterlp    asig, kfreq                             butterlp
  as        butterhp    asig, kfreq                             butterhp
  as        butterbp    asig, kfreq,kband                       butterbp
  as        butterbr    asig, kfreq,kband                       butterbr
```

`kfreq` - Grenzfrequenz bzw. Mittenfrequenz in Hz
`kband` - Bandbreite von Bandpaß bzw. Bandsperre in Hz

REGELVERSTÄRKER

rms ermittelt den quadratischen Mittelwert (*root mean square*) des Audiosignals asig.

```
  ks        rms         asig [,ihp, istor]                    rms
```

ihp - 3 dB-Grenzfrequenz des internen Tiefpaßfilters in Hz (optional, default 10 Hz).
istor - Speicherstatus für Filterfeedback, 0 = löschen, 1 = nicht löschen (optional, default 0)

gain modifiziert die Amplitude des Audiosignals asig so, daß dessen Mittelwert dem Wert von krms entspricht.

```
  as        gain        asig, krms [,ihp, istor]              gain
```

krms - Steuersignal für die Amplitude
ihp - 3 dB-Grenzfrequenz des internen Tiefpaßfilters in Hz (optional, default 10 Hz).
istor - Speicherstatus für Filterfeedback, 0 = löschen, 1 = nicht löschen (optional, default 0)

balance modifizert die Amplitude von asig, so daß dessen quadratischer Mittelwert gleich dem des Vergleichssignals acomp ist. Sollte nach der Verwendung von Filtern benutzt werden, um Signalverluste (**tone, butterbp** etc.) bzw. starke Überhöhungen (**reson**) auszugleichen. **balance** entspricht einer Kombination von **rms** und **gain** mit gleichem ihp.

```
  as        balance     asig,acomp [,ihp,istor]               balance
```

asig - Eingangssignal
acomp - Vergleichssignal
ihp - (optional) Grenzfrequenz eines internen Tiefpasses zur Mittelwertbildung (default 10)
istor - Speicherstatus für Filterfeedback, 0 = löschen, 1 = nicht löschen (optional, default 0)

dam ist ein Dynamikprozessor. Die Dynamik des Signals asig wird abhängig von den Kompressionsgraden icomp oberhalb bzw. unterhalb eines Schwellwerts verändert.

```
  as        dam         asig,ktresh,icomp1, icomp2, rtime, ftime   dam
```

asig - Eingangssignal
ktresh - Vergleichssignal
icomp1 - Kompressionsgrad oberhalb des Schwellwerts (1 = keine Kompression, < 1 = Kompression, >1 = Expansion)
icomp2 - Kompressionsgrad unterhalb des Schwellwerts
rtime - Einregelzeit des Verstärkers (sollte im ms-Bereich liegen)
ftime - Abklingzeit des Verstärkers (in der Regel viel größer als rtime)

DELAY UND REVERB

delay stellt einen Verzögerungsweg dar. **delay1** verzögert um 1 Sample (1/**sr**).

```
  as       delay      asig, idlt [,istor]                    delay
  as       delay1     asig [,istor]                          delay1
```

```
asig     - Eingangssignal
idlt     - Verzögerungszeit in Sekunden
istor    - Speicherstatus für Delay-Loop, 0 = löschen, 1 = nicht löschen (optional, default 0)
```

Mit den folgenden Modulen kann eine variable Delay-Line aufgebaut werden:

```
  as       delayr     idlt [,istor]                          delayr
  as       deltap     kdlt                                   deltap
  as       deltapi    xdlt                                   deltapi
           delayw     asig                                   delayw
```

```
asig     - Eingangssignal
idlt     - maximale Verzögerungszeit in Sekunden, minimal 1/kr !
kdlt,xdlt- variable Verzögerungszeit in Sekunden, maximal idlt, minimal 1/kr !
istor    - Speicherstatus für Delay-Loop, 0 = löschen, 1 = nicht löschen (optional, default 0)
```

Mit **delayr** (read) und **delayw** (write) wird ein Verzögerungsweg aufgebaut. Beide bilden immer ein Paar und müssen in dieser Reihenfolge geschrieben werden. Zwischen **delayr** und **delayw** können beliebige andere Statements stehen, insbesondere **deltap** bzw. **deltapi**. (**delay** ist eine Kombination aus **delayr** und **delayw**.)
Mit **deltap** oder **deltapi** (interpolierend) kann das Signal in der Delay-Line abgegriffen (*tap*) werden, wobei die Verzögerungszeit variabel ist. Diese Verzögerungszeit kann dabei zwischen einem Kontroll-Sample 1/**kr** und der Gesamtzeit der Delay-Line idlt variieren. Zwischen einem read/write-Paar könne beliebig viele *taps* eingerichtet werden.

vdelay ist ein interpolierendes variables Delay und arbeitet wie eine Kombination aus **delayr**, **deltapi** und **delayw**.

```
  as       vdelay     asig, adel, imaxdel                    vdelay
```

```
asig     - Eingangssignal
adel     - Delay in *Samples* (nicht in Sekunden !)
imaxdel  - Maximalwert des Delays in *Samples*
```

multitap kombiniert mehrere tap-delays.

```
  as       multitap   asig, itime1, igain1 [, itime2, igain2, ...]    multitap
```

```
asig     - Eingangssignal
itime    - Verzögerungszeit in Sekunden
igain    - relative Amplitude
```

reverb ist ein monophoner Hall, bestehend aus 4 parallelen Kammfiltern und 2 Allpässen in Reihe.

```
  as       reverb     asig, ktime [, istor]                    reverb
```

asig - Eingangssignal
ktime - *reverb time* in Sekunden
istor - Speicherstatus für interne Delay-Loop, 0 = löschen, 1 = nicht löschen (optional, default 0)

reverb2 ist ein aufwendigerer monophoner Hall aus 6 parallelen Kammfiltern mit 5 Allpässen in Reihe.

```
  as       reverb2    asig, ktime, khdif                       reverb2
```

asig - Eingangssignal
ktime - *reverb time* in Sekunden
khdif - regelt die relative Abklingzeit der höheren Frequenzen. (0 ... 1)
=0 : alle Frequenzen klingen gleichmäßig ab
> 0 : höhere Frequenzen klingen schneller als tiefere aus

comb und **alpass** sind geloopte Delays bzw. Echos.
Ein Kammfilter (**comb**) „färbt“ das Signal, indem es Vielfache des Kehrwerts von ilpt, der Looptime, im Spektrum betont und die dazwischen liegenden Frequenzen absenkt.
Ein Allpaß hat einen ebenen Frequenzgang, ändert aber die Phasen des gesamten Spektrums.

```
  as       comb       asig, ktime,ilpt [, istor]               comb
  as       alpass     asig, ktime,ilpt [, istor]               alpass
```

asig - Eingangssignal
ktime - Abklingzeit (Feedback) in Sekunden
ilpt - Looptime des Delays in Sekunden

SIGNAL INPUT & OUTPUT

soundin liest ein mono-, stereo- oder quadrophones Soundfile von der Harddisk.

```
    a1[,a2[,a3,a4]]  soundin    ifilcod [,iskptim][,iformat]          soundin
```

a1,a2... - Audiovariablen
ifilcod - Name oder Nummer des Soundfiles; bei Angabe einer Nummner *n* muß das Soundfile *soundin.n* benannt worden sein.
iskptim - Anfangszeit (optional, default 0)
iformat - Einlese-Format für das Soundfile : 1=8bit, 4=16bit, 5=32bit, 6=Fließkomma, 0=Original-Format des Soundfiles (optional, default 0)

diskin liest ein Soundfile von der Harddisk und transponiert es durch langsameres oder schnelleres Lesen.

```
    a1[,a2[,a3,a4]]  diskin     ifilcod, kpitch [,iskptim]            diskin
                                [,iwraparound][,iformat]
```

kpitch - Transpositionsverhältnis. 1=normal, 2=Oktave hoch, 0.5=Oktave runter -1=rückwärts, -2=rückwärts und Oktave hoch etc.
iwraparound - gibt an, ob am Ende des Files wieder begonnen werden soll (loop) 0=aus, 1=an.

in liest die Daten vom Standard Input. Mit Hilfe des *i*-Flags kann ein Soundfile als Input festgelegt werden.

```
  a1                 in                                              in
  a1,a2              ins                                             ins
  a1,a2,a3,a4        inq                                             inq
```

out sendet die Audiosamples als Soundfile auf die Harddisk. Mit Hilfe des *o*-Flags kann ein auch auf den Standard Output geschrieben werden. **outs** - Stereo-Output, **outq** - 4-Kanal-Output.

```
                     out        asig                                  out
                     outs       asig1, asig2                          outs
                     outs1      asig
                     outs2      asig
                     outq       asig1, asig2, asig3, asig4            outq
                     outq1      asig
                     outq2      asig
                     outq3      asig
                     outq4      asig
```

in und **out** gibt es auch in je einer 6-Kanal- (hexaphon) und einer 8-Kanal- (oktophon) Version: **inh, ino, outh, outo**.

pan verteilt ein Audiosignal auf 4 Kanäle in Abhängigkeit der Steuerparameter kx und ky.

```
a1, a2, a3, a4  pan        asig,kx,ky,ifn [,imode][,ioffset]          pan
```

kx - steuert das Signal auf der links-rechts-Ebene (0=links, 1=rechts)
ky - steuert das Signal auf der vorne-hinten-Ebene (0=hinten, 1=vorn)

Die einzelnen Lautsprecher sind also über folgende Werte zu erreichen:
ch1=links-vorn (0,1) , ch2=rechts-vorn (1,1) , ch4=links-hinten (0,0) , ch3=rechts-hinten (1,0)

ifn - Tabelle mit der Übergangscharakteristik für kx und ky.
imode - Indizierung der Tabelle:
1=Normalisiert (0-1) ,
0=einzelne Werte (0...Tabellengröße)
(optional, default 0)
ioffset - Offset für kx und ky: 0=ch3, 1=Quadro-Mitte (optional, default 0)

locsig verteilt ein Audiosignal auf 2 oder 4 Kanäle in Abhängigkeit der Steuerparameter der Polarkoordinaten (Winkel und Abstand). **locsend** erzeugt ebenfalls 2 oder 4 Signale, in Abhängigkeit des vorhergehenden **locsig** und dessen krevsend als Anteile für eine Verhallung.

```
a1, a2, a3, a4  locsig     asig,kdegree,kdist, krevsend          locsig
        a1, a2  locsig     asig,kdegree,kdist, krevsend          locsend
a1, a2, a3, a4  locsend
        a1, a2  locsend
```

kdegree - Winkel (0...360). a1 = 0, a2 = 90, a3 = 180, a4 = 270
kdist - Entfernungsparameter: Werte > 1 dämpfen die Ausgangssignale.
krevsend - Anteil für den Hall in Prozent (0...1)

SIGNAL DISPLAY

print zeigt den gegenwärtigen Wert von *i-time* Variablen oder Ausdrücken an.

print iarg [,iarg,...] **print**

printk zeigt den gegenwärtigen Wert von *k-time* Variablen oder Ausdrücken an.

printk kval, ispace [,itime] **printk**

ispace - Anzahl der Leerzeichen vor dem angezeigten Wert kval.
itime - Zeit in Sekunden bis zum nächsten angezeigten Wert (default 1).

display stellt in einem neuen Fenster die Wellenform des Audio- bzw. Kontrollsignals dar.

display xsig,iprd [,inprds][,iwtflg] **display**

xsig - darzustellendes Signal
iprd - darzustellender Zeitabschnitt
inprds - (optional) Anzahl der im Fenster gleichzeitig dargestellten Zeitabschnitte (default 1)
iwtflg - (optional) 1 = warten auf Mausklick nach jeder Periode, 0 = kein Warten (default 0)

dispfft stellt in einem neuen Fenster die Fourier-Transformierte des Audio- bzw. Kontrollsignals dar.

dispfft xsig,iprd,iwsiz[,iwtyp][,idbouti][,iwtflg] **dispfft**

xsig - darzustellendes Signal
iprd - darzustellender Zeitabschnitt
iwsiz - Größe des Analysefensters in Samples (muß eine 2-er Potenz sein)
iwtyp - (optional) Analysefenstertyp. 0 = Rechteck, 1 = Hanning (default 0).
idbouti - (optional) Skalierung der Fourierkoeffizienten: 0 = linear, 1 = dB (default 0).
iwtflg - (optional) 1 = warten auf Mausklick nach jeder Periode, 0 = kein Warten (default 0).

GRANULARSYNTHESE

Der **fof**-Generator stellt eine spezielle Form der Granularsynthese zur Erzeugung von Wellenformen mit einem Formanten zur Verfügung. as setzt sich aus vielen kurzen Tonimpulsen (*Grains*) zusammen, deren Einsatzabstände von xfund bestimmt werden.

```
as      fof     xamp,xfund,xform,koct,kband,kris,kdur,kdec,      fof
                iolaps,infa,ifnb,itotdur[,iphs][,ifmode]
```

xamp - Amplitude der einzelnen Grains.
Die Gesamtamplitude kann infolge der Überlappung größer als xamp werden.
xfund - Grundton in Hz. Gibt die Wiederholrate für die Grains an.
xform - Formantfrequenz in Hz. Bestimmt die Tonfrequenz innerhalb der Grains.
koct - Oktavierungsindex (normal = 0). Werte größer 0 verringern die Grundfrequenz durch Dämpfung ungeradzahliger Grains.
kband - Bandbreite des Formanten.
kris,kdur,
kdec - Hüllkurvenparameter für die einzelnen Grains.
iolaps - Speicherreservierung für die maximale zu erwartende Anzahl von Overlaps.
ifna,ifnb- Tabellennummer für Wellenform der Grains (a) und Einschwingcharakteristik (b)
itotdur - Gesamtdauer der fof-Aktivierung (normal = p3)
iphs - (optional, default 0) Initialphase der Grundfrequenz (0...1)
ifmode - Formantfrequenzmodus.
0 = jedes Grain behält die Frequenz xform mit der es aufgerufen wurde,
1 = die Grains ändern sich kontinuierlich mit xform.

Der **fog**-Generator ist eine Variante von **fof** zur genaueren Kontrolle der Granularsynthese. as setzt sich aus vielen kurzen Tonimpulsen (*Grains*) zusammen, deren Einsatzabstände von xdens bestimmt werden.

```
as      fog     xamp,xdens,xtrans,xspd,koct,kband,kris,          fog
                kdur,kdec, iolaps,infa,ifnb,itotdur
                [,iphs][,ifmode]
```

xamp - Amplitude der einzelnen Grains.
Die Gesamtamplitude kann infolge der Überlappung größer als xamp werden.
xdens - Anzahl der Grains pro Sekunde.
xtrans - Transpositionsfaktor. Bestimmt die Auslesegeschwindigkeit von Tabelle a innerhalb eines Grains. Negative Werte erzeugen rückwärts gelesene Grains.
xspd - "Schrittgeschwindigkeit" über die Tabelle a. xspd ist ein Zeiger auf Tabelle a mit Werten im Bereich [0...1], der angibt, von welcher Stelle der Tabelle das nächste Grain gelesen wird. Gut geeignet für Timestretching etc.
koct - Oktavierungsindex (normal = 0). Werte größer 0 verringern die Grundfrequenz durch Dämpfung ungeradzahliger Grains.
kband, kris,kdur,
kdec - Hüllkurvenparameter für die einzelnen Grains.
iolaps - Speicherreservierung für die maximale zu erwartende Anzahl von Overlaps.
ifna,ifnb- Tabellennummer (a) für das zu granulierende Soundfile (meist GEN01) sowie (b) Ein- und Ausschwingschwingcharakteristik
itotdur - Gesamtdauer der fog-Aktivierung (normal = p3)
iphs - (optional, default 0) Initialphase der Grundfrequenz (0...1)
ifmode - Transpositionsmodus.
0 = jedes Grain behält die Transposition xtrans mit der es aufgerufen wurde,
1 = die Grains ändern sich kontinuierlich mit xtrans.

sndwarp ist spezieller Soundfilegranulierer für Timestretching und Pitchshifting.
sndwarpst ist die Variante für Stereosoundfiles.

```
  as[,acmp] sndwarp   xamp,xtimewarp,xresample,ifn1,ibeg,                sndwarp
                      iwsize,irandw,ioverlap,inf2,itimemode

  as1,as2[,acmp1,acmp2]                                                 sndwarpst
            sndwarpst xamp,xtimewarp,xresample,ifn1,ibeg,
                      iwsize,irandw,ioverlap,inf2,itimemode
```

`acmp` - Vergleichssignal zum Ausgleichen starker Amplitudenüberhöhungen infolge mehrerer Überlappungen.
Sollte in einer nachfolgenden `balance`-Stufe verwendet werden.
`xamp` - Amplitude der einzelnen Grains.
Die Gesamtamplitude kann infolge der Überlappung größer als `xamp` werden.
`xtimewarp` - Bestimmt die Änderung des Zeitverlaufs des Originalsignals.
Wenn `itimemode` = 0, dann ist `xtimewarp` ein Zeitskalierungsfaktor.
Wenn `itimemode` ≠ 0, dann ist `xtimewarp` ein Zeitzeiger (wie in `pvoc`).
`xresample` - Transpositionsfaktor. Beeinflusst nicht den Zeitverlauf.
`ifn1` - Tabellennummer für das zu granulierende Soundfile (meist `GEN01`)
`ifn2` - Tabellennummer mit der Grainhüllkurve.
`ibeg` - Offset für den Anfang des Soundfiles in der Tabelle (in Sekunden).
`iwsize` - Fenstergröße in Samples (Grainsize). Üblicherweise im Bereich [200...2000].
`irandw` - Zufallsoffset für die Fenstergröße in Samples.
Üblicherweise im Bereich [0...`iwsize`/4].
`ioverlap` - Anzahl der Overlaps.
`itimemode` - siehe `xtimewarp.`

grain generiert Texturen durch Abtasten einer Funktionstabelle, z.B. eines Soundfiles (GEN 01).
Im Gegensatz zu **fof** oder **fog** gibt es hier auch Zufallsparameter.

```
  as        grain     xamp, xpitch, xdens, kampoff, kpitchoff,          grain
                      kgdur, igfn, iwfn, imgdur
```

`igfn` - Nummer der Tabelle, die die abzutastende Wellenform enthält.
Das kann ein Sinus oder ein gesampleter Klang sein
`iwfn` - Tabellennummer für die Hüllkurve der Grains.
`imgdur` - Maximale Graindauer in Sekunden.
Sollte `kgdur` größer sein als imgdur, wird `imgdur` verwendet.
`xamp` - Amplitude der einzelnen Grains.
`xpitch` - Grainfrequenz in Hz.
`xdens` - Dichte: Anzahl der Grains pro Sekunde.
Ist dieser Wert konstant, ist das Ergebnis, ähnlich **fof**,
synchrone Granularsynthese.
`kampoff` - Maximale Zufallsabweichung gegenüber `xamp`,
d.h. die Maximalamplitude eines Grains ist `xamp + kampoff`
und das Minimum `xamp`.
Ist `kampoff` gleich Null, gibt es keine zufälligen Amplitudenänderungen.
`kpitchoff` - Maximale Frequenzabweichung von `xpitch` in Hz. (ähnlich `kampoff`)
`kgdur` - Graindauer in Sekunden.
Der maximal zu erwartende Werte sollte gleich `imgdur` sein.
Falls `kgdur` zu irgendeiner Zeit größer als `imgdur` wird,
so wird er auf `imgdur` begrenzt.

granule ist komplexer als **grain**. Eine Granulartextur kann aus bis zu 128 *Voices* bestehen, d.h. parallelen Grainströmen.

```
ar      granule   xamp, ivoice, iratio, imode, ithd, ifn,          granule
                  ipshift, igskip, igskip_os, ilength,
                  kgap, igap_os, kgsize, igsize_os,
                  iatt, idec, [iseed],
                  [ipitch1], [ipitch2], [ipitch3], [ipitch4],
                  [ifnenv]
```

`xamp` - Amplitude.
`ivoice` - Anzahl der Stimmen.
`iratio` - Relative Geschwindigkeit des *grain skip pointers* bezogen auf die normale Geschwindigkeit.
z.B. 0.5 ist die halbe Geschwindigkeit.
`imode` - +1 tastet die Tabelle vorwärts ab, -1 rückwärts und 0 schaltet zufällig hin und her.
`ithd` - Schwellwert. Falls das Signal in der Tabelle kleiner als `ithd` ist, wird es übersprungen.
`ifn` - Tabellennummer der abzutastenden Wellenform bzw. des Samples.
`ipshift` - Steuerung der Transposition. Wenn `ipshift` 0 ist, wird jedes Grain zufällig im Bereich ± 1 Oktave transponiert.
Ist `ipshift` 1, 2, 3 or 4, werden die verschiedenen Voices mit den optionalen Faktoren `ipitch1`, `ipitch2`, `ipitch3` und `ipitch4` transponiert.
`igskip` - Offset für den Tabellenbeginn in Sekunden.
`igskip_os` - Zufallsoffset für den Grainzeiger in Sekunden. 0 ergibt keinen Offset.
`ilength` - Länge der Tabelle, die ab `igskip` genutzt werden soll, in Sekunden.
`kgap` - Lücke zwischen den Grains in Sekunden.
(Die Grains einer Voice überlappen sich nicht.)
`igap_os` - Zufallsoffset für die Lücke in % von `kgap`. 0 ergibt keinen Offset.
`kgsize` - Graindauer in Sekunden.
`igsize_os` - Zufallsoffset für die Grainsdauer in % von `kgsize`, 0 ergibt keinen Offset.
`iatt` - Attack der Grainhüllkurve in % der Graindauer `kgsize`.
`idec` - Decay der Grainhüllkurve in % der Graindauer `kgsize`.
`iseed` - Startwert für den internen Zufallsgenerator. (optional, default ist 0.5)
`ipitch1`, `ipitch2`, `ipitch3`, `ipitch4` - Transpositionsfaktoren, die benutzt werden, wenn `ipshift` 1, 2, 3 oder 4 ist.
z.B. ist 1.5 die Quinte nach oben, 0.5 die Oktave nach unten.
(optional, default für alle ist 1, die Originaltonhöhe)
`ifnenv` - Optionale Tabellennummer für die Kurvenform der Ein- und Ausschwingphase der Grainhüllkurve.

PLUCKED STRINGS

pluck realisiert den Karplus-Strong-Algorithmus.

```
  as        pluck       kamp,kcps,icps,ifn,imeth[,iparm1,iparm2]          pluck
```

`kamp` - Amplitude
`kcps` - Grundfrequenz
`icps` - Grundfrequenz für Speicherreservierung. Kann auch ungleich `kcps` sein.
`ifn` - Tabelle für Initial-Wellenform. 0 = weißes Rauschen.
`imeth` - Decay-Methode:
1 - *simple averaging*. Einfacher Tiefpaß (Mittelwertfilter).
2 - *stretched averaging*. Wie 1, mit längerem Decay (`iparm1` >1).
3 - *simple drum*. Rauhigkeit (Verhältnis tonhaft-geräuschhaft) über `iparm1`. `iparm` = 0.5 entspricht Rauschen („snare drum“).
4 - *stretched drum.* `iparm1` - Rauigkeit, `iparm2` - stretch factor.
5 - *weighted averaging*. Variante von 1 mit zusätzlicher Gewichtung des aktuellen (`iparm1`) und des vorhergehenden (`iparm2`) Samples.
6 - Rekursives Tiefpaßfilter.
`iparm1, iparm2` - Parameter in Abhängigkeit von `imeth`

wgpluck und **repluck** sind physikalische Modelle einer gezupften Saite. **repluck** arbeitet zusätzlich mit einem Audiosignal, das die Saite zum Schwingen anregt.

```
  as        wgpluck     iplk, xamp, icps, kpick, krefl                    wgpluck
  as        repluck     iplk, xamp, icps, kpick, krefl, axcite            repluck
```

`iplk` - Ort, an dem die Saite gezupft wird (0...1)
`xamp` - Amplitude
`icps` - Grundfrequenz
`kpick` - Ort des “Tonabnehmers” (0...1)
`krefl` - Reflexionskoeffizient. Bestimmt die Dämpfung und damit die Abklingdauer. 0 = 100 % Reflexion (langer Ton) , 1 = 0% Reflexion (kurzer Ton)
`axcite` - anregendes Audiosignal

ADDITIVE RESYNTHESE

Die additive Resynthese besteht in Csound aus 2 getrennten Prozessen.
Die Analyse wird mit dem Utility-Programm HETRO als Preprocessing ausgeführt:

```
csound -U hetro [flags] infilename outfilename
```
hetro

hetro zerlegt das Soundfile *infilename* in Sinuskomponenten und speichert deren Amplituden- und Frequenzverlauf in Form von Breakpoint-Tracks als Analysefile *outfilename* ab.
Folgende *flags* stehen zur Verfügung:

-s	sampling rate des Inputfiles. Default ist der Wert im Header des Soundfiles bzw. 10000, falls kein Header existiert.
-c	der zu analysierende Kanal. Default ist 1.
-b	Beginn des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0
-d	Dauer des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0, d.h. bis zum Ende des Soundfiles.
-f	Erwartete Grundfrequenz. Default ist 100 (Hz).
-h	Anzahl der Harmonischen, die gesucht werden sollen. Default ist 10, Maximum 50.
-M	Maximale Amplitude summiert über alle gleichzeitigen Tracks. Default ist 32767.
-m	Minimale Amplidude als Schwellwert für die Analyse. Typische Werte sind: 128 (-48 dB), 64 (-54 dB), 32 (-60 dB), 0 (0 dB). Default ist 64 (= -54 dB).
-n	Anzahl der Breakpoints in jedem Amplituden und Frequenztrack. Default ist 256. Breakpoints sind über die gesamte Dauer gleichverteilt.
-l	Grenzfrequenz für ein alternatives Tiefpaßfilter anstelle des normalen Mittelwertkammfilters . Default ist 0, d.h. kein Tiefpaß.

Beispiel: √

```
csound -U hetro -h32 -f80 testsound testad
```

Das Soundfile `testsound` wird in 32 Teiltöne, beginnend mit 80 Hz, zerlegt.
Die Analysetracks werden in `testad` gespeichert.

Das File mit den Analysedaten kann dann im Orchestrafile mit dem Csoundgenerator **adsyn** gelesen und mit veränderten Parametern resynthetisiert werden:

```
asig    adsyn    kamod, kfmod, ksmod, ifilcod
```
adsyn

`kamod` - Faktor für die Werte der Amplitudentracks, z.B. ist 0.5 die Hälfte, 2.0 das Doppelte des Normalwertes
`kfmod` - Faktor für die Werte der Frequenztracks,z.B.ist 0.5 eine Oktave tiefer
`ksmod` - Faktor für die Geschwindigkeit, mit der die Breakpoints gelesen werden, 2.0 ist doppelt so schnell.
`ifilcod` - Name oder Nummer des Analysefiles; bei Angabe einer Nummner *n* muß das File *name.n* benannt worden sein.

Beispiel: √

```
kf      line      1,p3,1.5
a1      adsyn     0.7, kf, 0.5,"testad"
```

Das Analysefile `testad` wird mit halber Geschwindigkeit, 70% Amplitude und dynamisch modulierter Frequenz (Glissando zur Quinte) resynthetisiert.

PHASENVOCODER

Der Phasenvocoder besteht in Csound aus 2 getrennten Prozessen.
Die Analyse wird mit dem Utility-Programm PVANAL als Preprocessing ausgeführt:

```
csound -U pvanal [flags] infilename outfilename
```

pvanal

pvanal zerlegt das Soundfile *infilename* in eine Reihe Kurzzeit-FFT-Frames und speichert Amplituden- und Phasenwerte als Analysefile *outfilename* ab.
Folgende *flags* stehen zur Verfügung:

-s	sampling rate des Inputfiles. Default ist der Wert im Header des Soundfiles bzw. 10000, falls kein Header existiert.
-c	der zu analysierende Kanal. Default ist 1.
-b	Beginn des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0
-d	Dauer des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0, d.h. bis zum Ende des Soundfiles.
-n	Größe der FFT-Frames in Samples. Dieser Wert muß immer eine 2er Potenz sein. Ein Frame sollte immer größer als die maximale Periodendauer (= tiefste Frequenz) des zu analysierenden Soundfiles sein. Der Defaultwert ist die kleinste Potenz von 2, die einer Dauer größer als 20 ms (= 50 Hz) entspricht. Bei 44.1 kHz wäre dies 1024 (entspricht 23.2 ms)
-w	Fensterüberlappungsfaktor. Dieser Wert bestimmt indirekt die Anzahl der Frames je Sekunde. Der normale Wert ist 4 (default), d.h. jedes Sample im Soundfile erscheint in 4 FFT-Windows. Sehr kleine Werte verwischen zeitliche Feinheiten des Klangs, sehr große Werte haben sehr große Analysefiles zur Folge. Das -w Flag darf nur alternativ zu -h benutzt werden.
-h	Fensteroffset. Dieser Wert gibt den Abstand des Beginns aufeinanderfolgender Analysefensterin Samples an. Das -h Flag darf nur alternativ zu -w benutzt werden.

Beispiel: √

```
csound -U pvanal -w6 testsound testpv
```

Das Soundfile `testsound` wird mit einem Overlapfaktor 6 und den übrigen Defaultwerten analysiert und als `testpv` gespeichert.

Das File mit den Analysedaten kann dann im Orchestrafile mit den Csoundgeneratoren **pvoc**, **pvread**, **pvbufread**, **pvinterp**, **pvcross** oder **vpvoc** gelesen und mit veränderten Parametern resynthetisiert werden:

```
  asig     pvoc       ktimpt, kfmod, ifilcod [,ispecwp]                    pvoc
```

`ktimpt` - Zeit-Zeiger, der die zu lesende Stelle im Analysefile in Sekunden angibt. Er muß immer positiv sein und kann sich mit beliebiger Geschwindigkeit ändern.
`kfmod` - Faktor für die Werte der Frequenztracks,z.B.ist 0.5 eine Oktave tiefer
`ifilcod` - Name oder Nummer des Analysefiles; bei Angabe einer Nummner *n* muß das File *name.n* benannt worden sein.
`ispecwd` - Wenn dieser Wert nicht Null ist, wird versucht, die spektrale Hüllkurve (Formanten) bei Transpositionen (`kfmod`) zu erhalten.Default ist 0.

Beispiel: √

```
kt          line          0,p3,10.0
a1          pvoc          kt, 1.2,"testpv"
```

10 Sekunden des Analysefiles `testpv` werden innerhalb der Zeit `p3`, transponiert um eine kleine Terz, resynthetisiert.

pvread liest fortlaufend die Frequenz- und Amplitudenwerte des angegebenen Analysekanals. Diese Werte können an anderer beliebiger Stelle weiterverarbeitet werden.

```
  kfreq,kamp pvread  ktimpt, ifilcod, ibin                                pvread
```

`ktimpt` - Zeit-Zeiger, der die zu lesende Stelle im Analysefile in Sekunden angibt. Er muß immer positiv sein und kann sich mit beliebiger Geschwindigkeit ändern.
`ifilcod` - Name oder Nummer des Analysefiles; bei Angabe einer Nummner *n* muß das File *name.n* benannt worden sein.
`ibin` - Nummer des Analysekanals (bin)

Beispiel: √

```
kt          line          0,p3,10.0
kf,ka       pvread        kt,"testpv", 34
a1          oscil         ka,kf,1
```

Der 34. Kanal des Analysefiles `testpv` wird benutzt, um einen Oszillator in Amplitude und Frequenz zu steuern.

pvbufread speichert die Analysedaten des angegebenen Files. Diese Werte können von darauffolgenden **pvinterp** - oder **pvcross**-Generatoren zusammen mit den Werten eines zweiten Files zur Resynthese benutzt werden. **pvinterp** interpoliert mit entsprechenden Skalierungen sowohl Frequenzen als auch Amplituden beider Files. **pvcross** mischt mit entsprechenden Skalierungen nur die Amplituden beider Files, die Frequenzen stammen immer vom 2. File.

```
            pvbufread  ktimpt, ifilcod                                    pvbufread
  asig      pvinterp   ktimpt, kfmod, ifilcod, kfreqscale1,               pvinterp
                       kfreqscale2, kampscale1, kampscale2,
                       kfreqinterp, kampinterp
  asig      pvcross    ktimpt, kfmod, ifilcod, kamp1, kamp2               pvcross
                       [,ispecwp]
```

`kfreqscale` - Frequenzskalierung für jedes der beiden Analysefiles, bevor interpoliert wird.
`kampscale` - Amplitudenskalierung für jedes der beiden Analysefiles, bevor interpoliert wird.

`kfmod` - gesamte Frequenzskalierung nach Interpolation.
`kfreqinterp, kampinterp`
- Interpolationswerte [0...1], 0 = 100 % des 1. Analysefiles (**`pvbufread`**), 1 = 100% des 2. Analysefiles (**`pvinterp`**).

`kamp` - Amplitudenskalierung der einzelnen Files vor Mischung der Amplituden.

Beispiel: √

```
kt1         line          0,p3,10.0
kt2         line          0,p3,15.0
kipl        linseg        0,p3/2,1,p3/2,0
            pvbufread     kt1,"test1pv"
amix        pvinterp      kt2,1,"test2pv",1,1,1,1,kipl, 1-kipl
```

Am Anfang und Ende stammen die Frequenzen zu 100% vom 1. File und die Amplituden zu 100% vom 2. File. In der Mitte der Notendauer ist es umgekehrt. Dazwischen wird interpoliert.

`vpvoc` entspricht **`pvoc`**, ändert aber die Amplitudenwerte der Analyse entsprechend einer spektralen Hüllkurve. Die Hüllkurve wird von einem vorangegangenen **`tableseg`**- oder **`tablexseg`**-Generator bereitgestellt. Die Gestalt der Hüllkurve ist durch Verwendung verschiedener Funktionstabellen dynamisch änderbar. **`tableseg`** interpoliert linear zwischen Tabellen, **`tablexseg`** interpoliert exponentiell (wie `linseg` und `expseg`).

```
            tableseg   ifn1, idur1, ifn2 [, idur2, ifn3 [...]      tableseg
            tablexseg  ifn1, idur1, ifn2 [, idur2, ifn3 [...]      tablexseg
  asig      vpvoc      ktimpt, kfmod, ifilcod [,ispecwp]           vpvoc
```

`ifn1` ... - Tabellen, die als spektrale Hüllkurven für **`vpvoc`** benutzt werden. Die Tabellengröße muss gleich der halben FFT-size der Analyse sein.
`idur1` ... - Interpolationszeiten zwischen zwei Tabellen.

Beispiel: √

```
ktime       line          0,p3,10.0
            tableseg      1,p3/2,2,p3/2,3
apv         vpvoc         ktime,1,"testpv"

;scorefile:
f1 0 256 5 .001 20 1 236 .001
f2 0 256 5 .001 256 1
f3 0 256 5 1 256 .001
```

Die spektralen Amplituden des Analysefiles `testpv` werden durch die Formen in den Tabellen 1-3 modifiziert (1 - "Bandpass", 2 - "Hochpass", 3 - "Tiefpass").

LINEAR PREDICITVE CODING

Das Utility-Programm LPANAL führt eine kombinierte LPC- (Multi-Pol-Filter-) und Grundtonanalyse durch und speichert als Analysefile die Filterkoeffizienten (als Repräsentation der spektralen Hüllkurve) sowie einige Steuerparameter für eine Resynthese ab.

```
csound -U lpanal [flags] infilename outfilename
```
lpanal

Folgende *flags* stehen zur Verfügung:

-s	sampling rate des Inputfiles. Default ist der Wert im Header des Soundfiles bzw. 10000, falls kein Header existiert.
-c	der zu analysierende Kanal. Default ist 1.
-b	Beginn des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0
-d	Dauer des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0, d.h. bis zum Ende des Soundfiles.
-p	Anzahl der Filterpole. Default ist 34, Maximum ist 50.
-h	Abstand des Beginns aufeinanderfolgender Analysefenster (*hopsize*) in Samples. Die Größe der Fenster ist immer doppelt so groß, so daß sich immer 2 Fenster überlappen. Wenn *hopsize* klein und die Anzahl der Filterpole groß ist, können die errechneten Filter instabil werden. Default ist 200, Maximum ist 500.
-C	Kommentartext
-P	tiefste Frequenz für das Pitchtracking. 0 bedeutet kein Pitchtracking. Default ist 70.
-Q	höchste Frequenz für das Pitchtracking. Je schmaler das Band zwischen P und Q, umso genauer kann die Analyse werden. Default ist 200.
-v	Anzeige von Analyseinformationen. 0=keine, 1=normal, 2=sehr ausführlich.

Beispiel: √

```
csound -U lpanal -p26 -P100 -Q250 testsound testlp
```

Die spektrale Hüllkurve des Soundfiles `testsound` wird als 26-pol-Filter dargestellt, die Grundtöne können zwischen 100 und 250 Hz liegen.
Filterkoeffizienten, Fehlerwerte, Grundton und Amplitudenmittelwerte werden als `testlp` gespeichert.

Die Analysedaten können dann im Orchestrafile mit dem Csoundgenerator **lpread** gelesen und die Parameter zur Ansteuerung eines dazugehörigen zeitvarianten Filters **lpreson** oder **lpfreson** benutzt werden:

```
krmsr,krmso,kerr,kcps lpread ktimpt,ifilcod[,inpoles][,ifrmrate]   lpread
af       lpreson    asig                                           lpreson
af       lpfreson   asig, kfrqratio                                lpfreson
```

`ktimpt` - Zeit-Zeiger, der die zu lesende Stelle im Analysefile in Sekunden angibt. Er muß immer positiv sein und kann sich mit beliebiger Geschwindigkeit ändern.
`ifilcod` - Name oder Nummer des Analysefiles; bei Angabe einer Nummner *n* muß das File *name.n* benannt worden sein.
`inpoles` - Anzahl der Filterpole, wenn das File keinen Header hat. (optional, default 0)
`ifrmrate` - Frames pro Sekunde, wenn das File keinen Header hat. (optional, default 0)
`krmsr` - quadratischer Mittelwert (rms) der Amplitude des n-pol Filters
`krmso` - quadratischer Mittelwert der Amplitude des Originalsignals
`kerr` - normalisiertes Fehlersignal (0...1)
`kcps` - analysierter Grundton in Hz
`kfrqratio` - Frequenzverschiebungsfaktor für das Spektrum (Formanten)

lpread und **lpreson** bzw. **lpfreson** müssen immer als Paar benutzt werden. Die Filter **lpreson** bzw. **lpfreson** werden immer vom dem Analysefile gesteuert, das vom vorhergehenden **lpread** geladen wurde.
Das Steuersignal `kcps`, das die Information über den Grundtonverlauf des analysierten Klangs enthält, läßt sich direkt als Frequenz eines Oszillators einsetzen. `krmso` eignet sich für die Amplitudensteuerung. Das (üblicherweise obertonreiche) Ausgangssignal dieses Oszillators kann nun durch **lpreson** oder **lpfreson** gefiltert werden. Somit verhalten sich Grundton und spektrale Hüllkurve dieses synthetischen Klangs wie der Originalklang.
Das Fehlersignal `kerr` widerspiegelt den Grad der Periodizität des analysierten Klanges. Niedrige Werte nahe Null verweisen auf hohe Periodizität, d.h. ausgeprägten Tonhöhencharakter. Höhere Werte nahe Eins verweisen auf starke (zufällige) Schwankungen im Klang, d.h. inharmonischen oder Geräuschcharakter. Man kann daher dieses Fehlersignal zum Umschalten oder Mischen zwischen einer harmonischen (Tonhöhen-) Quelle und einer Rauschquelle benutzen. Diese Signale bzw. die Mischung aus beiden kann durch **lpreson** oder **lpfreson** spektral geformt werden.

Beispiel: √

```
kt          line          0,p3,10.0
krr,kro,ke,kf lpread      kt,"testlp"
a1          oscil         kro,kf,1
a2          rand          kro
af          lpreson       (kerr > 0.3 ? a2,a1)
```

10 Sekunden des Analysefiles `testlp` werden innerhalb der Zeit `p3` linear gelesen. `kro` bestimmt die Amplituden des Ton- und des Rauschgenerators. `kf` steuert die Frequenz von `a1`. Wenn `kerr` > 0.3 ist, wird das Rauschen gefiltert, ansonsten der Ton `a1`. Die *ftable* 1 sollte eine sehr obertonreiche Wellenform enthalten, z.B. Sägezahn, Puls o.ä.

Mit Hilfe der folgenden opcodes kann man zwischen den spektralen Hüllkurven zweier LPC-Analysen interpolieren.

lpslot islot **lpslot**
lpinterpol islot1, islot2, kmix **lpinterpol**

islot - Nummer des Zwischenspeichers für nachfolgend gelesene LPC-Daten [0...20]
islot1,
islot2 - Nummern der Daten, zwischen denen interpoliert werden soll.
kmix - Interpolationsindex [0...1], 0 = 100% der 1. Analyse, 1=100% der 2. Analyse

lpslot legt fest, unter welcher internen Nummer die Analysedaten des unmittelbar danach folgenden **lpread**-Moduls zwischengespeichert werden.
lpinterpol interpoliert zwischen beiden Analysen entsprechend dem Index kmix und legt das Ergebnis im aktuellen Zwischenspeicher für Analysedaten ab. Diese Werte können von einem darauffolgenden **lpreson** zu Filterung benutzt werden.

Beispiel: √

```
asrc        buzz          30000, 400, 10, 1
ktime       line          0,p3,p3
            lplot         0
krr,kro,ke,kf lpread      ktime,"oboelp"
            lplot         1
krr,kro,ke,kf lpread      ktime,"geigelp"
kndx        line          0,p3,1
            lpinterp      0,1,kndx
aout        lpreson       asrc
aout        balance       aout, asrc
```

Die Interpolation zweier Analysefiles (oboelp und geigelp) wird benutzt, um einen obertonreichen Klang dynamisch zu filtern.

CONVOLUTION

Eine Convolution oder Faltung besteht in Csound aus 2 getrennten Prozessen.
Zunächst wird mit dem Utility-Programm CVANAL als Preprocessing eine Konvertierung des Faltungssignals (Impulsantwort) in ein einziges FFT-Frame durchgeführt:

```
csound -U cvanal [flags] infilename outfilename
```
cvanal

Folgende *flags* stehen zur Verfügung:

-s	sampling rate des Inputfiles. Default ist der Wert im Header des Soundfiles bzw. 10000, falls kein Header existiert.
-c	der zu analysierende Kanal (1,2,3 oder 4). Wird kein Wert angegeben, werden alle Kanäle analysiert.
-b	Beginn des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0
-d	Dauer des zu analysierenden Audiosegmentes in Sekunden. Default ist 0.0, d.h. bis zum Ende des Soundfiles.

Beispiel: √

```
csound -U cvanal -b3.0 -d2.0 hall hallcv
```

2 Sekunden des Soundfiles `hall` werden ab der 3. Sekunde mit allen Kanälen analysiert und die Fouriertransformierte (das Spektrum) dieses Abschnitts als `hallcv` gespeichert.

Im Orchestrafile kann mit Hilfe des Csoundgenerators **`convolve`** ein Audiosignal mit dem Analysefile (= die FFT der Impulsantwort) gefaltet werden:

```
a1[,a2[,a3,a4]] convolve asig, ifilcod, channel
```
convolve

`asig`	-	zu filterndes Signal.
`ifilcod`	-	Name oder Nummer des Analysefiles; bei Angabe einer Nummner *n* muß das File *name.n* benannt worden sein.
`channel`	-	Nummer des zu verwendenden Kanals der Analyse (1...4). Default ist 0, d.h. alle Kanäle des Analyse werden benutzt. Die Anzahl der Output-Signale muß mit der Anzahl der verwendeten Analysekanäle übereinstimmen (1, 2 oder 4).

Bei einer Faltung mit **`convolve`** ist das Ergebnis immer um den Betrag der Dauer der Impulsantwort verzögert, mindestens jedoch um eine k-Periode (1/kr).
Bei der Implementierung einer Mischung eines trockenen und des verhallten Signal, sollte ebenfalls der trockene Anteil mit Hilfe von **`delay`** um den gleichen Betrag verzögert werden.

Beispiel: √

```
asig        soundin       "voice.aiff"
a1,a2       convolve      asig,"hallcv"
adry        delay         asig,2.0
aleft       =             adry*0.8 + a1*0.2
aright      =             adry*0.8 + a2*0.2
            outs          aleft,aright
```

Das Soundfile `voice.aiff` wird mit der FFT von `hall` gefaltet und im Verhältnis 4:1 gemischt.
Die Verzögerung durch `convolve` wird mit Hilfe eines Delays von 2 Sekunden ausgeglichen.

SCORE STATEMENTS

ALLGEMEINES

Statements im Scorefile haben die Form:

```
opcode p1 p2 p3 p4 .... ; comments
```

opcode ist ein einzelner Buchstabe **f**, **i**, **a**, **t**, **s** oder **e**, und steht am Anfang einer Zeile.

p1 p2 p3 p4 ... sind Parameterfelder (*pfields*) und beinhalten Fließkommazahlen. Sie werden durch Leerzeichen oder Tabulatoren voneinander getrennt. **pfields**
Kommentare sind optional und können hinter dem letzten Parameter oder am Anfang einer Zeile mit einem Semikolon (**;**) begonnen werden.

PREPROCESSING

Ein Score kann in zeitlich aufeinanderfolgende Sektionen gegliedert werden. Bevor die Instrumente des Orchestrafiles die Informationen aus dem Scorefile lesen, wird es in 3 Stufen vorverarbeitet:
1. Carry - Innerhalb einer Gruppe aufeinanderfolgender i-Statements mit gleichem Instrument **. (carry)**
(p1) werden leere pfields mit den Werten entsprechender pfields vorhergehender **i**-Statements gefüllt. Ein leeres pfield wird mit einem Punkt (**.**) gekennzeichnet. Nach dem letzten nichtleeren pfield einer Zeile ist kein Punkt erforderlich. Der Carry-Prozeß wird durch ein **i**-Statement mit anderem p1 oder durch ein anderes Statement (**f, a,** etc.) abgebrochen. Beim Parameter p2 (Einsatzzeit) kann durch Kennzeichnung mit + der Wert aus vorhergehender Einsatzzeit und Dauer (p3) automatisch berechnet werden.

2. Tempo - Dieser Prozeß ermittelt aus den *beats* in p2 (Einsatzzeit) und p3 (Dauer) und den Tempoangaben die realen Dauern und Einsatzzeiten in Sekunden.

3. Sort - Dieser Prozeß sortiert alle Statements in zeitlicher Folge entsprechend dem Wert in p2. Haben ein **f**- und ein **i**-Statement den gleichen Wert, so wird das **f**-Statement vorgzogen.

Am Ende jeder dieser Operationen werden eventuell vorhandene weitere Symbole ausgewertet:

Next-P , Previous-P - **i**-Statement-pfields, die die Symbole **np**x bzw. **pp**x (x ist eine ganze **np**
positive Zahl) beinhalten , werden durch den Wert des x-ten pfields des vorhergehenden (ppx) **pp**
oder folgenden (npx) Statements ersetzt. Zum Beispiel wird für das Symbol np7 der Wert aus p7 der nächsten Note des gleichen Instruments eingetragen. Existiert ein solches pfield nicht, so wird der Wert 0 angenommen. Diese Symbole sind nicht in p1,p2 und p3 erlaubt.

Ramping - **i**-Statement-pfields, die das Symbol **<** beinhalten, werden durch Werte einer linearen **< { ~**
Interpolation ersetzt. Interpoliert wird zwischen dem letzten realen Wert des gleichen pfields vor und dem ersten realen Wert nach den Statements mit dem Symbol **<** . Ramping in p1,p2 und p3 ist nicht erlaubt. Mit dem Symbol **{** wird exponentiell statt linear interpoliert. Mit **~** werden gleichverteilte Zufallswerte zwischen der ersten und letzten Zeile erzeugt.

Expressions - In eckigen Klammern **[]** können numerische Ausdrücke mit **+ - * / ()** **[ ]**
und Zahlen anstelle einfacher pfield-Werte benutzt werden, z.B. [8/3 + 0.4].

I-STATEMENT (INSTRUMENT BZW. NOTE)

i p1 p2 p3 p4 ... **i-Statement**

Dieses Statement ruft ein Instrument des Orchestrafiles auf.

p1 - Instrumentennummer .
Ein negativer Wert beendet eine gehaltene Note des gleichen Instruments
p2 - Einsatzzeit in beats. Das voreingestellte Tempo ist 60.
p3 - Dauer in beats. Ein negativer Wert erzeugt eine gehaltene Note.

p4 ... - Parameter, deren Bedeutung im Orchestrafile bestimmt wurde.

Leere Parameterfelder werden durch einen Punkt (**.**) bezeichnet. Bei leeren Parameterfeldern nach dem letzten nichtleeren Feld, kann der Punkt weggelassen werden.
Leere Parameterfelder werden automatisch mit den vorherigen Werten des gleichen Instruments gefüllt.

Für p2 kann ein **+** geschrieben werden: dann wird die Summe aus p2 und p3 des vorherigen + Statements eingesetzt. (≈ “legato”).

Für die lineare Interpolation zwischen mehreren aufeinanderfolgenden gleichen Parameterfeldern kann anstelle der Zwischenwerte das Symbol **<** eingesetzt werden. Dann werden alle **<** mit Interpolationswerten ersetzt, die aus dem letzten und dem nächsten angegebenen Wert des Parameterfeldes errechnet werden.

A-STATEMENT (ADVANCE)

a p1 p2 p3 **a-Statement**

Dieses Statement fügt leere beats im Score ein, d.h. es werden keine Samples berechnet. Das advance-statement dient im wesentlichen als Platzhalter für noch nicht fertiggestellte Partiturteile.

p1 - Ohne Bedeutung, sollte 0 sein.
p2 - Beginn in beats. Das voreingestellte Tempo ist 60.
p3 - Dauer in beats.

T-STATEMENT (TEMPO)

t p1 p2 p3 p4 p5 ... **t-Statement**

Dieses Statement setzt das Tempo für die jeweilige Sektion. Default ist 60. (**t** 0 60)

p1 - Muß 0 sein.
p2 - Anfangstempo in beats / minute .
p3,p5,p7... - beats für neues Tempo.
p4,p6,p8... - neues Tempo ab entsprechendem beat.

Werden p3,p4 oder weitere Parameter angegeben, wird zwischen dem Anfangstempo und diesem Wert ein kontinuierliche Tempoänderung durchgeführt (*accel.* bzw. *rit*.).

V-STATEMENT (TIME VARYING)

v p1 **v-Statement**

Dieses Statement gibt mit p1 einen Multiplikator für nachfolgende Einsatzzeiten und Dauern an (p2 und p3). Es ist gültig bis zum nächsten **v**-Startement oder bis zum Ende der Section.

S-STATEMENT (SEKTIONSENDE)

s ... **s-Statement**

Dieses Statement markiert das Ende einer Sektion. Innerhalb einer Sektion werden **i**-,**f**- und **a**-Statements sortiert und Tempi berechnet. In jeder neuen Sektion werden die beats bzw. Sekunden von neuem gezählt.

E-STATEMENT (ENDE)

e ... **e-Statement**

Dieses Statement markiert das Ende des Scorefiles, beendet alle gehaltenen Noten und schließt das geschriebene Soundfile. Nachfolgende Zeilen werden ignoriert.

F-STATEMENT (FUNCTION TABLE)

(siehe GEN ROUTINES...)

GEN ROUTINES

Die GEN Unterprogramme (*subroutines*) stellen verschiedene Methoden zum Generieren von Tabellen bereit. Mit Hilfe eines **f** - Statements im Scorefile können diese Prozeduren aufgerufen werden. Zum Zeitpunkt `p2` werden die Tabbellen errechnet und stehen dann für alle Instrumente zur Verfügung. Der Zugriff auf die Tabellen erfolgt über die Angabe der Nummer `p1` in den entsprechenden Statements der Instrumente.

f - Statements haben die Form:

```
  f # time size GEN p5 p6 ....
```

f-Statements

`#` - Nummer der Tabelle (von 1 bis 200).
Eine zweite Tabelle mit gleicher Nummer löscht die erste.
Eine negative Nummer löscht die Tabelle mit der entsprechenden positiven Nummer zur angebenen Zeit `time` in `p3.`

`time` - Aktivierungszeit in beats.

`size` - Größe der Tabelle in Punkten.
Sie muß einer 2-er Potenz bzw. einer 2-er Potenz + 1 entsprechen, die maximale Größe ist 16777216 (2^24).

`GEN` - Nummer der aufzurufenden GEN Prozedur.
Ein negativer Wert verhindert das Normalisieren.

`p5,p6 ...` - spezielle Parameter in Abhängigkeit von der jeweiligen Prozedur.

GEN01

Lesen eines Soundfiles in eine Tabelle.

```
f # time size 1 filecode skiptime format channel
```

`size` - Anzahl der Punkte (Samples) - normalerweise eine 2-er Potenz (+1). Bei AIFF- und SoundDesigner-Dateien kann die explizite Angabe der Größe durch Einsetzen von 0 für `size` entfallen - in diesem Fall wird beim Lesen des Soundfiles die Größe der Datei verwendet. Wenn die Größe des Soundfiles keiner 2-er Potenz entspricht, kann die Tabelle nur vom **`loscil`**-Modul benutzt werden.

`filecode` - Nummer oder Dateiname des Soundfiles.
Eine Nummer bezeichnet eine Datei `"soundin.filecode"`. Das Soundfile wird zuerst im aktuellen Ordner gesucht, danach im *Soundsample Directory* sowie im *Soundfile Directory.*

`skiptime` - Lesen ab `skiptime` Sekunden vom Anfang des Files.

`format` - Angabe des Audio Formats:

1 - 8-bit	4 - 16-bit
2 - 8-bit A-law codiert	5 - 32-bit
3 - 8-bit µ-law codiert	6 - 32-bit Fließkommazahlen

`channel` - Lesen eines bestimmten Kanals (1,2,3 oder 4) oder aller Kanäle (0).

Wenn p4 positiv ist (1), wird die Tabelle nach dem Einlesen normalisiert, d.h. alle ursprünglichen Werte (bei 16-bit zwischen - 32768 bis + 32767) werden so skaliert, daß der Maximalwert 1.0 beträgt.
Ein negativer Wert für p4 (-1) verhindert das Normalisieren.

Beispiele: √

```
f1 1 0 8192 1 23 0 4 0
f2 2 0 0 -1 "trumpet" 0 4 1
```

Tabelle 1 wird mit allen Kanälen vom Soundfile "soundin.23" bis zur angegebenen Größe von 2^13 Samples gefüllt. Anschließend wird normalisiert.
Tabelle 2 liest von "trumpet" den ersten Kanal mit der Originallänge und ohne Normalisierung.

GEN02

Unmittelbares Setzen der Tabellenwerte.

```
f # time size 2 v1 v2 v3 ...
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`v1,v2,v3, ...` - direkte Angabe der Tabellenwerte. Die Anzahl der Werte ist derzeit auf 150 begrenzt. Werden weniger Werte als `size` angegeben, wird für den Rest 0 eingesetzt.

Wenn p4 positiv ist (2), wird die Tabelle nach dem Einlesen normalisiert, d.h. alle Werte werden so skaliert, daß der Maximalwert 1.0 beträgt. Ein negativer Wert für p4 (-2) verhindert das Normalisieren.

Beispiel: √

```
f1 1 0 16 -2 0 1 2 3 4 5 6 7 8 9 10 11
```

Tabelle 1 wird mit den angegebenen 12 Werten gefüllt.
Die restlichen 4 Positionen werden auf 0 gesetzt. Keine Normalisierung.

GEN03

Generierung der Tabellenwerte durch Auswertung eines Polynoms im angegebenen Intervall.

```
f # time size 3 xval1 xval2 c0 c1 c2 ... cn
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`xval1,xval2` - Linke und rechte Begrenzung des x-Intervalls (`xval1<xval2`)

`c0,c1,c2, ... cn` - Koeffizienten des Polynoms n-ten Grades.

$$x) = c_0 + c_1 x + c_1 x^2 + c_2 x^3 + \ldots + c_n x^n$$

Wenn p4 positiv ist (3), wird die Tabelle nach dem Einlesen normalisiert, d.h. alle Werte werden so skaliert, daß der Maximalwert 1.0 beträgt. Ein negativer Wert für p4 (-3) verhindert das Normalisieren.

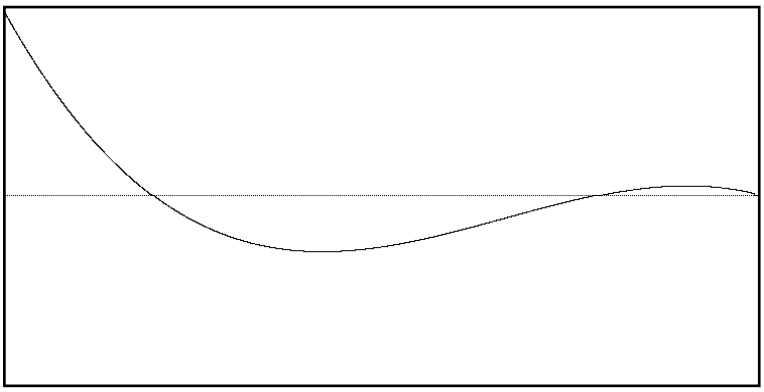
Beispiel:

```
f1 0 8193 3  -3 4 -10 0.2 3 -0.6
```

Die Tabelle 1 wird mit Funktionswerten eines Polynoms 3. Grades

$$x) = -10 + 0,2x + 3x^2 - 0,6x^3$$

im Intervall {-3<x<4} gefüllt und anschließend normalisiert.



GEN05, GEN07

Generieren von Funktionen aus linearen (**GEN07**) oder exponentiellen (**GEN05**) Segmenten.

```
f # time size 5 a n1 b n2 c ...
f # time size 7 a n1 b n2 c ...
```

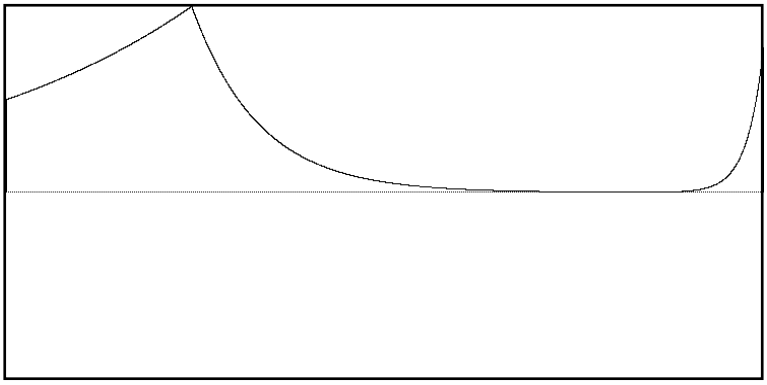
`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`a,b,c ...` - Funktionswerte in ungeraden *pfields* (p5,p7,p9 ...). Bei GEN05 müssen die Werte ≠ 0 sein und gleiches Vorzeichen haben. Bei GEN07 gelten keine Einschränkungen.

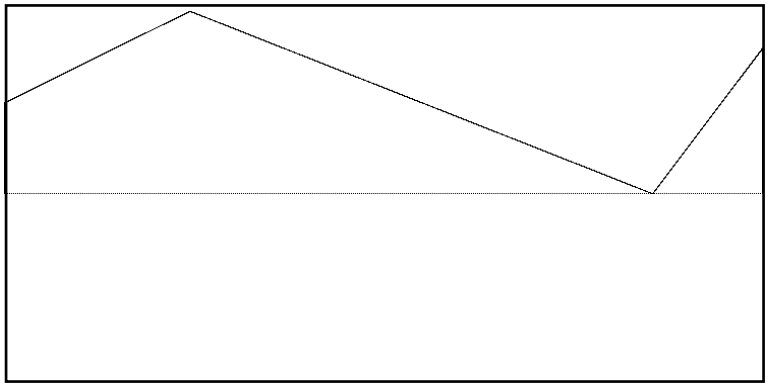
`n1,n2 ...` - Länge der Segmente in Punkten in geraden *pfields* (p6,p8,p10 ...). Segmente können auch die Länge 0 haben. Die Summe aller `n` sollte `size` entsprechen - ist sie kleiner, werden die restlichen Punkte null gesetzt, ist sie größer, werden nur Werte bis `size` gespeichert.

Beispiele: √

```
f2 0 8193 5 0.5 2000 1 5000 0.001 1193 0.8
```



```
f3 0 8193 7 0.5 2000 1 5000 0.001 1193 0.8
```



GEN06

Generieren einer aus Segmenten kubischer Polynome bestehenden Funktion.

```
f # time size 6 a n1 b n2 c n3 d ...
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`a,c,e ...` - lokale Maxima oder Minima aufeinanderfolgender Segmente.

`b,d,f ...` - Umkehrpunkte, Segmentbegrenzungen.

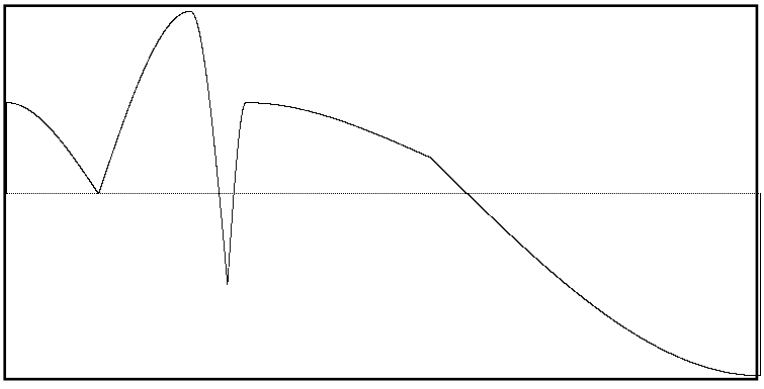
`n1,n2,n3 ...` - Anzahl der gespeicherten Werte zwischen den angegebenen Punkten. Segmente können auch die Länge 0 haben. Die Summe aller `n` sollte `size` entsprechen - ist sie kleiner, werden die restlichen Punkte null gesetzt, ist sie größer, werden nur Werte bis `size` gespeichert.

Wenn p4 positiv ist (6), wird die Tabelle nach dem Einlesen normalisiert, d.h. alle Werte werden so skaliert, daß der Maximalwert 1.0 beträgt. Ein negativer Wert für p4 (-6) verhindert das Normalisieren.

Die Segmente werden durch jeweils 3 Punkte beschrieben: Umkehrpunkt, Maximum/Minimum, Umkehrpunkt. Das erste komplette Segment wird aus `b,c,d` mit der Länge `n2+n3` gebildet, das nächste aus `d,e,f` mit der Länge `n4+n5` usw. Der erste Abschnitt `a,b` mit `n1` beginnt mit einem Maximum/Minimum.

Beispiel: √

```
f4 0 8193 6 0.5 1000 0 1000 1 400 -0.5 200 0.5 2000 0.2 3593 -1
```



GEN08

Generieren einer Funktion durch Spline-Interpolation.

```
f # time size 8 a n1 b n2 c n3 d ...
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`a,b,c ...` - Funktionswerte.

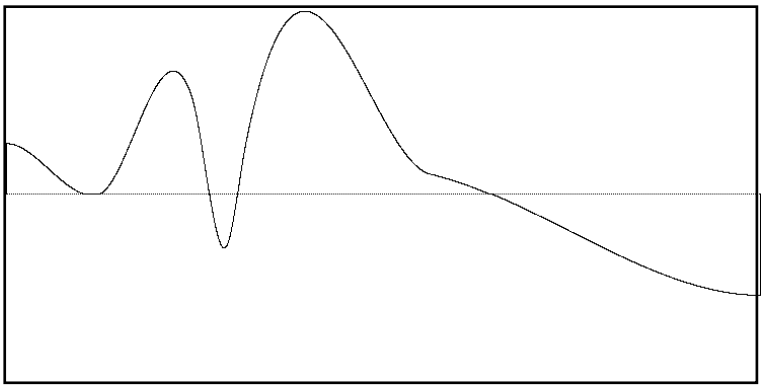
`n1,n2,n3 ...` - Anzahl der gespeicherten Werte zwischen den angegebenen Punkten. Die Länge darf nicht 0 sein, gebrochene Zahlen sind aber möglich. Die Summe aller `n` sollte `size` entsprechen.

Wenn p4 positiv ist (8), wird die Tabelle nach dem Einlesen normalisiert, d.h. alle Werte werden so skaliert, daß der Maximalwert 1.0 beträgt. Ein negativer Wert für p4 (-8) verhindert das Normalisieren.

GEN08 interpoliert zwischen den angegebenen Funktionswerte mittels Splines, d.h. die Punkte werden durch eine stetige Kurve verbunden.
Der Anstieg der Funktion an beiden Endpunkten ist 0.

Beispiel:

```
f5 0 8193 8 0.5 1000 0 1000 1 400 -0.5 200 0.5 2000 0.2 3593 -1
```



GEN09, GEN10, GEN19

Prozeduren zum Generieren von Wellenformen durch Überlagerung von Sinusfunktionen.

```
f # time size 9  pna stra phsa pnb strb phsb ...
f # time size 10 str1 str2 str3 str4 ...
f # time size 9  pna stra phsa dcoa pnb strb phsb dcob ...
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`pna,pnb ...` - Nummer des Teiltones in Relation zur Gesamtlänge der Tabelle. Darf nicht negativ sein. Gebroche Zahlen sind möglich (für nichtharmonische Teiltöne). Die Reihenfolge ist beliebig.

`stra,strb ...` - relative Stärke (Amplitude) der Teiltöne.
Negative Werte bedeuten eine Phasendrehung (180°).

`str1,str2 ...` - relative Stärke der ersten, zweiten usw. Harmonischen (für GEN10).

`phsa,phsb ...` - Phasenverschiebung der Teiltöne pna, pnb usw. in Grad

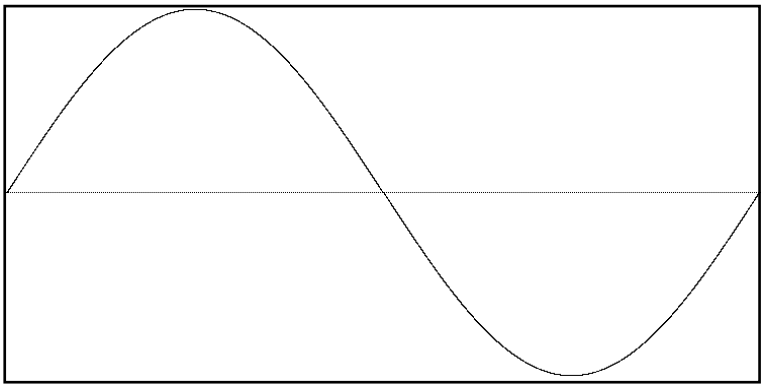
`dcoa,dcob ...` - Gleichspannungsoffset der Teiltöne pna, pnb ...
z.B. wird ein Ton mit der relativen Stärke 2 , Bereich [-2,2], in den Bereich [0,4] verschoben.

Wenn p4 positiv ist (9,10,19), wird die Tabelle nach dem Einlesen der kompletten Wellenform normalisiert, d.h. alle Werte werden so skaliert, daß der Maximalwert 1.0 beträgt.
Ein negativer Wert für p4 (-9,-10,-19) verhindert das Normalisieren.

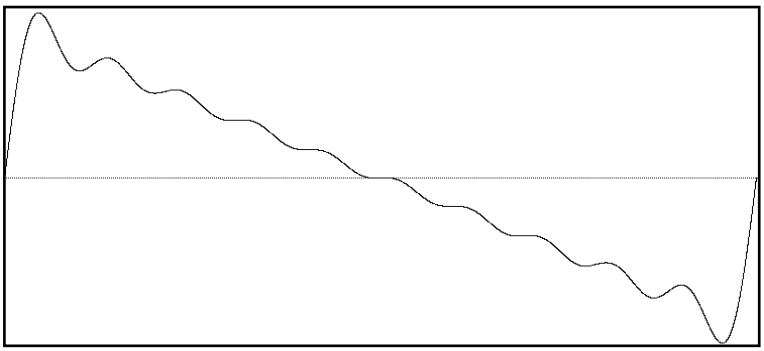
GEN10 stellt nur Harmonische (Grundton und Obertöne) zur Verfügung.
GEN9 erlaubt auch inharmonische Verhältnisse sowie Phasenverschiebungen.
GEN19 ermöglicht darüberhinaus die Addition von Konstanten (Gleichspannungsoffset), d.h. die Verschiebung der Funktion in y-Richtung.

Beispiele: √

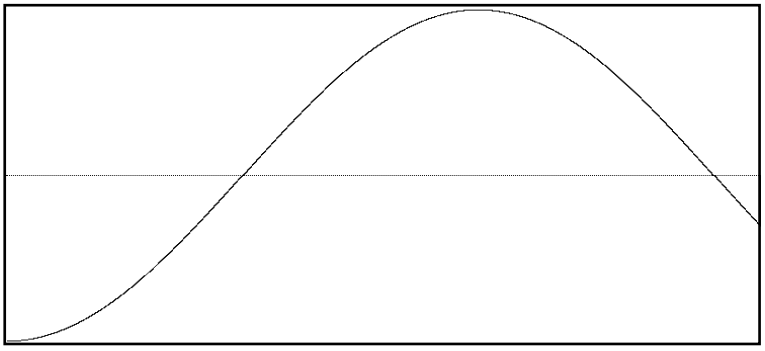
```
f6 0 8193 10 1
```



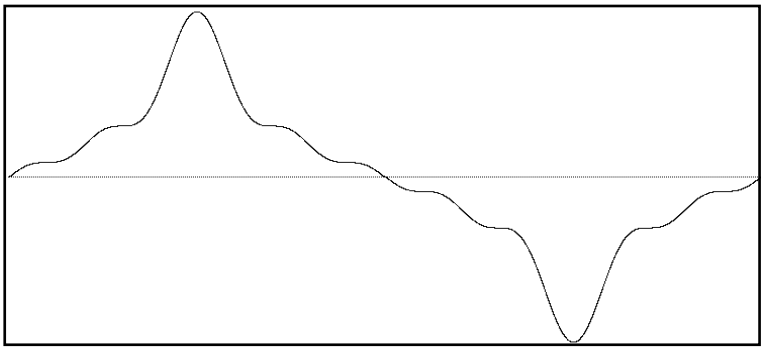
```
f7 0 8193 10 1 .5 .33 .25 .2 .166 .1428  .125 .111  .1
```



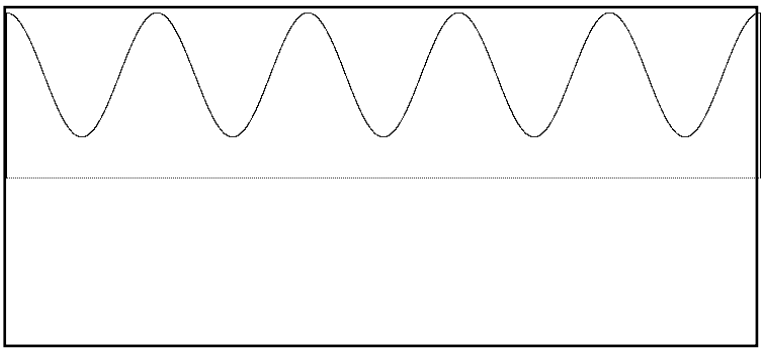
```
f8 0 8193 9  0.8 1 270
```



```
f9 0 8193 9  1 1 0   3 .33 180   5 .2 0  7 .1428 180   9 .111 0
```



```
f10 0 8193 19 5 0.3 90 0.5
```



GEN11

Prozedur zum Generieren von Wellenformen durch Überlagerung von Cosinusfunktionen.

```
f # time size 11 nh lh r
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`nh` - Anzahl der gewünschten Harmonischen. Muß positiv sein.

`lh` - (optional) Tiefste Harmonische, die verwendet werden soll. Kann positiv, negativ oder Null sein. Negative Harmonische werden in den positiven Bereich reflektiert - ohne Phasendrehung, da der Cosinus symmetrisch zur Nullinie verläuft. Die Angabe von `lh` ist optional, der voreingestellte Wert ist 1.

`r` - (optional) Multiplikator in der geometrischen Reihenbildung für die Amplitudenwerte.

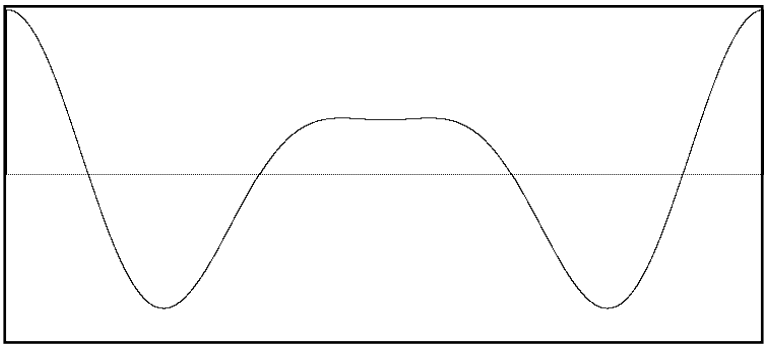
$$\sum_{k=lh}^{lh+nh} A_k r^{k-lh}$$

Die Amplitudenwerte folgen somit einer Exponentialfunktion. `r` kann positiv, negativ und gebrochen sein. Der voreingestellte Wert ist 1.

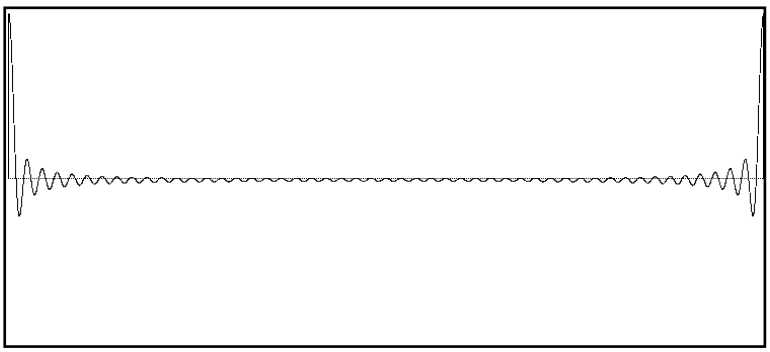
Wenn p4 positiv ist (11), wird die Tabelle nach dem Einlesen der kompletten Wellenform normalisiert. Ein negativer Wert für p4 (-11) verhindert das Normalisieren.

Beispiele:

```
f11 0 8193 11 2 2 0.5
```



```
f12 0 8193 11 50 1 1
```



GEN13,GEN14

Prozeduren zum Generieren von Polynomfunktionen aus Tschebyschew-Koeffizienten.
Diese Generatoren rufen GEN03 auf, um die Polynomfunktion zu zeichnen.

```
f # time size 13 xint xamp h0 h1 h2 ... hn
f # time size 14 xint xamp h0 h1 h2 ... hn
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`xint -` x-Intervall symmetrisch zum Nullpunkt [`-xint,+xint`].
Für Waveshaping ist der normale Wert 1.

`xamp` - relative Stärke eines Eingangssignals (Sinus) bei der das angegebene Spektrum erzeugt werden soll.

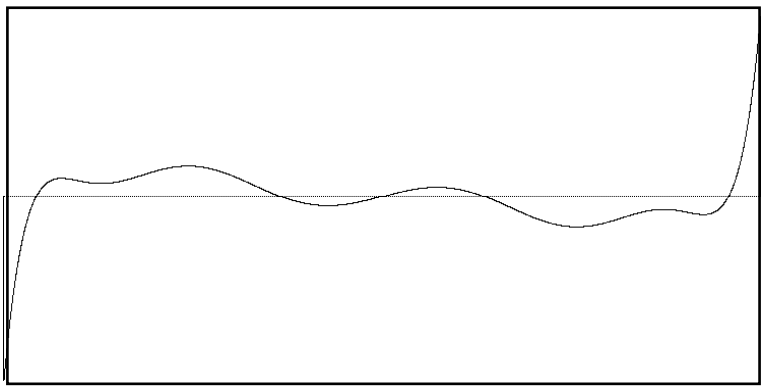
`h0,h1,h2, ... hn` - relative Amplitude der Teiltöne 0 (Gleichanteil),1,2 ... die entstehen, wenn ein Sinuston der Amplitude `xamp * int(size/2)/xint` durch die Tabellenfunktion verzerrt wird (Waveshaping).

GEN13 leitet Polynomkoeffizienten für GEN03 aus Tschebyschew-Polynomen erster Art ab.
GEN14 leitet Polynomkoeffizienten für GEN03 aus Tschebyschew-Polynomen zweiter Art ab.

Beispiele: √

```
f1 0 8193 13 1 1 0 2 0 3 0 2 0 1 0 1
```

Diese Werte generieren ein Polynom 9. Grades im Bereich [-1,+1], das beim Waveshaping aus einem Sinuston mit maximaler Amplitude 5 ungeradzahlige Harmonische mit der relative Stärke 2:3:2:1:1 erzeugt.



GEN17

Erzeugen von stufigen Funktionen.

```
f # time size 17 x1 a x2 b x3 c ...
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

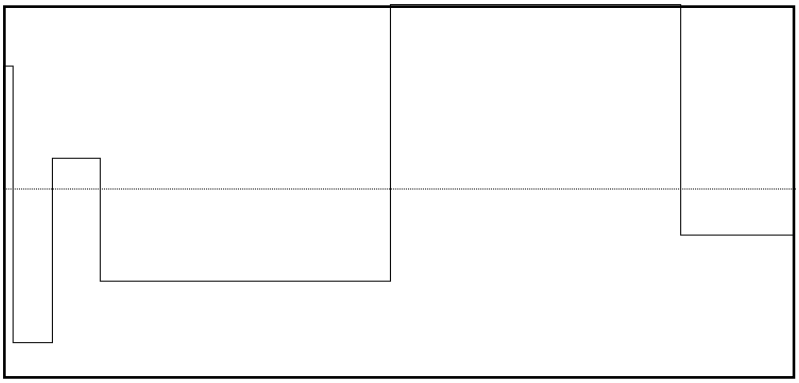
`x1,x2,x3 ... -` x-Werte in aufsteigender Ordnung (in Punkten). x1 = 0.

a,b,c ... - y-Werte an den jeweiligen x-Werte bis zum nächsten x-Wert. Der letzte y-Wert wird bis zum Ende der Tabelle gehalten.

Wenn p4 positiv ist (17), wird die Tabelle nach dem Einlesen der kompletten Wellenform normalisiert. Ein negativer Wert für p4 (-17) verhindert das Normalisieren.

Beispiel: √

```
f1 0 8193 17 0 8 100 -10 500 2 1000 -6 4000 12 7000 -3
```



GEN20

Erzeugen von Fensterfunktionen, z.B. für Spektralanalyse oder Granularsynthese.

```
f # time size 20 window max opt
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`window` - Typ des zu generierenden Fensters
1 - Hamming
2 - Hanning
3 - Bartlett (triangle)
4 - Blackman (3 - term)
5 - Blackman - Harris (4 - term)
6 - Gauss
7 - Kaiser
8 - Rechteck
9 - Sinc

`max` - bei Nichtnormalisierungf (p4 positiv)gibt dieser Wert das absolute Maximum des Fensters an.

opt - spezieller Parameter für Kaiser-Fenster, der angibt, wie „offen“ der Funtionsverlauf ist.
0 ergibt ein Rechteckfenster, 10 ein Hamming-ähnliches Fenster.

Beispiel: √

```
f1 0 1024 -20 9 1000
```

Dies erzeugt ein Gauss-Fenster mit einem Maximum von 1000.

GEN21

Erzeugen von wahrscheinlichkeitsverteilten Zufallstabellen
(siehe auch Zufallsgeneratoren in den Orchestra-Statements).

```
f # time size 21 type level arg1 arg2
```

`size` - Anzahl der Punkte - muß eine 2-er Potenz (+1) sein.

`type` - Wahrscheinlichkeitsverteilung
1 - Uniform
2 - Linear
3 - Triangular
4 - Exponential
5 - Biexponential
6 - Gauss
7 - Cauchy
8 - Positiv Cauchy
9 - Beta
10 - Weibull
11 - Poisson

`level` - bei Nichtnormalisierungf (p4 positiv)gibt dieser Wert das absolute Maximum an.
arg1, arg2 - nur bei Beta und Weibull nötig,
siehe entsprechende Parameter für Beta und Weibull in den Orchestra-Statements.

Beispiele: √

```
f1 0 128 -20 2 100
```

Dies erzeugt 128 linear verteilte Zufallswerte im Bereich 0 bis 100.

```
f2 0 1024 -20 9 500 0.1 0.5
```

Dies erzeugt 1024 Werte nach einer Beta-Verteilung, wobei es eine starke Häufung bei 0 und eine geringere bei 500 gibt.

GEN23

```
f # time size -23 “filename.txt”
```

Dieser Tabellengenerator liest eine Textdatei ein. Die einzelnen Werte müssen durch Leerzeichen, Tabs, neue Zeilen oder Kommata separiert sein. Alles hinter einem Semikolon bis zum Ende der Zeile wird ignoriert. Nichtnumerische Zeichen (Wörter, Buchstaben) werden ebenfalls ignoriert.