CDP Spectral STRETCH Functions

(with Command Line Usage)

Functions to Stretch the Time or Frequency Components of a Sound

(Names in brackets mean that these are separate programs. The others are sub-modules of STRETCH.)

STRETCH SPECTRUM
stretch/compress the frequency components of a sound in an inharmonic way

STRETCH TIME
stretch/compress a sound in time without changing the pitch

[STRETCHA]
Utility to Calculate *timestretch* factor relating to beats and tempo for use with STRETCH TIME

Technical Discussion
Stretching a Spectrum in Time / Stretching Frequency Components

STRETCH SPECTRUM – stretch/compress frequency components of *infile*

Usage

stretch spectrum mode *infile outfile frq_divide maxstretch exponent* **-d***depth*

Modes

1 Stretch above the *frq_divide*
2 Stretch below the *frq_divide*

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
frq_divide is the frequency above or below which spectral stretching takes place
maxstretch is the transposition ratio of the topmost spectral components
exponent specifies the type of stretching required (> 0)
-d*depth* stretch effect on source (from 0 (no effect) to 1 (full effect)

depth can vary over time

Understanding the STRETCH SPECTRUM Process

The STRETCH SPECTRUM process is a means to warp the spectrum of a sound. For example, a harmonic sound may become inharmonic, in a controllable way. (It is derived from CDP's former function SPECE.)

The STRETCH SPECTRUM process gives us control over the way in which we stretch the spectrum of a sound. It should be used for stretching as opposed to shifting the spectrum of a sound. The stretching operation begins at the *frq_divide* and proceeds upwards.

The main thing to appreciate regarding STRETCH SPECTRUM is that the stretch factor is scaled across the frequencies, beginning at 1 at the *frq_divide* (no stretch) and gradually increasing or decreasing until reaching *maxstretch* of the uppermost channel in which a frequency is found (not > Nyquist). Overall, it is rather like stretching or compressing a rubber sheet – the whole sheet is warped by the stretching process and the partials are thereby moved.

The scaling of *maxstretch* is linear (even increments) when *exponent* is an integer. If *exponent* is < 1.0, the stretch curve will rise quickly and then tend to even out. If *exponent* is > 1.0, the stretch curve will start to rise slowly and then gradually rise more quickly.

The stretching operation expands or contracts the frequency gap between partials. The **Technical Discussion** offers some supplementary observations about how to think about what this means.

Under this operation the frequency ratios between the partials of the original sound are not preserved. Thus, for example, a sound with harmonically related partials, e.g., 100, 200, 300, 400 Hz (common factor, 100) will end up with partials which are not harmonically related, e.g., 110, 213, 316, 419 Hz (no common factor). Such an inharmonic spectrum will normally be perceived as containing several pitches or (with a small stretch) strong spectral colouration. So the stretching operation usually increases inharmonicity and the timbral richness of a sound.

Use of *maxstretch*

Maxstretch specifies the amount of stretch to be applied to the partial found in the uppermost channel. The partial in the channel corresponding to *frq_divide* does not move (*maxstretch* = 1). Inbetween components will be moved by an amount corresponding to their relative position.

Special Note - components stretched beyond the Nyquist frequency (sr/2) are made to disappear so that the maximum component we will 'hear' will be the one marked 'X' in the above diagram. Hence a *maxstretch* value > 1 may be something of a fiction in practice because partials stretched beyond Nyquist actually disappear.

Musical Applications

The image of frequencies on a rubber sheet, referred to above, can help point us in the direction of the possible applications of this process. We can picture the frequencies drawn on the sheet with the lower frequencies at the bottom and the higher frequencies at the top. When we stretch or compress the bottom of the sheet, the frequencies get further apart (i.e., pushing the bottom frequencies lower) or closer together (higher). This will deepen or tighten the tone, as well as make it become more inharmonic at the same time.

Similarly, when the top of the sheet is stretched or compressed, the frequencies become further apart (higher: top ones pushed up) or closer together (this will lower and bunch up the high frequencies), with a similar increase in inharmonicity.

As ever, the results are greatly affected by the nature of the *infile*. Experiments with a vocal *infile* should reveal the way it acts on a sound in an easily perceptible manner. Probably the single most important parameter is *frq-divide*: where the split point is placed. The quickest way to determine the frequency range of a sound is to use a tuning fork. Trial and error (i.e., guessing) can also be a fairly quick way to get results. Charts of the frequency ranges occupied by various types of sound can also be instructive.

End of STRETCH SPECTRUM

STRETCH TIME – stretch/compress duration of *infile* without changing the pitch

Usage

stretch time 1 *infile outfile timestretch*
stretch time 2 *infile timestretch*

In Mode 2, the program calculates the length of the output only, and does not produce an *outfile*.

Parameters

infile – input analysis file made with PVOC
outfile – output analysis file
timestretch single value or the name of a *time timestretch* breakpoint file of values. *Timestretch* is used as a multiple.

Understanding the STRETCH TIME Process

The key here is to time stretch the sound without altering the pitch. This can be used either to lengthen or compress the duration. Formerly SPECSTR, meaning 'spectrum time-stretching', the internal prolongation of spectral data is itself subject to temporal control according to the amount of stretch specified at different times.

The fact that the time can be set in the breakpoint file provides an important tool. For example, it can be used to retain the ability to recognise the source of the sound. We often can identify a sound according to the way it begins, its 'attack transient'. This can be kept unchanged by giving a *timestretch* of 1 for this part of the sound. Then the stretching can begin, so we retain the recognisability of the sound while stretching it in time. Retaining recognisability is often a very important compositional issue.

The breakpoint file sets *time* and *timestretch* factors in such a way that *change is gradual* (interpolated) between different times with different stretch factors. For example:

```
[time stretch]
0.0   1.0
1.0   1.0
2.0   2.5
```

During the 1st second of the sound, there will be no change. During the 2nd second of the sound, the spectrum will be gradually stretched to 2.5 times its original length.

Two different stretch factors cannot be assigned to the same time. At least a slight difference in time is required. So, for virtually instantaneous changes in the stretch factor, use values such as:

```
[time stretch]
2.0   2.5
2.01  1.5
```

Musical Applications

The stretching can be imagined in a quite literal way. This is illustrated by time stretching a sound which contains strong beat patterns. As the stretch gets longer, one really begins to hear the pulse slow down.

Very long time stretches might best be done in stages rather than all at once. If the whole sound is not required, one way to proceed is to cut out a portion after each time stretch (say a stretch factor between 5 and 10) with SPEC CUT and then restretch that portion. Otherwise, time is wasted as the sound gets longer and longer.

End of STRETCH TIME

STRETCHA – Utility to calculate *timestretch* factor relating to beats and tempo for use with STRETCH TIME

Usage

stretcha -c -d*dur* | **-f***sndfile* | (**-b***beats* **-t***tempo*) **-T***outtempo*
stretcha -s -d*dur* | **-f***sndfile* | (**-b***beats* **-t***tempo*) **-D***outdur* | (**-B***outbeats* **-T***outtempo*)
stretcha -s -t*intempo* **-T***outtempo*

Example command lines (in these examples 'bdt.wav' is 4.9 seconds long):

Mode **1**. **-c** flag:
(1) **`stretcha -c -d8 -T90`** [Result: 12 beats (8 seconds fills 12 beats at MM = 90]
(2) **`stretcha -c -fbdt.wav -T108`** [Result: 8.819878 beats (the input soundfile fills 8.819878 beats at MM=108]
(3) **`stretcha -c -b8 -t60 -T72`** [Result: 9.6 beats (8 beats at MM=60 becomes 9.6 beats at MM=72]

Mode **2**. **-s** flag:
(1) **`stretcha -s -d8 -D12`** [Result: *stretchfactor* = 1.5 (use to make 8 seconds fill 12 seconds]
(2) **`stretcha -s -d8 -B4 -T50`** [Result: *stretchfactor* = 0.6 (use to make a specified duration fill 4 beats at MM=50]
(3) **`stretcha -s -fbdt.wav -D12`** [Result: *stretchfactor* = 2.449014 (use to make a 4.9 second soundfile fill 12 seconds]
(4) **`stretcha -s -fbdt.wav -B4 -T50`** [Result: *stretchfactor* = 0.977605 (use to make a 4.9 second soundfile fill 4 beats at MM=50]
(5) **`stretcha -s -b8 -t84 -D12`** [Result: *stretchfactor* = 2.449014 (use to make 8 beats at MM=84 fill 12 seconds]
(6) **`stretcha -s -b8 -t84 -B6 -T96`** [Result: *stretchfactor* = 0.5625 (use to make 8 beats at MM=72 fill 6 beats at MM=96]

Mode **3**. **-s** flag:
`stretcha -s -t66 -T120` [Result: *stretchfactor* = 0.550 (use to convert a passage moving at tempo MM=66 to one moving at tempo MM=120]

Modes (as illustrated by the above command lines)

Mode **1**: (three input options ⇒ a specified tempo)
-c – takes as inputs: a duration OR a soundfile OR (a number of (possibly fractional) beats and a tempo) and also an output tempo

1. IN: **-d***dur* OUT: **-T***outtempo* ⇒ displays how many beats there will be at the new tempo

2. IN: **-f***sndfile* OUT: **-T***outtempo* ⇒ displays how many beats the length of the input soundfile will fill at the new tempo
3. IN: **-b***beats* **-t***tempo* OUT: **-T***outtempo* ⇒ displays how many beats there are at the new tempo

Mode **2**: (three input options ⇒ to a specified duration OR to a number of beats at a given tempo)
-s – takes as inputs: a duration OR a soundfile OR a number of (possibly fractional) beats and a tempo and also an output duration OR a number of (possibly fractional) output beats and an output tempo

1. IN: **-d***dur* OUT: **-D***outdur* ⇒ displays the *stretchfactor* needed to change one duration to the other
2. IN: **-d***dur* OUT: **-B***beats* **-T***tempo* ⇒ displays the *stretchfactor* needed to convert an input duration to a given number of beats at a given tempo
3. IN: **-f***sndfile* OUT: **-T***outdur* ⇒ displays the *stretchfactor* needed to convert a given input soundfile to a specified duration
4. IN: **-f***sndfile* OUT: **-B***beats* **-T***tempo* ⇒ displays the *stretchfactor* needed to convert a given input soundfile to a given number of beats at a given tempo
5. IN: **-b***beats* **-t***tempo* OUT: **-T***outdur* ⇒ displays the *stretchfactor* needed to convert a given number of beats at a given tempo to a specified duration
6. IN: **-b***beats* **-b***tempo* OUT: **-T***outtempo* ⇒ displays the *stretchfactor* needed to convert a given number of beats at a given tempo to specified number of beats a specified tempo

Mode **3**: (input tempo ⇒ output tempo)
-s – takes as inputs: an input tempo and an output tempo;

1. IN: **-t***tempo* OUT: **-T***tempo* (displays the *stretchfactor* to change from the first tempo to the second)

Parameters

dur – an input length of time in seconds
sndfile – an input soundfile
beats – an input number of (possibly fractional) beats
tempo – an input tempo at which the beats occur
outdur – an output length of time in seconds
outbeats – an output number of (possibly fractional) beats
outtempo – an output tempo

Understanding the STRETCHA Function

STRETCHA is a utility program for use with STRETCH TIME. It takes a bit of calculating to work out how to make a soundfile fit specific numbers of possibly fractional beats. STRETCHA does the work for you. When a soundfile is named as in input, its duration is automatically determined. It can also handle durations and beats as inputs or as outputs to add extra flexibility.

Note that, if you do not have the environment variable CDP_SOUND_EXT SET, you will have to include the .wav extension with the soundfile name. The commandline environment also enables you to write the output to a file rather than just see it displayed on the screen. This is done with the redirection symbol **>** as in this

example command line:
`stretcha -s -fbdt.wav -D8 >strfac.txt.`

The stretch factor is then used with the CDP program STRETCH TIME. While most of the options are fairly straightforward, note that some possibilities are not covered directly but require two steps. The three **-c** options of Mode **1**, for example, which convert to a specified tempo, display only "number of beats at new tempo" – it does not actually give a *stretchfactor*. It gives the number of beats that will result. You can then use this information to fill in the 'beats' parameter value in Mode **2** options to get the *stretchfactor* to convert the input beats and tempo to a specified duration (option 5) OR tempo (option 6).

> To illustrate this two step process, if you input a *duration* of 4 seconds and an *outtempo* of MM = 90: Mode **1** option 1 (`stretcha -c -d4 -T90`), the program tells you that 6 crotchets will result. Now you can use that information to calculate the required *stretchfactor* to fill a specified duration: Mode **2**, option 5 (`stretcha -s -b6 -t90 -D10`), yielding a stretch factor of 2.5. Thus 4 seconds changes to 6 crotchets at MM = 90, and this will fill 10 seconds with a stretch factor of 2.5. You have thus gone from duration to duration while specifying a specific number of beats at a given tempo. What you need to do depends on the nature of the material with which you start.

It may also be useful to know the actual duration that results from a given number of beats at a given tempo. For example, if you want 4 beats at MM = 72 to be changed to 7.5 beats at MM = 108, you will have to know what duration the later produces (to use Mode **2** option 5). The program doesn't provide this facility, but it is easily calculated as **`numbeats * 60.0/tempo`**, which in this case is `7.5 * (60.0/108)`, giving 4.1666 seconds. Then you can use: `stretcha -s -b4 -t72 -D4.1666`, giving a stretch factor of 1.249980.

You can also calculate the duration of the input portion of beats at a tempo and then use the input and output duration option (Mode **2** option 1). Thus `4 * (60.0/72) = 3.3333`: `stretcha -s -d3.3333 -D4.1666` gives a stretch factor of 1.249980.

Musical Applications

It may be helpful to separate out all the options. This reveals how easy it is to use. In the examples, the soundfile *bdt.wav* is used. Its duration is 4.90 seconds, as shown by `dirsf bdt.wav`. I've used numbers that enable you to see and compare the results quite easily.

> Mode **1**:
>
> **1. Inputs: duration and output tempo, Output: number of beats**
>
> `stretcha -c -ddur -Touttempo`
> Example: `stretcha -c -d4 -T120`
> Result: number of beats at new tempo = 8.0 crotchets
>
> **2. Inputs: soundfile and output tempo, Output: number of beats**
>
> `stretcha -c -finsnd.wav -Touttempo`
> Example: `stretcha -c -fbdt.wav -T120`
> Result: number of beats at new tempo = 9.799864 crotchets

3. Inputs: beats & tempo and output tempo, Output: number of beats

```
stretcha -c -bbeats -ttempo -Touttempo
```
Example: `stretcha -c -b4 -t60 -T120`
Result: number of beats at new tempo = 8.0 crotchets

Mode **2**:

1. Inputs: duration and output duration, Output: stretch factor

```
stretcha -s -ddur -Doutdur
```
Example: `stretcha -s -d4 -D8`
Result: stretch factor is 2.0

2. Inputs: duration and output beats & tempo, Output: stretch factor

```
stretcha -s -dduration -Bbeats -Ttempo
```
Example: `stretcha -s -d4 -B8 -T90`
Result: stretch factor is 1.333333

3. Inputs: soundfile and output duration, Output: stretch factor

```
stretcha -s -finsnd.wav -Doutdur
```
Example: `stretcha -s -fbdt.wav -D8`
Result: stretch factor is 1.632676

4. Inputs: soundfile and output beats and tempo, Output: stretch factor

```
stretcha -s -finsnd.wav -Boutbeats -Touttempo
```
Example: `stretcha -s -fbdt.wav -B8 -T90`
Result: stretch factor is 1.08845

5. Inputs: beats & tempo and output duration, Output: stretch factor

```
stretcha -s -bbeats -ttempo -Doutdur
```
Example: `stretcha -s -b4 -t60 -D8`
Result: stretch factor is 2.0

6. Inputs: beats & tempo and output beats & tempo, Output: stretch factor

```
stretcha -s -bbeats -ttempo -Boutbeats -Touttempo
```
Example: `stretcha -s -b4 -t60 -B8 -T90`
Result: stretch factor is 1.333333

Mode **3**:

1. Input: tempo, Output: tempo

```
stretcha -s -ttempo -Touttempo
```
Example: `stretcha -s -d60 -D90`
Result: stretch factor is 0.666667

Historical Note

I (AE) asked Trevor to write this program several years ago in order to help create passages of music using regular beat patterns. The source code has been lost for all these years, but an enquiry from Arthur Green, who found the program in his directory of CDP programs and discovered that it would not work on a 64-bit PC, led Richard Dobson to search out the source and recompile it for us.

End of STRETCHA

Technical Discussion of Stretching in the Spectral Domain

To help visualise what might be happening with the spectrum, here are some observations about 'interval', 'frequency' and 'partials':

> Bear in mind that the partials of a tone are closer together in pitch the higher their frequency. This is because pitches are the logarithm of frequency. This means that going up in pitch by equal amounts, we **add intervals** – such as going up by one octave – but we **multiply frequencies** – times 2 to go up one octave.
>
> For example, going up one octave from 400 Hz takes us to 800 Hz, but going up an equivalent one octave from 4000 Hz takes us to 8000 Hz, i.e., 10 times as far in terms of frequency.
>
> Looking at this the other way round, going up by 400 Hz, e.g., from 400 Hz to 800 Hz, takes us up an octave; but going up 400 Hz from 4000 Hz to 4400 Hz takes us up only a very small interval.
>
> The harmonic partials, being integer multiples of the fundamental, are equally spaced in **frequency** (e.g., 400 (400 * 1), 800 (400 * 2), 1200 (400 * 3), 1600 (400 * 4) – all are 400 Hz apart). As a result, the pitch-intervals between them get smaller and smaller as we rise in frequency.

The stretching operation therefore expands or compresses the frequency (i.e., partial) spacings. But in the higher end of the spectrum, it is useful to remember that the intervals involved will be very small. The aural effect of the stretching process will relate in part to the inharmonic relationships created (or altered) among the intervals defined by the frequencies.

End of Technical Discussion

Last Updated 18 June 2015
Documentation: Archer Endrich, revised R. Fraser