Music Sequencing with Dynamic Transposition

TECTRAL*

November 9, 2025

Music Sequencing with Dynamic Transposition

Summary

The purpose of this software is to read music scores in Dynamic Transposition notation (PDF) and convert them into WAV files. This simplifies music sequencing and uses a more harmonious musical scale.

Contents

- Method
- Implementation
- Files
- Usage
- References

Method

In just intonation scale frequencies of musical tones are given by rational fractions of the base frequency. For example, if the base frequency of note A in 4-th octave is freq(A4)=440Hz then frequencies of related harmonic intervals, corresponding to C#5 and E5 in octave 5 will be: `5/4*440=550` and `3/2*440=660` respectively.

Corresponding minor chord frequencies will be: 440 (A4), `440*6/5=528` (C5), and 660 (E5). Thus, the frequencies for A4, C5, C#5, E5 are:

```
440 A4
528 C5
550 C#5
660 E5
```

*Tectral.

This is in contrast to the standard Chromatic scale where the corresponding frequencies are approximately:

```
440.00 A4
523.25 C5
554.37 C#5
659.26 E5
```

From this comparison it is clear that the scale based on rational fractions produces musical intervals with better resonating frequencies. This makes music more harmonious when plaid in such scale. We will refer to the scale based on rational fractions (i.e. fractions of integers) as *rational scale*.

Acoustic instruments can be tuned in rational scale but then they can only play in one octave, i.e. in a single key. Going to a different octave will change the intervals, so the same chords will sound differently. This is the reason why rational scale is not commonly used.

That's where the method of dynamic transposition comes to the rescue. This technique makes it possible to play in rational scale in different keys and different octaves using computer generated tones. All one needs to do to switch from one key to another is to shift the frequency of all notes by a corresponding rational fraction. For example, the frequencies of the major chord above in key A and then in key D will be:

```
Key A: 440, 550, 660
Key D: 6/5*440, 6/5*550, 6/5*660
```

To play in key D we multiplied all three frequencies of key A by 6/5 which corresponds to the frequency shift from A4 to D5. This is akin to transposition which can be used on electronic instruments. Here transposition is done on-the-fly, switching from one key to another while playing the composition, thus the name *dynamic*. Also, the shift in frequency is done by rational numbers which produces more harmonious intervals than in conventional Chromatic scale.

Implementation

A music score in rational scale for several instruments can be written in a simple text file. Two notations can be used: explicit and implicit. Explicit format uses numbers and implicit - letters. For example, explicit format will look like this:

```
5:3 1:1:2 5:4:2 3:2:2
```

which instructs three instruments to play a chord of three notes at frequencies:

```
instrument 1 (1:1:1) plays at frequency: base_freq*(5/3)*(1/1*2^2)
instrument 2 (5:4:2) plays at frequency: base_freq*(5/3)*(5/4*2^2)
instrument 3 (3:2:2) plays at frequency: base_freq*(5/3)*(3/2*2^2)
```

where "`base_freq`" is the base frequency which is the same for the whole composition and is selected at the beginning (e.g. 440Hz).
The fraction given at the beginning of each row (5:3 in our example) is used to multiply the base frequency of the composition and make it base frequency of that row. In this example:

```
base_freq_row = base_freq * 5/4
```

Essentially the base frequency was *transposed* to become the frequency of that row and all notes on that row will use it to set their frequencies. Thus, the fraction in the first column determines *dynamic transposition* since it changes the base frequency on-the-fly during the composition.

All other numbers in the row determine frequencies of the notes plaid by respective instruments. Notation "a:b:c" defines the factor that the base frequency of the row is multiplied by, that is

```
note_freq = base_freq_row * a/b * 2^c
```

where

```
a is the numerator,
b - the denominator, and
c - the octave (notation 2^c means raising 2 to power c).
```

We can also substitute letters for fractions for convenience. For example, that line can also be written in implicit format as

```
A C2 E2 G2
```

where number 2 on the right of each letter represents the octave and the fractions are assigned to letters as:

```
A=5:3
C=1:1
E=5:4
G=3:2
```

The letters can be arbitrary, but in this case we chose them to correspond to key C. So, the above line (A C2 E2 G2) corresponds to chord `C-E-G` transposed to key A, which will result in chord `A2-C#3-E3` actually plaid. This way one can play all keys using *only* the notes in key C which is another advantage of this method.

For example, if we want to repeat the sequence of main triad notes in four different keys: C, Am, F, G (i.e. three major and one minor - Am) we can do it as this:

```
C C2 E2 G2
A C2 Eb2 G2
F C2 E2 G2
G C2 E2 G2
```

where the first letter in each row corresponds to the frequency shift for the entire row (i.e. the key plaid). Note that we did not need to change the notes C,E,G for any of major keys, and only introduced Eb=6/5 (i.e. E-flat) for a minor key Am. The resulting sequence is much simpler than it would be in a conventional music score.

The complete composition may look like this:

```
; Definitions:
Am = 5:6
C  = 1:1
E  = 5:4
Eb = 6:5
G  = 3:2
; Score:
C C2 E2 G2
Am C2 Eb2 G2
F C2 E2 G2
G C2 E2 G2
```

Thid will play chords in keys: C, Am, F, and G. Note that fraction `5:6` was labeled as `Am` to signify that this is a minor chord. This was done mainly for clarity and would work with any other label. The semicolon (;) is used for comments.

Time duration for each row is referred to as *tick* and is set at the beginning of the score file which is called *preamble*. Each tick can be further subdivided into smaller time intervals referred to as *pics*. The number of picks in a tick is also defined in the preamble.

Several ticks comprise a *beat*, and several beats comprise a *measure* where corresponding numbers (i.e. ticks-per-beat and beats-pre-measure) are also set in the preamble. These parameters are used when one wants to output only specific measures from a composition. This can be done by supplying *clip* parameters for the DT compiler (run DT compiler with –help option to see more information).

To make note duration longer than one tick one should put '+' in the next line corresponding to the position of respective instrument. For example, in this chord:

```
A B2 C2 D2
A +  -  -
```

instrument 1 plays note B2 for the duration of two ticks (two consequtive lines) and the rest of instruments play respective notes (C2,D2) for one tick only. The dash symbol '-' is used to indicate pause for respective instrument.

To make notes with a duration less than one tick (i.e. several picks) one can use this notation:

```
A -     C2,2
```

```
A B2.1  D2
```

In this case note `B2.1` starts one pick before the tick corresponding to the line where this note is located and note `C2,2` starts two picks before the tick corresponding to the next line (i.e. before D2).

There are also special lines to set the volume (amplitude) and panning for each note. These lines begin with special tags: *amp* for amplitude and *pan* for pan. Sample score files supplied with this package provide examples of their usage. In the examples file-extention `dtt` is used for explicit and `dtm` for implicit file formats.

The source codes included in this package are written in GO language. They produce the DT-executable that compiles dtm or dtt files into MIDI or SCound SCO files. The supplimentary Unix shell scripts are used to run the DT-compier and convert the MIDI/SCO file to WAV or MP3 file.

Files

- On Tectral
- On GitLab

The files in this package include the source files and build scripts for dynamic transposition (DT) compiler as well as examples of score files and compiled compositions.

- Build files
  - `src/Makefile` - Unix make file to build the DT compiler executable named `dtc` that compiles dynamic transposition score into MIDI file or CSound score file.
  - `src/make_sco.sh` - Unix script that compiles dynamic transposition file (dtm-file) into CSound score file and calls CSound engine to convert it to a WAV file using CSoud orchestra file.
  - `src/make_midi.sh` - Unix script that converts dynamic transposition score file (dtm-file) into MIDI file and calls fluidsynth to convert the MIDI file to WAV file, or play it using system soundfonts.
  - `src/make_drums.sh` - Unix script that converts dynamic transposition score file (dtm-file) into MIDI file with percussion track and calls fluidsynth to convert MIDI file to WAV file, or play it using system soundfonts.
- Source files of DT compiler
  - `src/def.go` - declarations of global variables and some common functions.
  - `src/compile.go` - main function and routines to write output.
  - `src/score.go` - functions and data structures to read the dtm file and create a sequence of notes in rational scale.
  - `src/frac.go` - structures and functions to manipulate fractions.

- Documentation
  - `man/dtc.md` - Introduction to Dynamic Transposition Method (this file)
  - `man/dtc.pdf` - Introduction to Dynamic Transposition Method
  - `man/dtm.pdf` - Paper on Dynamic Transposisition Method
  - `man/dtn.md` - Introduction to Dynamic Transposition Notation
  - `man/dtn.pdf` - Introduction to Dynamic Transposition Sequencer Notation
  - `man/dtn.mp4` - Introductory Video to Dynamic Transposition Notation
- Score Samples
  - `man/canon.dtm` - dynamic transposition score of Pachelbel's Canon in D written with letters and rendered with soundfonts.
  - `man/canon.dtt` - dynamic transposition score of Pachelbel's Canon in D written with fractions and rendered with soundfonts.
  - `man/canon.mp3` - MP3 file produced by compiling cannon.dtm file with DT compiler.
  - `man/drums.dtt` - example of using drum track in DT score wiht soundfonts.
  - `man/drums.mp3` - MP3 file produced by compiling drums.dtt file with DT compiler.
  - `man/avemaria.dtm` - dynamic transposition score of Ave Maria song written with letters and rendered with CSound.
  - `man/avemaria.orc` - CSound orchestra file.
  - `man/avemaria-head.sco` - header for CSound score file.
  - `man/avemaria.mp3` - MP3 file produced by compiling avemaria.dtm file with DT compiler.

Usage

Building DT compiler

- Install go language compiler.
- To compile `dtc` executable on Linux change to `src` directory and run `make` command.

Editing DTM/DTT files

To edit DTM/DTT files on Linux you can use vim editor with syntax highlighting set to `asn68k`. In the score part use tabs instad of spaces to separate columns for respective instruments (see manual in `man` directory).

Using DT compiler

The DT compiler executable by itself converts dynamic transposition scores into MIDI or SCound files. To produce actual audio files, such as WAV or MP3 one can use different tools, such as Fluidsynth, or FFmpeg on Linux. To simplify this we provide two Bash scripts: `make_midi.sh` to create a MIDI file and then produce a WAV file with `fluidsynth` and `make_sco.sh` to create a CSound score file and then compile it into a WAV file with CSound.

Examples below show how to compile dynamic transposition formats, DTM (using letters) or DTT (using numbers) into WAV or MP3 files using MIDI or CSound isntruments on Linux.

Compiling to MIDI To compile a DTM or DTT file to MIDI file a GoMIDI go-library is used by the DT compiler. The produced MIDI file can then be converted to WAV file with FluidSynth on Linux. The `make_midi.sh` script provided in this package will run the DT compiler and convert MIDI to WAV. For example this command

```
sh make_midi.sh canon.dtm
```

will convert `canon.dtm` file into `canon.mid` file and then call fluidsynth to produce `canon.wav` file. One can also play the MIDI file directly with fluidsynth.

To include tracks with percussion instruments into MIDI file one can use use example `drums.dtt` provided in this package which can be compiled with `make_drums.sh` script as:

```
sh make_drums.sh drums.dtt
```

which will produce the drums.mid file and convert it to drums.wav by calling fluidsynth engine.

To convert WAV file to MP3 on Linux one can use the FFmpeg utility.

Compiling to CSound To compile a DTM or DTT file to CSound sco-file one can use `make_sco.sh` file provided with this package. For example, to compile the Ave Maria example score one can use this command:

```
sh make_sco.sh avamaria.dtm
```

This will produce a CSound score file `avemaria.sco` and then call csound engine to convert it to `avemaria.wav` file using CSound orchestra file `avemaria.orc` and a header file `avemaria-head.sco` file which are included in this package.

Compilation options Sometimes it can be useful to compile only certain measures from the score. For that one can use special compiler options. For example, to compile a midi file with two measures starting with the third measure, one can invoke the compiler with this command:

```
dtc -m 3 -n 2 score.dtm output.mid
```

or the respective Bash script as:

```
sh make_midi.sh score.dtm 3 2
```

If the first measure starts after a few introductory beats which one wants to skip one can indicate this with the `-s` option, like this:

```
sh make_midi.sh score.dtm 3 2 1
```

This will skip one introductory beat. That is, measures will be counted from the second beat in the compisition.

References

- Dynamic Transposition Method: homepage (tectral.com/dtm, PDF).
- Dynamic Transposition Method: paper on Dynamic Transposition method (tectral.com/pub/dtm.pdf).
- Dynamic Transposition Sequencer Notation (PDF, MP4): Introduction to Dynamic Transposition sequencer notation (tectral.com/pub/dtn.pdf).
- DTM on GitLab.
- GO compiler for compiling DT source codes.
- GoMIDI: Go libraries for MIDI (github.com/gomidi).
- CSound for coverting CSound SCO files into WAV files.
- FluidSynth for converting MIDI files into WAV files.
- FFmpeg utility for converting from WAV to MP3 files.
- Unix shell scripts for executing DT compiler, CSound, and FluidSynth.