Microtonal music with ABC (**microabc** tutorial)

Hudson Lacerda <hfmlacerda@yahoo.com.br>

August 29, 2010

Contents

1 Introduction

How to make microtonal music using a computer? How to obtain a score with a special, customised, microtone notation? How to get an audible "previewing" of a microtonal composition? Is all that feasible today with 100% free software? The answer for the last question is yes, thanks to some ABC music notation softwares. Answers for the other questions will be addressed through this text. **microabc** was written to make it easy to get microtonal music scores and MIDI files with ABC tools.[1]

2 Software requirements

First, you will need to obtain some pieces of software:[2]

- Get the (free/ libre/ open source) ABC software tools:[3]
  - **abcpp**, the ABC preprocessor.
  - **abc2midi**, an ABC to MIDI converter.
  - **abcm2ps**, a music typesetter that generates PostScript files.
- Get a program to compute musical scales:
  - **Scala** (free of charge).
- Get a PostScript interpreter and previewer:
  - **ghostscript**.
  - **ghostsview**.
- Get a MIDI player with Standard MIDI Tuning support:
  - **timidity++**.
- And do not miss **microabc**, of course!

You should be aware that **abcpp** must be compiled with sufficient room for the macros generated by **microabc**. Find the definition of MAX_MACROS in the source code and (re)define it as 512 or so,[4] then compile the program. In order to work with "normal" versions of **abcpp**, several samples in this tutorial use a limited pitch range.

The sample commands in this tutorial were tested on a Debian GNU+Linux operating system. For other systems, they may need a few adaptations.

3 Understanding the basic concepts

One needs to know a set of concepts, in order to get started to **microabc**.

[1] This text is focused on presenting some examples, it is not comprehensive and it does not abord details of **microabc** commands and options. Please read also the reference file microabc.html (or microabc.txt) found in the **microabc** package (and also online at http://hudsonlacerda.webs.com/abc/microabc.html).

[2] URIs for the sites from which the programs can be obtained are provided at the end of this document.

[3] Tip for Windows users: download the "ABC command line tools for Windows" package from http://abcplus.sourceforge.net/#abctools

[4] This is already done in **abcpp** version 1.4.2. For **abcpp** version 1.3.2, the line 43 of abcpp.c may be changed to, for instance:

```
#define MAX_MACROS 512 /* # of #defined macros */
```

3.1 ABC and microtones

To use **microabc**, a basic knowledge of ABC music notation is required. ABC is a music notation system using characters, originally devised by Chris Walshaw. There are several computer programs to convert ABC to music scores and MIDI files. ABC is easy – you can learn to use it in a short time.[5]

There are a few ABC programs with support for microtonalism. **abcm2ps** and **abc2midi** provide rich features to get microtonal PostScript scores and MIDI files.[6]

Both **abcm2ps** and **abc2midi** represent microtonal pitches with a fraction associated to an accidental. It is assumed that the fraction is a fraction of a conventional tempered semitone. For instance, ^1/2C is the pitch C raised by 1/2 semitone, while _14/100E is the pitch E lowered by 14 cents.

With **abc2midi**, one can temper the fifth and the octave for pitches with conventional accidentals. That way, one can use a scale with up to 35 pitches per octave (using simple and double flats and sharps).[7] The following ABC example uses that feature.

```
X:1
%%maxshrink 1
%%continueall
T:microabc tutorial
T:abc2midi temperamentlinear
M:4/4
L:1/4
Q:1/4=90
K:C
%---------- Pythagorean tuning ----------------------------
%%MIDI temperamentlinear 1200.0 701.955001
[V:1] G A G G | G2 z2 ||
[V:2 merge] E F E D | E2 x2 ||
[V:3 merge] C C C B, | C2 x2 ||
%---------- 1/4-comma meantone temperament -----------------
%%MIDI temperamentlinear 1200.0 696.578428
[V:1] G A G G | G2 z2 ||
[V:2] E F E D | E2 x2 ||
[V:3] C C C B, | C2 x2 ||
%---------- 22-equal divisions of octave -------------------
%%MIDI temperamentlinear 1200.0 709.090909
[V:1] =C _D ^B, ^C | =D _E _F ^D | =E2 z2 ||
[V:2] Z3 || [V:3] Z3 ||
%---------- Standard 12-equal divisions of octave ----------
%%MIDI temperamentnormal
[V:1] ^E =F ^D _E | [C=EG]3 z || [V:2] Z2 || [V:3] Z2 ||
```

[5] In ABC, notes are represented by letters A–G, and rests are represented by z. C is the middle C; notes of the upper octave are written in small letters (c d e...). Additional commas (,) or apostrophes (') render the notes of the lowers or upper octaves. Durations are represented by appending multipliers: C3 lasts three times more than C, while A/4 lasts a half of the duration of A/2, which duration is a half of duration of A. The length of a note with no multipliers is defined by an instruction such as L:1/4, which sets the base note length as a quarter note. Accidentals are represented by: __ (double flat), _ (flat), = (natural), ^ (sharp) and ^^ (double sharp). They are written left side the note letters as in ^C. A tune is started by a line with X: followed by a number (for indexing purposes). The field T: defines the music title. The time signature is set with M: (e.g. M:6/8). A line as K:F#m defines the key signature (here, F-sharp minor) and closes the "tune header"; the following lines are the music notes. Notes can be assigned to a voice specified with V:identifier. The tune ends with a blank line. A good introduction and reference for ABC is "Making Music With ABC Plus" by Guido Gonzato. Visit: http://abcplus.sourceforge.net/#ABCGuide.

[6] **jcabc2ps** and **BarFly** provide some (limited) support for microtonalism. Also, **tclabc** includes a program named **tkabc**, which is a graphical interface to write ABC files – **tkabc** supports the same quarter-tone accidentals as **abcm2ps**.

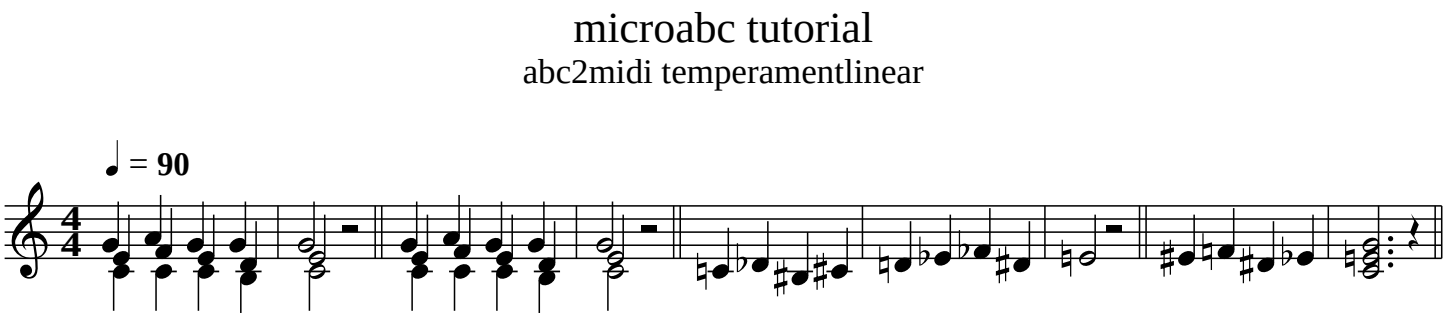
[7] **abc2midi** (since 2006-09-26) supports temperaments of the linear class, thanks to the command %%MIDI temperamentlinear octave_cents fifth_cents, where octave and fifth sizes are given in cents.

To generate a MIDI file, save the ABC code above as temperlin.abc and run the command:

```
abc2midi temperlin.abc
```

The PostScript score is generated with:

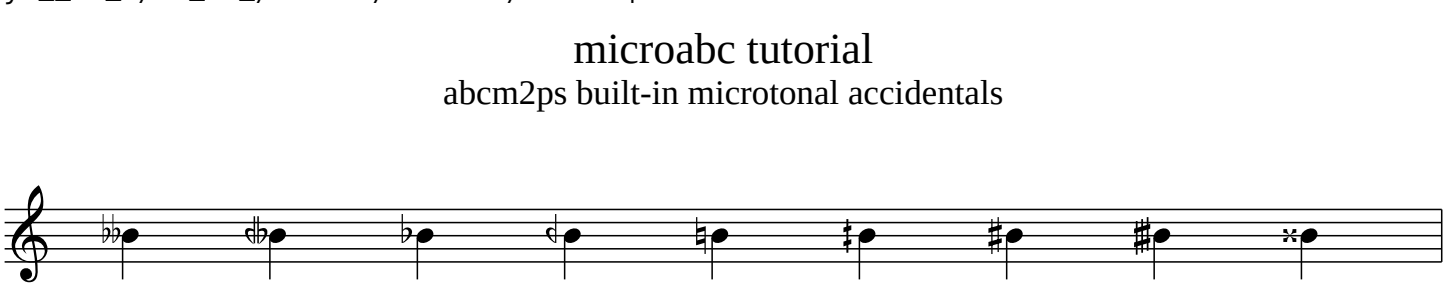
```
abcm2ps temperlin.abc -O=
```



However, there are limitations: **abcm2ps** only includes glyphs for conventional and quarter-tone accidentals (in a Tartini-Couper style; see example below). Any other accidentals are indefinite, hence the user should provide the PostScript definitions for their glyphs. In turn, **abc2midi** cannot deal freely with microtonal accidentals in chords, because normally the notes inside a chord go into a same MIDI channel, and the pitch bend messages (which assign the tuning to the notes) are applied to all simultaneous notes sharing a same channel.[8] Another difference is that **abcm2ps** only accepts fractions which numerator and denominator are lesser than 256.

To handle such limitations and differences between **abcm2ps** and **abc2midi**, and to provide shorter ways to represent pitches like _137/1000A, and also to compute the proper fractions for each note, it seems to be a good idea to use a preprocessor and a tool to generate the macro definitions required for a given tuning.

```
X:1
T:microabc tutorial
T:abcm2ps built-in microtonal accidentals
M:none
L:1/4
%%stretchlast
K:C
y __B _3/B _B _/B =B ^/B ^B ^3/B ^^B |
```



3.2 Macros and preprocessors

microabc is a generator of *macros*, that is, a definition of a character string to be replaced with another character string. A *preprocessor* is a tool which (among other features) can carry out macro replacements. This works much like the "Find/Replace" feature present in text editors.

For instance, let us suppose that we want to use a short macro name to be replaced with the string %%begintext. For the **abcpp** preprocessor, we could define a macro like this:

[8] Since 2006-10-03, **abc2midi** supports microtonal chords provided that a number of MIDI channels be reserved with the command %%MIDI makechordchannels n, where n is the number of additional channels to allocate for the current voice.

```
#define %%BT %%begintext
```

Given the definition above, all occurrences of `%%BT` in the input file will be replaced with `%%begintext`. You can try with macros just now, by using **abcpp**.

First, create a text file named `macroin.txt` with this content:

```
#define SCALE CDEFGABc
#define CHORD  [CEGc]
A C-major scale in written in ABC as:
        SCALE
A C-major tonic chord is written as:
        CHORD
```

Then, run **abcpp** on the file `macroin.txt`, by using this command:

```
abcpp macroin.txt macroout.txt
```

abcpp will read the file `macroin.txt`, and convert it in a new file named `macroout.txt`, replacing all defined macro names with their corresponding replacement texts:

```
A C major scale in written in ABC as:
        CDEFGABc
A C major tonic chord is written as:
        [CEGc]
```

3.3 microabc is a generator of macros

microabc was primarily intended to be a program to generate macro definitions to represent microtonal music in `ABC`. Let us try its basic features with a few short examples.

Create a file `ex1.txt` with this content:

```
range: -5 7
alias: 0
0
1
2
```

Now run **microabc** on that file:

```
microabc < ex1.txt
```

microabc will print on the screen a list of macro definitions based on the file `ex1.txt`, including these lines:

```
#define [1,,] =G,
#define [2,,] ^G,
#define [0,] =A,
#define [1,] ^A,
#define [2,] =B,
#define [0] =C
#define [1] ^C
#define [2] =D
#define [0'] ^D
#define [1'] =E
#define [2'] =F
#define [0''] ^F
#define [1''] =G
```

Please note that, by default, the macro names are generated with square braces, and the replacements are conventional ABC pitches in a chromatic sequence, with =C as the central pitch (our first scale degree, named [0]). Such chromatic sequences (mappings) are useful, for instance, when retuning MIDI files with the program **Scala**.

Note that there are 5 macros before [0], and 7 macros after it. This is the result of the command range: -5 7 of the input file.

Please note yet that the macros use the given numbers 0, 1 and 2, with additional commas (,) or apostrophes (') for lower or upper instances (normally used for octaves).

Finally, the command alias: 0 was used to tell **microabc** that the first column (starting from 0) in the input file contains the strings for the macro names. One can create several macro definitions ("aliases") for each pitch (same replacement), thus the name of the command.[9]

Of course, to use the definitions generated with **microabc**, we will want to store the output in a file. This can be done by either of these commands:

```
microabc < ex1.txt > ex1.abh
microabc -iex1.txt -oex1.abh
```

3.4 microabc as a preprocessor

Although preprocessing was not its initial goal, **microabc** can be used as a specialised preprocessor. In special, **microabc** preprocessor provides considerable support for the Sagittal Notation System.[10] That feature is presented in the section 5.4 (page 19).

3.5 MIDI limitations

One should be aware that MIDI has severe limitations to represent microtonal music.

One way to accomplish microtonal pitches is by using Pitch Bend messages to change the tuning of the note; unfortunately that message affects *all* notes which share a same MIDI channel. Thus, the number of differently tuned pitches played simultaneously is limited by the number of channels in use. Sophisticated algorithms can optimise the use of pitch bend messages, redirecting each note to a suitable MIDI channel. This can be done with **Scala**, when using its *.seq file format, but there are not currently any ABC programs with such a feature. **abc2midi** usually outputs each voice to a different channel, therefore chords in a same voice share a same channel, so there is no easy way to tune the individual notes in a chord.[11] [12]

Another approach is to map every MIDI pitch to a given frequency. For MIDI devices with such capability, one can have available up to 128 different frequencies. There is not the problem with simultaneous notes in a same channel. That can be fine in several cases, specially when the scale has not too many degrees in an octave, for instance: 19-EDO or 24-EDO.[13] But with 31 degrees in an octave (e.g. 31-EDO), the range is reduced to four octaves only (128/31 = 4.129). The frequency mapping usually is device-specific. However there is the Standard MIDI Tuning specification, which set the frequencies by using System Exclusive messages (those messages can be inserted in a MIDI file). Standard MIDI Tuning is not supported by some MIDI devices, thus it is not recommended to use in MIDI files for distribution. An alternative is to use the program **Scala** to retune files which assume a frequency mapping, converting them to files which use Pitch Bend messages instead.

[9] The complete list and description of **microabc** commands is distributed along with the program, in both text (microabc.txt) and HTML (microabc.html) formats. The latter one can also be read online at the address http://hudsonlacerda.webs.com/abc/microabc.html.

[10] Sagittal Notation System website is http://www.sagittal.org.

[11] The program **abc2alias** included in **microabc** package can split ABC chords in a set of temporary voices. However it is still somewhat limited understanding ABC syntax, so that the user may need to change a bit the ABC source to get the proper results. Nonetheless, **abc2alias** can be very useful if used with care. See inside the tools/ folder.

[12] See also the footnote 8 at page 5.

[13] "EDO" stands for "Equal Divisions of Octave".

One should also know that 14bit MIDI Pitch Bend gives a precision of 1/4096 semitone; that corresponds to 0.024414 cents. Standard MIDI Tuning is more precise: 1/16384 semitone or 0.0061035 cents.

Such MIDI limitations force users of ABC programs to choose between those two main approaches to obtain micro-tuned MIDI files. **microabc** provides the *chromatic* and *diatonic* modes for frequency mapping (ABC with conventional accidentals or with no accidentals respectively), and the *microtonal* mode to use pitch bend messages (ABC with microtonal accidentals).

4 The steps

With abcpp — These are the traditional steps to write microtonal music with **microabc**:

1. Define the scale in a **microabc** text file;
2. Generate the file(s) containing the macros definitions, by running **microabc**;
3. Write the music in ABC code with macros;
4. Concatenate or include the macros definitions file(s) with the ABC code and generate pure ABC file(s), by running **abcpp**;
5. Run **abcm2ps** to get the score as a PostScript file and/or run **abc2midi** to get a MIDI file.

Without abcpp — To use **microabc** as preprocessor, the steps are:

1. Define the scale in a **microabc** text file;
2. Write the music in ABC code with macros;
3. Preprocess the music file with **microabc** to obtain ABC file(s) for MIDI and/or PostScript;
4. Run **abcm2ps** to get the score as a PostScript file and/or run **abc2midi** to get a MIDI file.

Sagittal — To use **microabc** as a preprocessor for Sagittal notation, the steps are simply these:

1. Write the music in ABC code with Sagittal pitches (between square braces);
2. Preprocess the music file with **microabc** to obtain ABC file(s) for MIDI and/or PostScript;
3. Run **abcm2ps** to get the PostScript score file and/or run **abc2midi** to get a MIDI file.

5 Examples

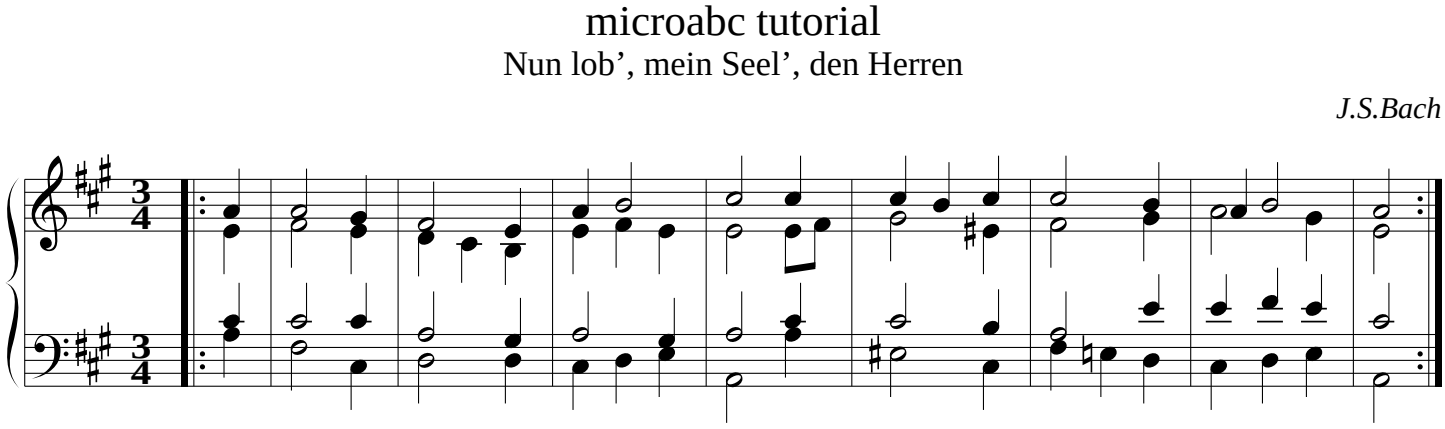
5.1 Quick start

Just start doing. This example requires only these programs:

- **microabc**;
- **abcm2ps**;
- **abc2midi**;
- Any MIDI player;
- Any PostScript viewer.

5.1.1 Bach's choral

Here is the sample music, from a J. S. Bach's choral:



And here is its corresponding code, written as ABC with Sagittal macros:

```
%%format sagittal.fmt
%%format sagittal-mixed.fmt
%%postscript sagmixed
%%continueall
%%maxshrink .8

%%microabc: equaltemp: 19

X:1
T:microabc tutorial
T:Nun lob', mein Seel', den Herren
C:J.S.Bach
L:1/4
M:3/4
K:A exp [F#‘][C#‘][G#‘]   % key signature
%
%%staves {(1 2) (3 4)}
%
V:1 clef=treble % Voice 1
%%MIDI nobeataccents
%%MIDI program 16
%%MIDI trim 1/5
    |: [A] | [A]2 [:G#] | [:F#]2 [E] | [A] [B]2 | [:C#‘]2 \
    [:C#‘] | [:C#‘] [B] [:C#‘] | [:C#‘]2 [B] | [A] [B]2 | [A]2 :|
V:2 clef=treble % Voice 2
%%MIDI nobeataccents
%%MIDI program 16
%%MIDI trim 1/5
    |: [E] | [:F#]2 [E] | [D][:C#][B,] | [E][:F#][E] | [E]2 \
    [E]/[:F#]/ | [:G#]2 [E#] | [:F#]2 [:G#] | [A]2 [:G#] | [E]2 :|
V:3 clef=bass % Voice 3
%%MIDI nobeataccents
%%MIDI program 16
%%MIDI trim 1/5
    |: [:C#] | [:C#]2 [:C#] | [A,]2 [:G#,] | [A,]2 [:G#,] | [A,]2 \
```

```
    [:C#] | [:C#]2 [B,] | [A,]2 [E] | [E] [:F#] [E] | [:C#]2 :|
V:4 clef=bass % Voice 4
%%MIDI nobeataccents
%%MIDI program 16
%%MIDI trim 1/5
    |: [A,] | [:F#,]2 [:C#,] | [D,]2 [D,] | [:C#,] [D,] [E,] | [A,,]2 \
    [A,] | [E#,]2 [:C#,] | [:F#,] [Ee,] [D,] | [:C#,] [D,] [E,] | [A,,]2 :|
```

The code above (saved as `bach.abp`) can be converted to `PostScript` and `MIDI` with the commands:

```
microabc -E -Pbach.abp -obach.abc
microabc -E -Mbach.abp -obach-midi.abc
abcm2ps -O= bach.abc
abc2midi bach-midi.abc
```

Use the `PostScript` viewer and the `MIDI` player to enjoy the results.

Then start to make changes in the code to learn more. . .

5.1.2 Make changes, learn more

The `MIDI` file of the Bach's choral was generated in 19-EDO. Note the line with `%%microabc: equaltemp: 19`. You may make experiments with various equal temperaments, by replacing `19` with other numbers. Or else disable that command to get fifths of 3/2 (Pythagorean just intonation) – to do this, simply remove one character `%` of the start of that line.

You may also get a score in pure Sagittal notation, just changing the `%%postscript sagmixed` to `%%postscript sagpure`.

Take some time trying to understand the code. Focus the attention on the lines of notes. Notice that the accidentals are not shown in the score for pitches starting with a colon, like in `[:C#]`. Also notice that a ♮ is represented with a small letter "`e`", in `[Ee,]`. Experiment to change the code and see the effects of your changes.

As an additional exercise, if you know a bit of Sagittal, use Pythagorean tuning (fifths of 3/2) and then change some accidentals to get pure thirds in the chords.

To learn more about how to use Sagittal with **microabc**, read the section 5.4 (page 19). See also the section 5.5 (page 21), and the **microabc** documentation. To learn more about the Sagittal Notation System itself, visit `http://www.sagittal.org`.

The following sections illustrate in some detail the main features of **microabc**.

5.2 A scale of harmonics – microabc output modes

microabc has four output modes: *microtonal*, *chromatic*, *diatonic* and *literal*. Chromatic and diatonic modes are useful to generate `MIDI` files to be re-tuned with the program **Scala** or played by **timidity++**. They can also be used to generate a mapping for standard staff notation (tablature-wise approach). Microtonal mode computes pitch quantisation (approximation) in semitone fractions relative to the standard equal temperament (quarter-tones, eighth-tones, integer cents, etc.), using microtonal `ABC` accidentals for **abc2midi** or **abcm2ps**. Literal mode uses given replacement text strings.

As an example to illustrate the **microabc** modes, we will take the first odd elements of the harmonic series up to 23rd harmonic, reduced to the range of an octave. That scale can be obtained by executing the following commands in the program **Scala**:

```
harm 1 23
norm
show
```

Here is the resulting scale:

```
  0:          1/1              0.000 unison, perfect prime
  1:         17/16           104.955 17th harmonic
  2:          9/8            203.910 major whole tone
  3:         19/16           297.513 19th harmonic
  4:          5/4            386.314 major third
  5:         21/16           470.781 narrow fourth
  6:         11/8            551.318 undecimal semi-augmented fourth
  7:         23/16           628.274 23rd harmonic
  8:          3/2            701.955 perfect fifth
  9:         13/8            840.528 tridecimal neutral sixth
 10:          7/4            968.826 harmonic seventh
 11:         15/8           1088.269 classic major seventh
 12:          2/1           1200.000 octave
```

5.2.1 The microabc input file

We will make the input file for **microabc** by adding these commands above the scale data generated by **Scala**:

```
alias:0     {column 0 gives the aliases}
scl:1       {column 1 gives the intervals of the scale for microtonal mode}
range:-40 40 {pitch range (ambitus)}
```

Then, we edit the output of **Scala** accordingly: we will use the numbers 0–11 as aliases (without the colons of the **Scala** output), and we will remove the last line (octave, 2/1, which is the pitch class 0 again, octave up).[14] Let us save this as `harmonic.txt`:

```
{----------begin----------}
alias:0     {column 0 gives the aliases}
scl:1       {column 1 gives the intervals of the scale for microtonal mode}
range:-40 40 {pitch range (ambitus)}
  0          1/1              0.000 unison, perfect prime
  1         17/16           104.955 17th harmonic
  2          9/8            203.910 major whole tone
  3         19/16           297.513 19th harmonic
  4          5/4            386.314 major third
  5         21/16           470.781 narrow fourth
  6         11/8            551.318 undecimal semi-augmented fourth
  7         23/16           628.274 23rd harmonic
  8          3/2            701.955 perfect fifth
  9         13/8            840.528 tridecimal neutral sixth
 10          7/4            968.826 harmonic seventh
 11         15/8           1088.269 classic major seventh
{-----------end-----------}
```

5.2.2 The ABC file with macros

To try with our example, let us write a music file. Remember, the pitches are defined as the numbers 0–11 delimited by square braces (provided by **microabc**); octaves up or down are represented like in ABC, with apostrophes or commas.[15] Let us save this as `harmonic.abp`.

[14] Unlike **Scala** files, the input format for **microabc** starts with pitch 1/1 and ends with the last pitch *before* the octave (or other equivalence interval). To use a different equivalence interval, insert a `sclmod:<interval>` line.

[15] The delimiters can be changed with the command `delim:`, and the octave modifiers with the command `updown:`.

```
X:1
T:microabc tutorial
T:harmonics
M:6/8
L:1/8
Q:1/8=90
V:1
%%MIDI program 16
V:2
%%MIDI program 16
K:none
%%continueall
%%maxshrink 0.9
V:1
[0'][1'][2'][3'][4'][5'] | [6'][7'][8'][9'][10'][11'] |
[0][1][2][3][4][5] | [6][7][8][9][10][11] |
[0,][1,][2,][3,][4,][5,] | [6,][7,][8,][9,][10,][11,] |]
V:2
[0,,]6- | [0,,]6 |
[0,,]6- | [0,,]6 |
[0,,]6- | [0,,]6 |]
%
```

5.2.3 *Microtonal* mode

The command `scl:1` included in the input file (section 5.2.1, page 11) is sufficient to activate the microtonal mode. To generate the macro definitions for `MIDI` files, execute:

```
microabc < harmonic.txt > harm4mid.abh
```

You may be interested in to see inside the generated file `harm4mid.abh`, to know the definitions. It will contain lines like these, where you can see microtonal `ABC` pitches at the resolution of 1/4096 semitone:

```
#define [9,] _2436/4096A,
#define [10,] ^2819/4096A,
#define [11,] _481/4096B,
#define [0] =C
#define [1] _3893/4096D
#define [2] ^160/4096D
#define [3] ^3994/4096D
#define [4] _561/4096E
#define [5] _1197/4096F
#define [6] ^2102/4096F
#define [7] _2938/4096G
#define [8] ^80/4096G
#define [9] _2436/4096A
#define [10] ^2819/4096A
#define [11] _481/4096B
#define [0'] =c
#define [1'] _3893/4096d
```

Let us create the `ABC` code and the `MIDI` file:[16]

[16] DOS/Windows users should use `type` instead of `cat` in the command line.

```
cat harm4mid.abh harmonic.abp | abcpp > harm4mid.abc
abc2midi harm4mid.abc
```

To get a score in microtonal mode, we need to quantise those accidentals from 1/4096 semitone to a smaller denominator. **abcm2ps** supports quarter-tones (1/2 semitone), but we may choose a finer resolution: 1/4 semitone, since **microabc** can provide the required glyphs for the accidentals.

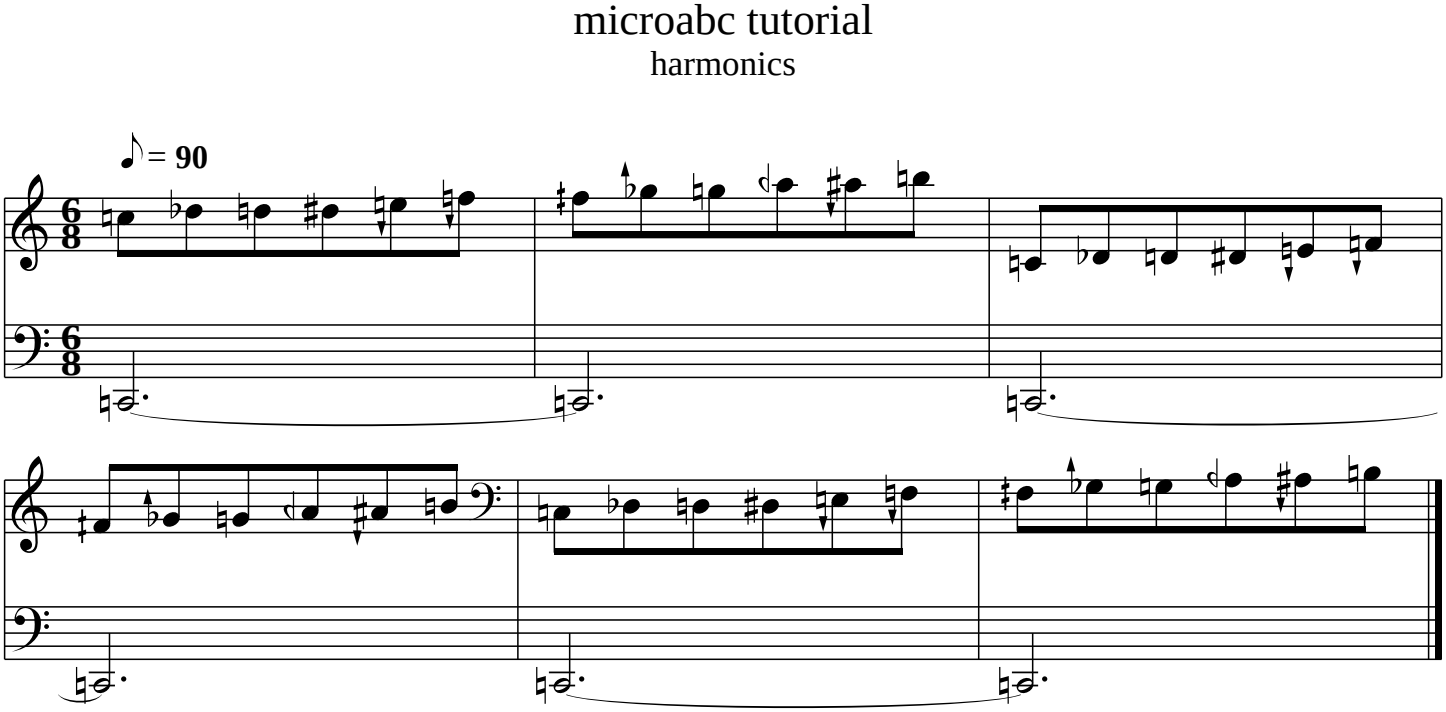
The command below will create a file with the macros for our scale quantised to eighth-tones (`harm4ps.abh`). Please note the command-line options `den:4`, which sets the denominator to 1/4 semitone, `abcm2ps:4` which sets the same quantisation for **abcm2ps**, and `psacc:1` which tells **microabc** to create the glyphs for the microtonal accidentals:

```
microabc den:4 abcm2ps:4 psacc:1 < harmonic.txt > harm4ps.abh
```

These commands will create the `ABC` code and the `PostScript` score file from it:

```
cat harm4ps.abh harmonic.abp | abcpp > harm4ps.abc
abcm2ps -O= harm4ps.abc
```

The score should look like this:



5.2.4 *Chromatic* mode, timidity++ and Scala

It was said before (section 3.5, page 7) that the microtonal mode is not very suitable to obtain `MIDI` files with **abc2midi** when chords are used. In such circumstances, a better solution may be using the chromatic mode and retune the `MIDI` file with **Scala**, or playing it with **timidity++**, or yet creating it with Standard `MIDI` Tuning messages.[17]

The following commands will create the macro definitions in chromatic mode, and then generate the `ABC` and `MIDI` files:

```
microabc chromatic:1 < harmonic.txt > harmchrm.abh
cat harmchrm.abh harmonic.abp | abcpp > harmchrm.abc
abc2midi harmchrm.abc
```

[17] Another solution can be using the program **abc2alias** (included in **microabc** package) to expand the chords to temporary voices (voice overlay).

The option `chromatic:1` in the command line will cancel the effect of the `scl:1` command from the file `harmonic.txt`, that is, the microtonal mode will be disabled and the chromatic mode will be enabled. As a consequence, the tuning information will be ignored – but we need using that information to retune or play the `MIDI` file.

To play the file with **timidity++**, we need firstly export the scale in a format suitable for that program. A frequency table will be output to the file `harmonic.tbl` with the command:[18] [19]

```
microabc timidity:harmonic.tbl < harmonic.txt > /dev/null
```

Now we can hear the `MIDI` file with **timidity++**, using the option `-Z` to load the frequencies file:[20]

```
timidity harmchrm1.mid -Z harmonic.tbl
```

We can also use **timidity++** to generate a `.wav` file by adding the option `-Ow`:[21]

```
timidity harmchrm1.mid -Z harmonic.tbl -Ow
```

Our alternative approach is retuning the `MIDI` file. We can export the tuning from the **microabc** input, in the **Scala** file format:

```
microabc scala:harmonic.scl < harmonic.txt > /dev/null
```

This command calls **Scala** to retune the `MIDI` file – supposed to be named `harmchrm1.mid` – as a new file called `harm-scl.mid`:

```
scala harmonic.scl --example/midi harmchrm1.mid harm-scl.mid --exit
```

Still using the `ABC` code generated with the chromatic mode, we could create a `MIDI` file with Standard `MIDI` Tuning messages. This option is illustrated in the section 5.3 (page 16).

5.2.5 *Diatonic* mode

The diatonic mode is not necessary in the present example of scale of harmonics. Nonetheless, it will be used for the purpose of illustration.

The following commands will generate a score using a diatonic mapping of the pitches of our scale (the **microabc** option `diatonic:1` is to set the diatonic mode):

```
microabc diatonic:1 < harmonic.txt > harmdiat.abh
cat harmdiat.abh harmonic.abp | abcpp > harmdiat.abc
abcm2ps -O= harmdiat.abc
```

The resultant score (below) uses only the natural pitches, without any accidentals. The diatonic mode can be used to create non-conventional staves. Note that, in this specific case in which there are 12 notes per octave, every octave instance spans on 6 staff lines.[22] Such property could be used to simulate a sort of "keyboard-view" like those present in some `MIDI` sequencer programs.

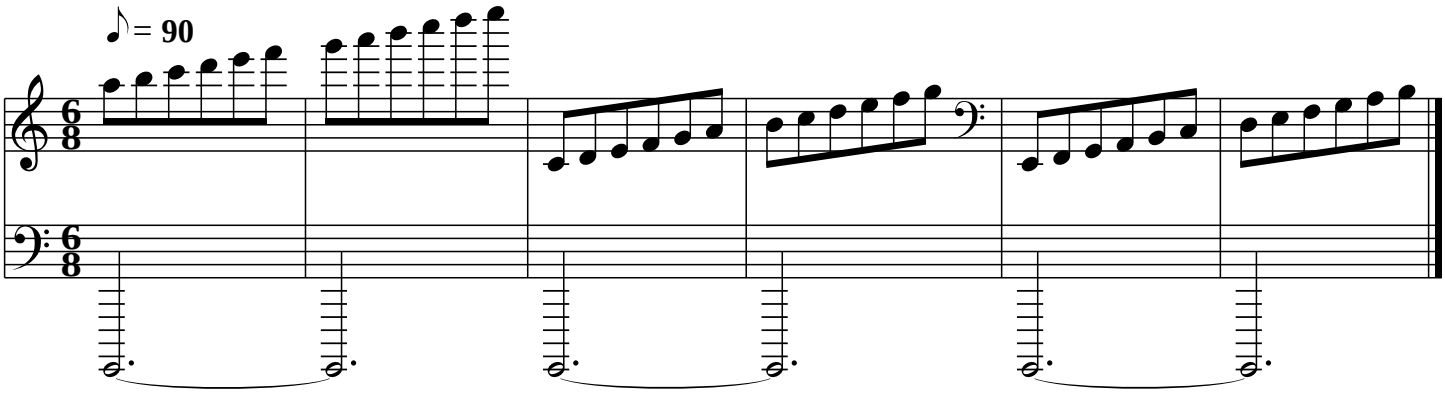
[18] DOS/Windows users should exclude the part "`> /dev/null`" from the commands; some data will be printed on the screen.

[19] Frequency tables for **timidity++** may be also created using **Scala**. For instance, by issuing this command:
`scala harmonic.scl --set synth 117 "--send/file %scl(.tbl)" --exit`

[20] Playing the file with **timidity++**, using a frequency table, will result in a finer tuning, compared with other approaches.

[21] To obtain a compressed audio file in `OGG Vorbis` format, run **timidity++** with option `-Ov`.

[22] An additional observation is that – as a coincidence – the pitch class [0] lies on the same helper line in both treble and bass clefs (see the first note in the upper voice, at measures 3 and 5).



5.2.6 *Literal* mode

The remaining **microabc** mode allows setting arbitrary replacement strings for the aliases. We will not show, at this point, any useful examples – we will rather simply inspect which macro definitions are generated when we use the second column of the input file (index 1: that column with the intervals of the scale) as replacement text.[23]

```
microabc replace:1 < harmonic.txt | less
```

The result printed on the screen should contain lines such as these ones:

```
#define [9,] 13/8
#define [10,] 7/4
#define [11,] 15/8
#define [0] 1/1
#define [1] 17/16
#define [2] 9/8
#define [3] 19/16
#define [4] 5/4
#define [5] 21/16
#define [6] 11/8
#define [7] 23/16
#define [8] 3/2
#define [9] 13/8
#define [10] 7/4
#define [11] 15/8
#define [0'] 1/1
#define [1'] 17/16
```

The **microabc** literal mode has a few variants, used when the replacement text is in a special format: `replaceabc:` and `replacesagittal:`, respectively for `ABC` and Sagittal formats.[24]

Also, there is the option `replaceupdn:` to insert "octave" modifiers (like `'` and `,` in `ABC`) in the replacements. In the example above, the replacement text does not contain such modifiers: any instance of pitch class [0] is replaced with 1/1, any instance of pitch class [11] is replaced with 15/8 and so on.

[23] DOS/Windows users should use `more` instead of `less` in the command line.

[24] Sagittal format does not work with **abcpp**; it will work only when using **microabc** as preprocessor. See section 5.4 at page 19.

5.3 An example in 19-EDO

Let us suppose we want to write a music in 19 equal divisions of octave. The score can be accomplished using the conventional accidentals flat, natural and sharp, but we need tune the pitches for `MIDI` files. Of course, we want to write only one `ABC` input code to obtain both `PostScript` and `MIDI` outputs.

5.3.1 The microabc file

A solution is to use macros for each pitch instead of simple `ABC` pitches. The replacement for each macro should be different, depending on output format: one for `PostScript` generation, another for `MIDI` generation. By default, the **microabc** macros are delimited by square braces `[` and `]`.

5.3.2 `PostScript`

For `PostScript`, we want "common" pitches and accidentals. Therefore: `C` can be represented as the macro `[C]`; `^C` as `[^C]`; `_D` as `[_D]`; and so on, that is: all macro names will be the corresponding `ABC` pitch names between square braces.

5.3.3 `MIDI`

For `MIDI`, the things are not so simple, because normally we have 12 chromatic pitches per octave, but our scale has 19. The frequencies for all pitches need to be set, to get a tuned `MIDI` file. Let us use **abc2midi**'s `%%MIDI snt` command for Standard `MIDI` Tuning.[25]

5.3.4 The code

Below is the basic **microabc** code for our scale, `tut.txt`. It sets a scale with 19 steps of size, which is an equal temperament defined by 19 divisions of a octave (2/1). The pitch names are like `ABC` pitch names, listed at columns 0 and 1. Enharmonic pitches share a same line in the following list, and here they are considered as synonyms ("aliases").

```
{----------begin----------}
size:19
equaltemp: 19 2/1
aliasabc: 0 1
range:-20 20
{}
=C C
^C __D
_D ^^C
=D D
^D __E
_E ^^D
=E E
^E _F
=F F
^F __G
_G ^^F
=G G
^G __A
```

[25] Standard MIDI Tuning is not implemented in several MIDI devices, but it is supported by the program **timidity++**. If you are creating MIDI files for distribution, you may prefer retune them with **Scala** using Pitch Bend messages, which are more portable. The **microabc** command `scala:` allows you to export a `*.scl` file. To retune a MIDI file with **Scala**, issue a command like this:
`scala scalefile.scl --example/midi inputfile.mid outputfile.mid --exit`

```
_A ^^G
=A A
^A __B
_B ^^A
=B B
^B _c
{}
{-----------end-----------}
```

5.3.5 Macro definitions files

In **microabc**, macros can have several aliases (synonyms), but only one meaning, that is, only one replacement. To maintain a strict correspondence between macros and pitches for a `PostScript` score (enharmonic pitches should be differentiated in a score), **microabc** needs to be ran twice, and the outputs concatenated. The first run sets the macro definitions for the first (indexed by 0) column of the input file; the second run sets the macros for the enharmonic pitches (second column of input). In this example, the replacement is one of the list columns, taken as an `ABC` pitch.

The command `cat` concatenates the files into `tut-ps.abh` (`MS-DOS` users should use `type` instead).

```
microabc -itut.txt aliasabc:0 replaceabc:0 -otut-ps.ab1
microabc -itut.txt aliasabc:1 replaceabc:1 -otut-ps.ab2
cat tut-ps.ab1 tut-ps.ab2 > tut-ps.abh
```

The macros for `MIDI` are easier to obtain, because both input columns 0 and 1 mean a same pitch (enharmonic differentiation is irrelevant here). Our **microabc** input file already has selected both columns with `aliasabc: 0 1` (above overridden for `PostScript` score). The additional command option `snt:` generates the file `tut.snt` for Standard `MIDI` Tuning.

```
microabc -itut.txt -otut-mid.abh snt:tut.snt
```

5.3.6 The ABC file with macros

The next step is to write the music in `ABC` notation with macros, stored in the file `tut.abp`. Remember that all pitches must be between square braces (they are macro names, and **microabc** macro names are delimited by square braces), and note the `#include` command to insert the Standard `MIDI` Tuning definitions from the file `tut.snt`.[26] Here is our scale:

```
X:1
T:microabc tutorial
T:19-edo
#include "tut.snt"
M:6/8
L:1/8
K:C
[C][^C][_D] [=D][^D][_E] |\
[=E]2 z [E][^E][F] |\
[^F][_G][=G] [^G][_A][=A] |\
[^A][_B][=B] [^B]3 | [c]3 z3 |]
```

[26] If you want to use **Scala** to retune the MIDI file, create a `*.scl` file with:
`microabc -itut.txt scala:tut.scl`
Then, remove the line
`#include "tut.snt"`
of the sample source given in this section and follow the instructions in the footnote number 25 (page 16) to retune the MIDI file according to the scale file `tut.scl`.

5.3.7 Preprocessing

The ABC code with macros (tut.abp) needs to be preprocessed and converted to pure ABC code.

ABC file for PostScript score generation:

```
cat tut-ps.abh tut.abp | abcpp > tut-ps.abc
```

ABC file for MIDI file generation:

```
cat tut-mid.abh tut.abp | abcpp > tut-mid.abc
```

5.3.8 Getting the MIDI file

The MIDI simulation is generated by **abc2midi**, with the following command:

```
abc2midi tut-mid.abc -o tut.mid
```

The synthesizer **timidity++** can play the MIDI file with Standard MIDI Tuning messages and convert it to other formats like .wav or .ogg.[27]

5.3.9 Getting the PostScript score file

Use **abcm2ps** to convert the ABC code into a graphical PostScript score:

```
abcm2ps tut-ps.abc -O tut.ps
```

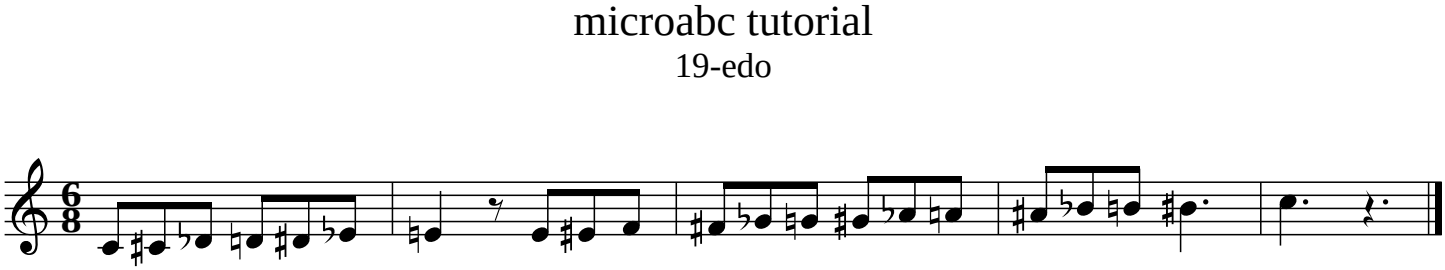
The output can be previewed with **ghostview**, and then printed or converted to PDF or other formats.

5.3.10 Customisation of the accidentals in the PostScript score file

Just to show another interesting feature of **abcm2ps** – the insertion of PostScript instructions to (re)define symbols, let us customise the score glyphs for the accidentals sharp and flat. Put this code in the original ABC file (tut.abp), before the music, then follow again the steps of the sections 5.3.7 and 5.3.9:

```
%%postscript /sh0{ gsave exch .45 add exch T .85 SLW
%%postscript    -1.2 -8.4 M 0 15.4 RL
%%postscript    1.4 -7.2 M 0 15.4 RL stroke
%%postscript    -1 0 translate .9 1 scale
%%postscript    -2.6 -3 M 5.4 1.6 RL 0 -2.2 RL -5.4 -1.6 RL 0 2.2 RL fill
%%postscript    -2.6 3.4 M 5.4 1.6 RL 0 -2.2 RL -5.4 -1.6 RL 0 2.2 RL fill
%%postscript    grestore}!
%%postscript /ft0{ gsave T .95 SLW
%%postscript    -1.8 2.5 M
%%postscript    6.4 3.3 6.5 -3.6 0 -6.6 RC
%%postscript    4.6 3.9 4.5 7.6 0 5.7 RC
%%postscript    currentpoint fill M
%%postscript    0 7.1 RM 0 -7.1 RL stroke
%%postscript grestore}!
```

Here is the resulting score:



[27] See page 14.

5.4 Sagittal notation – using microabc as preprocessor

microabc can do a specialised form of preprocessing. Unlike **abcpp**, macro definitions are not declared with `#define` statements. The definitions are based on the **microabc** input file itself (option `-i`), which defines the working mode, the scale, the aliases and other options – or the definitions are built-in, in the case of using the Sagittal microtonal notation. To omit the input file, use the option `-i-`.

The command line options `-p`, `-M` and `-P` preprocess the file given as argument. The option `-S` enables the support for Sagittal notation.

5.4.1 Just Intonation

You find below the sample music file `sagit.abp`. It uses the format file `sagittal-pfb.fmt`, which is included in **microabc** package.[28] [29] Note that the octaves are indicated with `,` (down) and `‘` (up) – the latest is a grave accent, not the apostrophe (`’`) used in `ABC`.

```
%%format sagittal-pfb.fmt
%%stretchlast

X:1
T:microabc tutorial
T:Sagittal preprocessing
K:C
%%text Pure intonation (fifth=2:3, octave=1:2)
V:1
[C][E\] [G][Bbt] [D‘][F^‘] [Aw‘][B\‘] |\
[C‘‘][B\‘] [Aw‘][F^‘] [D‘][Bbt] [G][E\] | [C]4 z4 |]
V:2
[C,]8 | [C,,]8 | [C,]4 z4 |]
```

To get a `MIDI` file, preprocess with the option `-M`:[30]

```
microabc -i- -S -Msagit.abp > sagit-mid.abc
abc2midi sagit-mid.abc
```

And to get a `PostScript` score, use the option `-P`:[31]

```
microabc -i- -S -Psagit.abp > sagit.abc
abcm2ps -O= sagit.abc
```

The score will look like this:

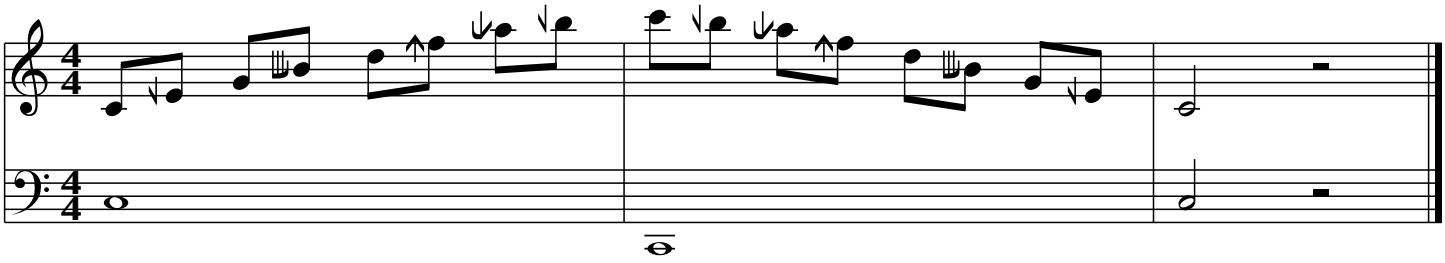
[28] **abcm2ps** expects that the format file be stored in the current directory or in the “default format directory” which you can know by issuing `abcm2ps -V`. The directory for format files can also be indicated with the **abcm2ps** option `-D`. For details, please consult the **abcm2ps** documentation.

[29] The use of `sagittal-pfb.fmt` assumes that the Sagittal `PostScript` Type 1 font is installed in your system. You can alternatively try with `sagittal.fmt` which embeds the font in the output `PostScript` file, or yet, use the flag `-e` when running **microabc**.

[30] The `MIDI` file will be generated in microtonal mode, that is, using `MIDI` Pitch Bend messages. It has the limitation of do not support chords. Splitting the chords into multiple voices can be made with **abc2alias**. You may alternatively use a special command for **abc2midi** (see note 8 at page 5).

[31] If you do not want to use **abcm2ps** format files, call **microabc** with the flag `-e`:
`microabc -i- -S -e -Psagit.abp > sagit.abc`

Pure intonation (fifth=2:3, octave=1:2)



5.4.2 Equal divisions of octave

The conversion of Sagittal pitches for MIDI is, by default, done in just intonation, based on pythagorean nominals – with fifths of 2:3 and octaves of 1:2.

Equal divisions of octave can be set with the **microabc** command `equaltemp:`, which can be given in an input file, from the command line or even from the music source file (`.abp`), in this format:

```
%%microabc: equaltemp:<ndivisions>
```

Here is an example:

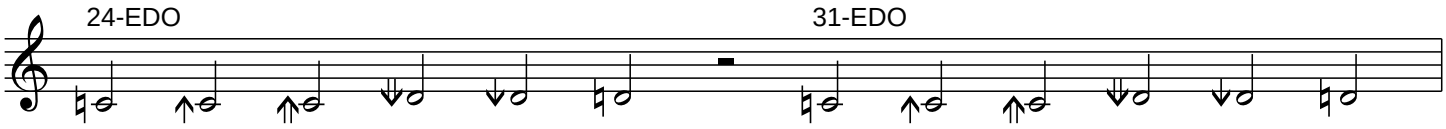
```
%%format sagittal-pfb.fmt
%%stretchlast
%%continueall

X:1
T:microabc tutorial
T:equal divisions of octave
M:none
L:1/2
%%MIDI program 16
%%MIDI trim 1/4
K:C
%%microabc: equaltemp:24
"^24-EDO"\
[Ce][C^][C/N\][D\U/][Dv][De] z
%%microabc: equaltemp:31
"^31-EDO"\
[Ce][C^][C/N\][D\U/][Dv][De] |
```

Just like in the previous example, these commands will convert the file `edo.abp` into `edo-midi.abc` (for **abc2midi**) and `edo.abc` (for **abcm2ps**):

```
microabc -i- -S -Medo.abp > edo-midi.abc
microabc -i- -S -Pedo.abp > edo.abc
```

The score and the MIDI file are obtained with:

```
abcm2ps -O= edo.abc
abc2midi edo-midi.abc -o edo.mid
```



This example (`edo.abp`) uses the "same" pitches in two different tunings: 24-EDO and 31-EDO. The difference is noticeable in the `MIDI` file: `C/N\` and `D\U/` are equivalent in 24-EDO, but not in 31-EDO.

5.5 Using Sagittal notation for arbitrary scales

This is an example which may be used as reference for many applications. It uses **Scala** to aid with Sagittal notation. The `MIDI` file is created in *chromatic* mode, and then retuned with **Scala**. As the scale size is small, there is no a "range *versus* chord" conflict.[32]

5.5.1 Pick out a scale — make the microabc input file

Start the program **Scala** and create or load a scale. For instance, click the button `Open`, select the file `barlow_17.scl` and then click on `OK`.

Now, we need to name the pitches. Let us use a Sagittal notation for just intonation, with short mixed names. Execute these commands in the **Scala** prompt (the bottom line of the interface):

```
set notation saji1
set sagittal short mixed
show
```

The result is shown below. Save these lines in a text file.[33]

```
 0:          1/1          C     B#\ unison, perfect prime
 1:         25/24         Db\   C#_ classic chromatic semitone, minor chroma
 2:         27/25         C#/   Db= large limma, BP small semitone
 3:          9/8          D        major whole tone
 4:         32/27         Eb    D#\ Pythagorean minor third
 5:         11/9          Eo    D#f undecimal neutral third
 6:         32/25         E/    Fb= classic diminished fourth
 7:          4/3          F     E#\ perfect fourth
 8:         25/18         Gb\   F#_ classic augmented fourth
 9:         36/25         F#/   Gb= classic diminished fifth
10:          3/2          G        perfect fifth
11:         25/16         Ab\   G#_ classic augmented fifth
12:         18/11         Av    G#q undecimal neutral sixth
13:         27/16         A        Pythagorean major sixth
14:         16/9          Bb    A#\ Pythagorean minor seventh
15:         50/27         B_    Cb\ grave major seventh
16:         48/25         B/    Cb= classic diminished octave
17:          2/1          C     B#\ octave
```

[32] See section 3.5, page 7.

[33] Tip: To copy the text, do click with the right button of the mouse on the text window of **Scala** and select the option `Edit with @edit...` Another option is to use the Scala command `@microabc`, which helps writting files for **microabc**.

We need editing the file to adapt it for **microabc**.

First, delete the last line. It is just the pitch 0 an octave higher.[34]

Note the presence of equivalent pitches, like `C` and `B#\`. There are a few pitches with no equivalents: `D`, `G` and `A`. For such pitches, fill the fourth column with a dot (`.`).[35]

Note yet that the pairs of equivalents {`C B#\`}, {`B_ Cb\`} and {`B/ Cb=`} do not specify the different octaves. Add a comma (`,`) to `B#\` to make clear that the `B#\,` is of the lower octave. Then add a grave accent (`‘`) to `Cb/` and `Cb=` to indicate the higher octave: `Cb/‘ Cb=‘`.

The next step is to tell **microabc** how to interpret the data. Insert these instructions above the scale data:

```
scl:1
aliassagittal:2 3
chromatic:1
```

`scl:1` means that the column 1 (start counting by zero) contains the tuning.

`aliassagittal:2 3` tells **microabc** that the columns 2 and 3 contains the pitch names in Sagittal.

`chromatic:1` selects the *chromatic* mode. Its effect is that, when generating the `MIDI` file, the pitches will be mapped to the chromatic scale (rather than be tuned with pitchbends). The `MIDI` file will be retuned later with **Scala**.

Save the file as `blw17.txt`. Here are its contents:

```
scl:1
aliassagittal:2 3
chromatic:1
  0:          1/1          C     B#\, unison, perfect prime
  1:         25/24         Db\   C#_ classic chromatic semitone, minor chroma
  2:         27/25         C#/   Db= large limma, BP small semitone
  3:          9/8          D     .    major whole tone
  4:         32/27         Eb    D#\ Pythagorean minor third
  5:         11/9          Eo    D#f undecimal neutral third
  6:         32/25         E/    Fb= classic diminished fourth
  7:          4/3          F     E#\ perfect fourth
  8:         25/18         Gb\   F#_ classic augmented fourth
  9:         36/25         F#/   Gb= classic diminished fifth
 10:          3/2          G     .    perfect fifth
 11:         25/16         Ab\   G#_ classic augmented fifth
 12:         18/11         Av    G#q undecimal neutral sixth
 13:         27/16         A     .    Pythagorean major sixth
 14:         16/9          Bb    A#\ Pythagorean minor seventh
 15:         50/27         B_    Cb\‘ grave major seventh
 16:         48/25         B/    Cb=‘ classic diminished octave
```

5.5.2 The microabc test file

We want to see and to listen to the “Barlow’s 11-limit rational 17-equal, Barlow, On the Quantification of Harmony and Metre” scale, to test and check our **microabc** input file.

Let **microabc** create a test file `blw.abp`:

```
microabc -tblw.abp < blw17.txt
```

[34]The equivalence interval (formal octave) is defined, in **microabc** with the instruction `sclmod:<interval>`.

[35]For certain scales, **Scala** shows some intervals in cents, adding the word “cents”. A similar procedure may be then necessary in order to assure the correspondence of columns for **microabc**.

Now, preprocess the test file. One command generates ABC code for **abc2midi**. The other command generates ABC code for **abcm2ps** (without using blw17.txt).

```
microabc -Mblw.abp < blw17.txt > blw-midi.abc
microabc -i- -S -Pblw.abp > blw.abc
```

Now, let us convert the ABC files to MIDI and PostScript. Put a copy of the file sagittal.fmt[36] in the same directory where the ABC files are, and execute:

```
abc2midi blw-midi.abc
abcm2ps blw.abc -O=
```

Recall that the MIDI file was not generated in the Barlow's scale, but in chromatic mapping. To tune it, load the scale file barlow_17.scl into **Scala** and access the menu Tools | Retune MIDI-file..., then select the input file blw-midi1.mid and choose the output file name blw.mid. Finally, click the OK button. Running **Scala** with this command line does the same:

```
scala barlow_17.scl --example/midi blw-midi1.mid blw.mid --exit
```

Check the resulting PostScript and MIDI files. You may find the MIDI file just boring, but the PostScript file contains information of interest, for example: the score covers the complete range (*ambitus*) available.

Now, let us generate a test file just covering just one octave, and without the equivalent pitches. Modify the file blw17.txt according to this:

1. Change "aliassagittal:2 3" to "aliassagittal:2"; that will omit the equivalent pitches.
2. Add a line with "range:0 17"; that will limit the range to the central octave.

The first four lines of blw17.txt will now be:

```
scl:1
aliassagittal:2
range:0 17
chromatic:1
```

Redo the process of generation of the test file and its conversion to PostScript and MIDI files:[37]

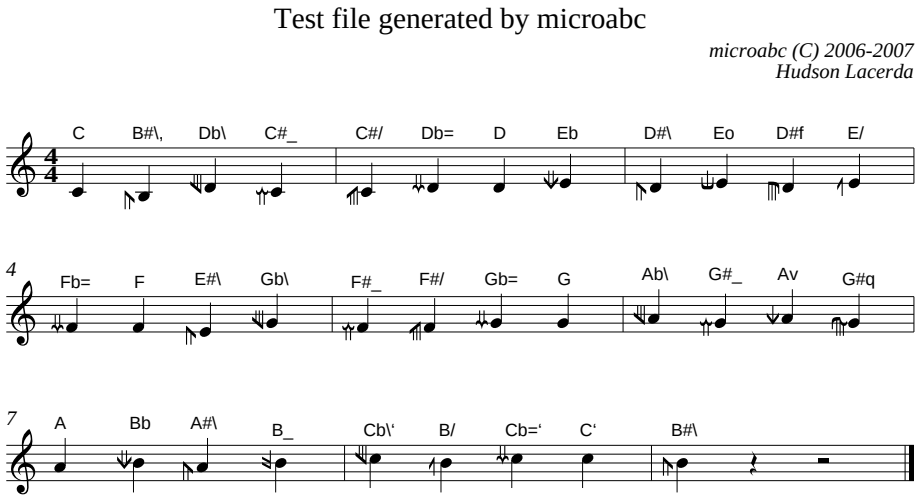
```
microabc -tblw.abp < blw17.txt
microabc -Mblw.abp < blw17.txt > blw-midi.abc
microabc -i- -S -Pblw.abp > blw.abc
abc2midi blw-midi.abc
abcm2ps blw.abc -O=
scala barlow_17.scl --example/midi blw-midi1.mid blw.mid --exit
```

The example below uses "aliassagittal:2 3" to show the equivalents:

[36] sagittal.fmt is included in the **microabc** package. It is used by **abcm2ps** to provide the Sagittal font. If you prefer do not deal with format files, add the flag -e to the **microabc** command line in the previous step:
`microabc -i- -S -e -Pblw.abp > blw.abc`

[37] You may want issue the third command as:
`microabc -i- -S -e -Pblw.abp > blw.abc`

COMMAND LINE: microabc -tblw.abp
MICROABC INPUT FILE:
MICROABC MODE: chromatic
SCALA *.scl INPUT FILE:
SCALA *.scl OUTPUT FILE:
SCALA *.kbm OUTPUT FILE:
TIMIDITY OUTPUT FILE:
SCALE SIZE: 17
RANGE: 0 17 (18)
NUMBER OF DEFINITIONS: 33



5.5.3 Create your own *.abp music file

If the testings with the test file succedded, we are ready to write some music.[38]

Scala will help us to write a scale. Issue `show/notation` in the command prompt (or press `F7`), then copy the pitches:

```
C Db\ C#/ D Eb Eo E/ F Gb\ F#/ G Ab\ Av A Bb B_ B/ C.1
```

Note that **Scala** uses numbers to indicate the octaves. Replace the last pitch `C.1` with `C‘` as we are using characters ‘ and , as octave modifiers.[39] Then enclose all pitches between brackets `[` and `]`, and add an `ABC` header. The example below includes also a few chords:

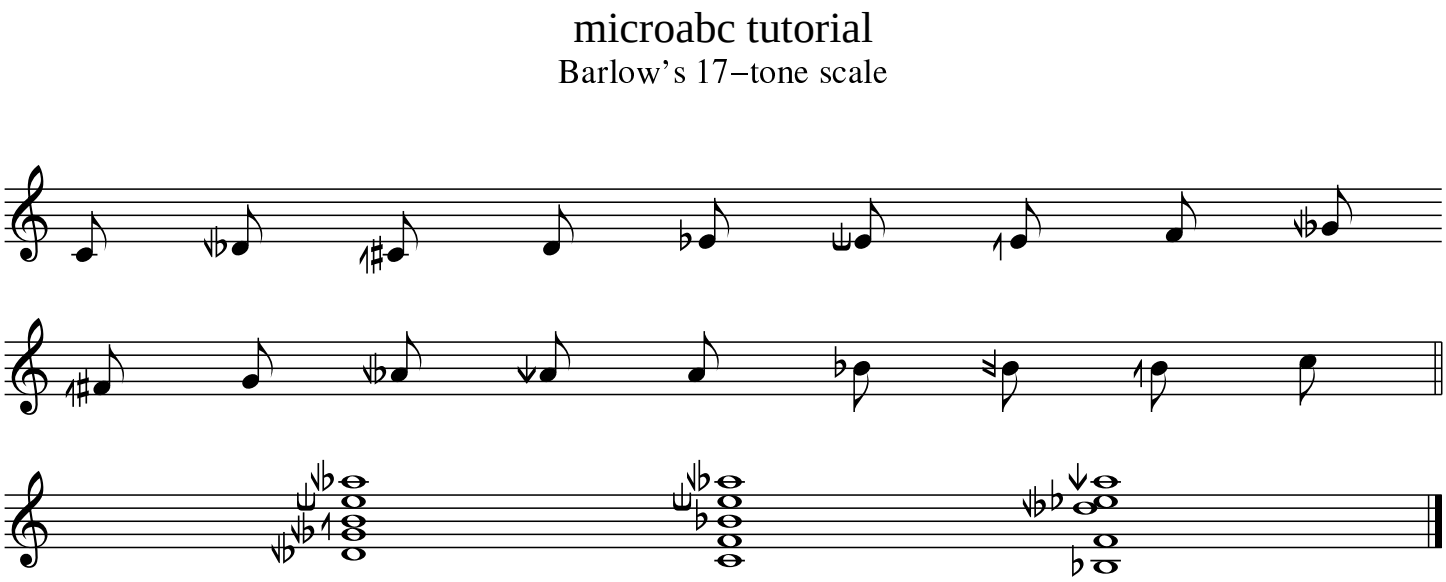
```
%%format sagittal-pfb.fmt
%%format sagittal-mixed.fmt
%%postscript sagmixed

X:1
T:microabc tutorial
T:Barlow's 17-tone scale
M:none
K:C
[C] [Db\] [C#/] [D] [Eb] [Eo] [E/] [F] [Gb\]
[F#/] [G] [Ab\] [Av] [A] [Bb] [B_] [B/] [C‘] ||
```

[38] Be sure to remove the "`range:`" limitation of the file `blw17.txt`!

[39] Although in this section we are using Sagittal pitch names, we are not preprocessing for `MIDI` with the flag `-S`. Hence, the Sagittal pitches from the `*.abp` files are not interpreted as such, but as ordinary macros: variants like `[Ce]` for `[C]`, or even `[:Ce,,]` for `[C,,]` will not work without the flag `-S`. However, do not use `-S` to generate `MIDI` with arbitrary scales: **microabc** would ignore the user-defined scale and interpret the Sagittal pitches in the default way. If you want to add variants, add more `sagittalalias:` columns to the **microabc** input file. Be rigorous to use the pitch names exactly as defined.

```
x4 \
[[Db\][Gb\][B/][Eo‘][Ab\‘]]8 \
[[C][F][Bb][Eo‘][Ab\‘]]8 \
[[Bb,][F][Db\‘][Eb‘][Av‘]]8 |]
```

Here is the resulting score:



5.6 Approximating Sagittal pitches to arbitrary scales

Besides the direct assignment of Sagittal pitches, it is possible to approximate a Sagittal pitch to the nearest pitch in an arbitrary scale. In the next example, we will use Sagittal notation for the George Secor’s 17-tone well temperament, which is an alternative to 17-EDO.

```
! secor17wt.scl
!
George Secor’s well temperament with 5 pure 11/7 and 3 near just 11/6
17
!
 66.74120
144.85624
214.44090
278.33864
353.61023
428.88181
492.77955
562.36421
640.47925
707.22045
771.11819
849.23324
921.66136
985.55910
1057.98722
1136.10226
2/1
```

Here is the **microabc** instructions file (`approx.txt`):

```
{---------begin----------}
{      approx.txt        }
inputscl: secor17wt.scl
sagittalapprox: 17
sagittalbasefreq: 440 A
basefreq: 440 13
key: -13
1
{----------end-----------}
```

According to the `approx.txt` file, **microabc** reads the tuning from `secor17wt.scl`. The instruction "`sagittalapprox: 17`" tells **microabc** that the pitches of the scale are close to the pitches of 17-EDO; therefore, Sagittal notation for 17-EDO will be used as base for the approximations.[40] Commands "`sagittalbasefreq: 440 A`" and "`basefreq: 440 13`" adjust the reference frequencies. In this example, the tuning is "transposed downwards" by 13 steps (`key: -13`). The line containing only the character `1` is just to create some "pitch name" in the scale; otherwise, **microabc** would not store the scale.

Below is the music file `approx.abp`.

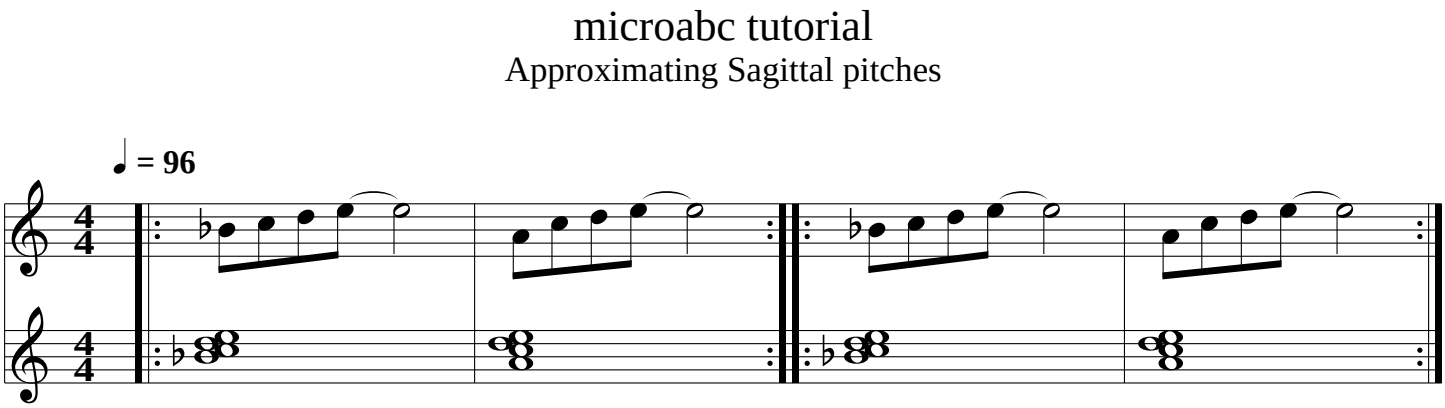
```
X:1
T:microabc tutorial
T:Approximating Sagittal pitches
Q:1/4=96
K:C
V:1
%% MIDI program 7
|: [Bb][C‘][D‘][E‘]- [E‘]4 | [A][C‘][D‘][E‘]- [E‘]4 :|\
|: [Bb][C‘][D‘][E‘]- [E‘]4 | [A][C‘][D‘][E‘]- [E‘]4 :|
V:2
%% MIDI program 79
%% MIDI makechordchannels 3
|: [[Bb][C‘][D‘][E‘]]8 | [[A][C‘][D‘][E‘]]8 :|\
|: [[Bb][C‘][D‘][E‘]]8 | [[A][C‘][D‘][E‘]]8 :|
```

To preprocess the music, use the flag `-S`. Here are the commands to obtain a `MIDI` file:

```
microabc -iapprox.txt -S -Mapprox.abp -oapprox-midi.abc
abc2midi approx-midi.abc
```

and to generate a `PostScript` score:

```
microabc -i- -S -e -Papprox.abp -oapprox-ps.abc
abcm2ps approx-ps.abc -O=
```



[40] The argument for `sagittalapprox:` can be the number of octave divisions (an integer) or the fifth size (as ratio or in cents).

5.7 Just intonation ratios

microabc can preprocess pitches described as just intonation (JI) ratios (relative to `basefreq:`). The preprocessing of JI ratios is activated by the command line option `-J`. That only works in microtonal mode.

The ratios must be enclosed between delimiters (default: `[` and `]`) and obey the format < *num* > / < *den* > [< *oct* >] where < *num* > and < *den* > are integers and < *oct* > is a string of octave indicators (defaults: `'` for octave up and `,` for octave down).

```
X:1
T:microabc tutorial
T:Just intonation ratios
Q:1/4=66
K:C
%%staves (1 2) 3
%%stretchlast
%%microabc: basefreq: 261.63 % frequency of [1/1]
V:1
[5/4][81/64] [9/7][81/64] [5/4][6/5] [5/4]2 | [1/1']2 z2 z4 |]
V:2
[3/2,]4 - [3/2,][4/5]  [3/2,]2 | [5/4]2 z2 z4 |]
V:3
[1/1,]4 - [1/1,][4/5,] [1/1,]2 | [1/1,,]2 z2 z4 |]
```

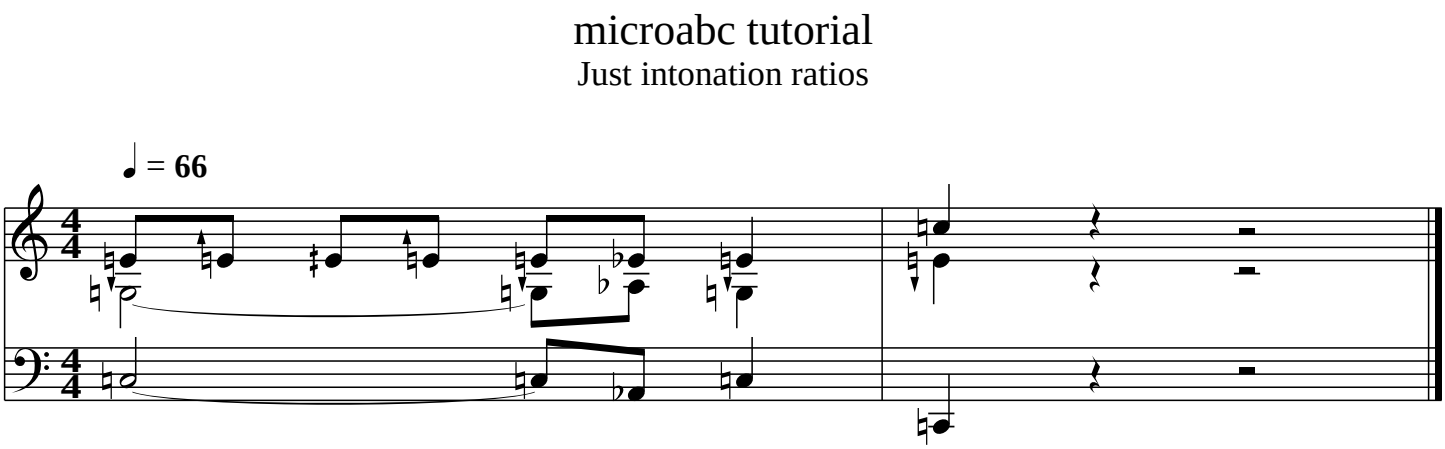
To preprocess the music, use the flag `-J`. Here are the commands to obtain a `MIDI` file:

```
microabc -i- -J -Mjiratios.abp -ojiratios4midi.abc
abc2midi jiratios4midi.abc
```

The JI ratios preprocessor does not support any defined notation system for scores, therefore the only option is to quantise the pitches, for example, to 8th-tones. To generate a `PostScript` score:

```
microabc -i- -J -Pjiratios.abp -ojiratios4ps.abc den:8 abcm2ps:4
abcm2ps jiratios4ps.abc -O=
```

Here is the resulting score:



5.8 More possibilities

There are many other interesting things to do with **microabc** and `ABC` music tools. The reader is invited to read the documentation of the programs, and to browse their sample files directories.

6 HTTP addresses

"Making Music With ABC Plus" (Guido Gonzato):
http://abcplus.sourceforge.net/#ABCGuide
"Sagittal – A Microtonal Notation System" (David Keenan & George Secor):
http://www.sagittal.org/
abc2midi: http://ifdo.pugmarks.com/~seymour/runabc/top.html
abcm2ps and **tclabc**: http://moinejf.free.fr
abcpp: http://abcplus.sourceforge.net
ghostscript and **ghostview**: http://www.cs.wisc.edu/~ghost
microabc: http://hudsonlacerda.webs.com
Scala: http://www.huygens-fokker.org/scala/
timidity++: http://timidity.sourceforge.net