A method for creating discrete note names for just intervals and vice versa;

Using the (modified) Ben Johnston Tuning System.

Hans Bezemer

December 28, 2017

Abstract

In this article I want to describe a method to determine the note name of a just interval using the (modified) Ben Johnston tuning system (BJTS). The derived note names are discrete so every interval has his own unique note name. I use the so called 3-limit intervals and note names as a base, here I differentiate from the BJTS where a combination of 3- and 5-limit intervals are used. To determine the right note name, each interval is written as a product of (multiples of) primes. The product is analyzed and the right comma's and chroma's are added to the basename. This process can be reversed, so for any given note name the corresponding ratio can be calculated. I assume that the readers are familiar with (the basics of) just intonation and the overtones / harmonic series.

Table 1: Prime commas

Prime	Note	Overtone ratio	Basenote ratio	comma
5	E^{-}	5/4	81/64	80/81
7	$B^{7\flat}$	7/4	16/9	63/64
11	$F^{\uparrow}$	11/8	4/3	33/32
13	$A^{13\flat}$	13/8	27/16	26/27
17	$C^{17\sharp}$	17/16	2187/2048	2176/2187
19	$E^{19\flat}$	19/16	32/27	512/513
23	$F^{23\sharp}$	23/16	729/512	736/729
29	$B^{29\flat}$	29/16	16/9	261/256
31	B^{7}	31/16	243/128	243/232

1 introduction

1.1 Ben Johnston Tuning system

Ben Johnston[1] uses an absolute system to determine the note name of a certain ratio. He sets C as ¹⁄₁. And he uses the commonly used CDEFGAB note names as basenotes. When needed extra symbols are added to describe the right note name. The symbols correspond with a discrete microtonal step. Each step is associated with a prime number higher then 2, this is explained below. Johnston uses a combination of 3-limit and 5-limit ratios for his base notes: C ¹⁄₁, D ⁹⁄₈, E ⁵⁄₄, F ⁴⁄₃, A ⁵⁄₃, B ¹⁵⁄₈. Although musically this makes sense, it makes things much more complicated when determining a note name. I choose to use the Pythagorean intervals[2], which are all multiples of the prime numbers 3 and 2: C ¹⁄₁, D ⁹⁄₈, E ⁸¹⁄₆₄, F ⁴⁄₃, A ²⁷⁄₁₆, B ²⁴³⁄₁₂₈.

1.2 One step at a time

Alright, time to get moving in the world of just intonation. In this world it is possible to take all sorts of steps. Small steps, bigger steps. Steps forward, steps backwards. Single steps, multiple steps. But one thing is defined: the size of a specific step is fixed.
For every step size a new prime number is involved. To calculate the steps I used the prime harmonics from the harmonic series (⁵⁄₄, ⁷⁄₄, ¹¹⁄₈, ¹³⁄₈, ¹⁷⁄₁₆, ¹⁹⁄₁₆, ²³⁄₁₆ and ³¹⁄₁₆) and the corresponding name as listed in a list of intervals[3]. Then the difference between the prime harmonic interval and the 3 limit basenote is calculated, see table 1. I could have used more primes but my rationale is that I want to be able to tune intervals by ear, and the higher the prime the harder it gets. Using primes up to 31 is way more then I can hear, but this number is arbitrary. Notice that for each prime comma there are specific symbols to indicate that a basenote is raised or lowered with a certain step. When a basenote is lowered with a comma or chroma, Johnston inverts the symbol for primes 7 and higher: ㄥ ↓ ƐI ㄥI 6I ƐZ 6Z IƐ. For primes 3 and 5 a '$^{-}$' or a '♭' is used.

1.3 Ratios as products of primes

The key to determine the note name is to write a ratio as a product of primes. For instance ³⁄₂ can be written as $2^{-1} \cdot 3^{1}$, or a more complex example: ²¹⁄₂₀ can be written as $2^{-1} \cdot 3^{1} \cdot 5^{-1} \cdot 7^{1}$. This way any ratio build of primes ≤ 31 can be written as: $2^{a} \cdot 3^{b} \cdot 5^{c} \cdot 7^{d} \cdot 11^{e} \cdot 13^{f} \cdot 17^{g} \cdot 19^{h} \cdot 23^{i} \cdot 29^{j} \cdot 31^{k}$.
But why should we want to do this?
Since we are in a logarithmic environment, calculating a new ratio is done by multiplying or dividing two ratios. But multiplying for instance 2^{3} with 2^{2} equals $2^{3+2} = 2^{5}$, thus the powers may be summed together. When we make a matrix of only the powers of the primes, every ratio can be written as a matrix. For instance: ²²⁄₂₁ can be written as $2^{1} \cdot 3^{-1} \cdot 5^{0} \cdot 7^{-1} \cdot 11^{1} \cdot 13^{0} \cdot 17^{0} \cdot 19^{0} \cdot 23^{0} \cdot 29^{0} \cdot 31^{0}$ (note that x^{0} equals 1, so all unused primes have a power of '0'). Then only the powers are written: ²¹⁄₂₀ $\rightarrow \begin{pmatrix} 1 & -1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$.
When two ratios are put together to create a new one, the two matrices only have to be summed or subtracted, which is a relatively simple action, especially for a computer. This will come in handy when determining the note name.

2 Determining the note name

I will describe the steps needed to determine the note name for any given ratio. To illustrate the process I've added pseudocode in C^{++} style.

[1] see John Fonville's article on the BJTS: `sacredrealism.org/catlamb/tuninginformation/articles/Fonville,%20John%20-%20Ben%20Johnston's%20Extended%20Just%20Intonation,%20A%20Guide%20for%20Interpreters.pdf`

[2] Mr. J. Monzo suggests the same, see `http://www.tonalsoft.com/enc/h/hewm.aspx` .

[3] `https://en.wikipedia.org/wiki/List_of_pitch_intervals`

2.1 Converting the ratio to a power of primes matrix

A given ratio must first be analyzed. Which primes are involved and how many times are they used? The way to do this is to:

1. split the ratio into its numerator and denominator
2. analyze both for available primes by dividing them with each prime and count the times an integer is formed
3. create a matrix for both
4. subtract the matrix of the denominator from the numerator

Again, let's take the ratio $^{22}/_{21}$ as our example and put it in our code[4]:

```
int rationum = 22;
int ratioden = 21;
int NRPRIMES = 11;
int PRIMES[NRPRIMES]={
     2,3,5,7,11,13,17,19,23,29,31};
int powofprimes[NRPRIMES]={0};

for(n=0; n<NRPRIMES; n++){
     float tempnumfl=0;
     int tempnumint=0;
     int i=0;
     while (tempnumint == tempnumfl){
     tempnumfl = rationum/PRIMES[n];
     tempnumint = tempnumfl;
     i++;
     }

     float tempdenfl=0;
     int tempdenint=0;
     int j=0;
     while (tempdenint == tempdenfl){
     tempdenfl = ratioden/PRIMES[n];
     tempdenint = tempdenfl;
     j++;
     }
     powofprimes[n]=i-j;
}
```

Now we have an array called 'powofprimes' which contains the powers of primes of the given ratio: $[\ 1\ \ -1\ \ 0\ \ -1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ ]$. [5]

2.2 Determining the basenote and adding the commas

Lets recall what happens in the world of just intonation. We start from a basenote and from that basenote we can take several steps backward and / or forward of different sizes. Each specific size involves a prime. When we've taken all the steps we want, we arrive at our new unique note. This can also be reversed and that is what we're gonna do now.

When we look at the matrix we notice that several prime numbers are involved: 2, 3, 7 and 11. Some are part of the numerator (indicated by the positive integers in the matrix) and some are part of the denominator (indicated by the negative integers in the matrix) of the ratio. The other prime numbers are out of the game for this ratio and their powers are set to '0'. When we look at our example, there were steps made which involved primes 2, 3, 7 and 11. Notice that in table 1 there are no step sizes determined for primes 2 and 3. 2 is only used to get a ratio between two octaves ($^1/_1$ and $^2/_1$) and 3 is our reference prime, because we use the Pythagorean tuning for our base notes.

Table 2: Example subtraction of comma's

$$
\begin{array}{r}
(\ 1\ \ -1\ \ 0\ \ -1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ )\\
+(\ -6\ \ 2\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ )\\
-(\ -5\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ )\\
\hline
(\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ )
\end{array}
$$

We now know that a negative step size 7 and a positive step size 11 was taken. Each step size is also a ratio and can therefore be written as a matrix. These matrices can be summed or subtracted from our example matrix, as shown in table 2. This results (for this example) in the matrix $(\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ )$ which equals $^1/_1$. C has a ratio of $^1/_1$, so our basenote is C, with a 7 step subtracted and a 11 step summed. This gives the following note name: $\mathrm{C}^{\angle\uparrow}$. When we put this in code we get something like this[6].

```
int powofprimes = {1,-1,0,-1,1,0,0,0,0,0,0}
static int scale3base[7][NRPRIMES] = {
     {2,-1,0,0,0,0,0,0,0,0,0}, //F 0
     {0,0,0,0,0,0,0,0,0,0,0}, //C 1
     {-1,1,0,0,0,0,0,0,0,0,0}, //G 2
     {-3,2,0,0,0,0,0,0,0,0,0}, //D 3
     {-4,3,0,0,0,0,0,0,0,0,0}, //A 4
     {-6,4,0,0,0,0,0,0,0,0,0}, //E 5
     {-7,5,0,0,0,0,0,0,0,0,0} //B 6
};
static int commas[NRPRIMES,NRPRIMES] = {
     {1,0,0,0,0,0,0,0,0,0,0},
     {-11,7,0,0,0,0,0,0,0,0,0},
     {-4,4,-1,0,0,0,0,0,0,0,0},
     {-6,2,0,1,0,0,0,0,0,0,0},
     {-5,1,0,0,1,0,0,0,0,0,0},
     {-10,4,0,0,0,1,0,0,0,0,0},
     {7,-7,0,0,0,0,1,0,0,0,0},
     {-9,3,0,0,0,0,0,1,0,0,0},
     {5,-6,0,0,0,0,0,0,1,0,0},
     {-8,2,0,0,0,0,0,0,0,1,0},
     {3,-5,0,0,0,0,0,0,0,0,1}
};
int basenotemat[NRPRIMES]={{0}};

for(int i=0; i<NRPRIMES; i++){
     for(int j=0; j<NRPRIMES; j++){
     basenotename[j]=
     basenotename*commas[i][j]*powofprimes[i];
     }
}
```

2.3 displaying the note name

With the basenote determined and the 'powofprimes'-matrix at hand we can easily display the right note name. If needed the

[4] there has to be checked if the ratio doesn't contain a prime number higher then 31, but that is not a problem to do so.

[5] Note that this method makes the ratio as small as possible, because powers of the small prime are subtracted from each other. For instance $^8/_6$ will be simplified to $^4/_3$.

[6] This have to expanded with a check for flats and sharps, which a present when the power of prime 3 is less then -1 or more then 6

difference between the ratio of the basenote and the ratio can be calculated and converted into an amount of cents.