IDEA: Navigating Spaces through Just-Intonation

Thomas Carroll

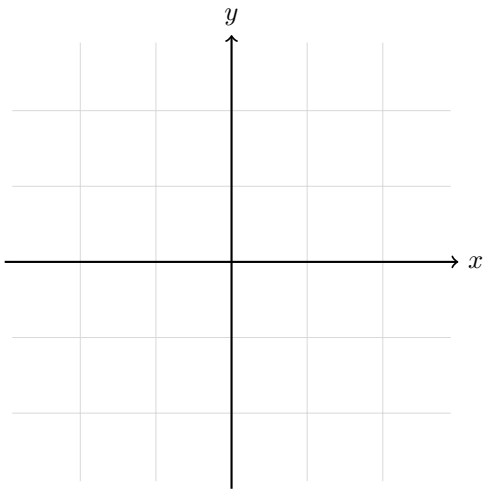
1 Introduction

An *interval* is a ratio of frequencies expressed as $\frac{f_1}{f_2}$, we're attempting to generate these. We are using the *just-intonation* system which is simply creating a system of tuning using these whole-number ratios. A p-limit is a system where you limit your intervals to those which prime-factorisation's highest prime is p. 5-limit is probably the first tuning system theorised about in the West, based on Pythagoras' experiments in tuning.

Instruments in the west tend to be tuned using a system called 12-EDO, EDO meaning Equal Division of Octave. The motivation for using these is that every interval consists only of members of the harmonic series, and thus is 'perfectly in tune'; EDO arrived out of a problem in tuning in 5-limit being the 'pythagorean comma' essentially an error that results from the octave being larger than it should be. EDO smooths this error out over the whole range making things sound 'good enough'.

The important part here is that we can generate ratios of frequencies and that integer ratios are 'good' i.e. a part of the harmonic series and thus should sound consonant.

2 In 2D



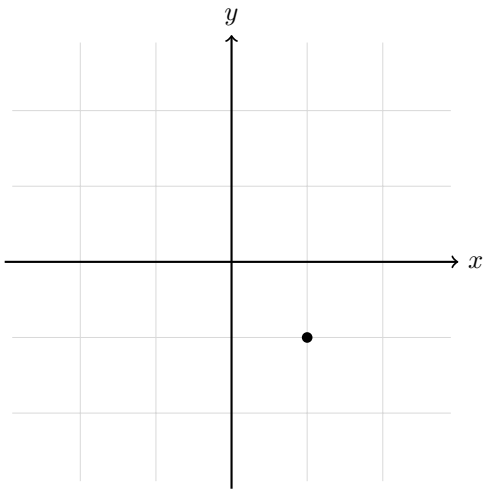
Given two directions x and y, we can create a ket-vector $|*\,x\,y\rangle$ which represents $2^n 3^x 5^y$, where n is any number, since there will be an equivalent interval possible i.e. $|1\,0\,0\rangle$ and $|2\,0\,0\rangle$ are 2 and 4 respectively but this corresponds to an octave (i.e. doubling the pitch), equally we can halve the interval until it's within the octave we choose.

As an example $3^1 \cdot 5^1 = 15$ which is a very large interval but doesn't make much sense as it can be reduced to $\frac{15}{8}$ (major 7th) by adding a term 2^{-3} which brings it back into the same octave . The interval is still a major 7th between octaves but some level of normalisation needs to be present for it to be within the same octave. These names, slightly confusingly come from the number of staff positions they take up (when normalised to one octave). As a rule we want the interval to look like $1 \leq \frac{f_1}{f_2} < 2$, if the interval is less than 1 multiply by 2 if it is greater than 2 then divide by 2.

2.1 Example

For all of the examples, establish a base note, for ease assume something like $A4 = 440\text{Hz} = f_1$. Perhaps this could change with a some sort of generative compositional element independent from the actual image on screen.

Now 'move' into the space somehow $x = 1$, $y = -1$.



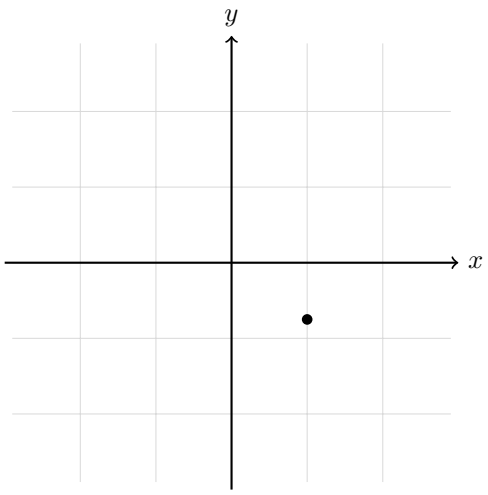
This is represented in the vector $|0\,1\,-1\rangle$ which corresponds to $2^0 3^1 5^{-1} = \frac{6}{5}$. $\frac{6}{5}$ is a minor-third and thus the gap is 3 'semitones'. We should expect a 'C'. Quotes here because we're not talking about the 12-EDO system, instead just intonation. 5-limit just happens approximate the intervals well (partly because of the history of how the western tuning system was developed).

$$\frac{6}{5} \cdot f_1 = f_2$$
$$\frac{6}{5} \cdot 440\text{Hz} = 528\text{Hz}$$

Or in general we can do this:

$$2^n \cdot 3^x \cdot 5^y \cdot f_1 = f_2$$

Thus if we get non-integer numbers for x and y we can calculate the frequencies too.



Where $x = 1$ and $y = 0.75$ we can simply find the frequency as $2^0 \cdot 3^1 \cdot 5^{-0.75} \cdot 440\text{Hz} = 394.772\text{Hz}$. This will sound dissonant and not particularly nice, perhaps the image may look disordered.

The idea here is simple: tones will be consonant in some 'focal point' and dissonant in-between, hopefully this will allow there to be 'touchstone' points in a space that allow them to be navigated. These will be the points on some grid where the parameters are integers.

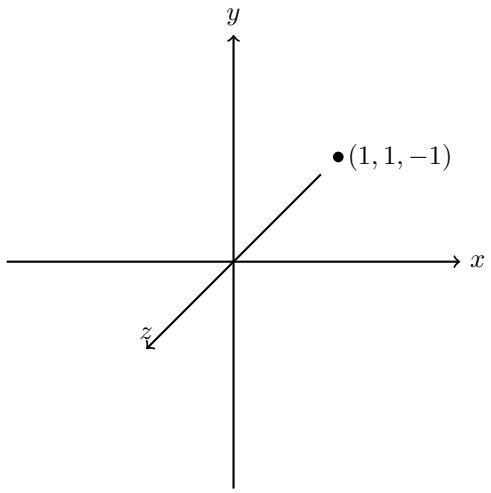
3 In Higher Dimensions

This idea extends into higher dimensions naturally, simply adding to the number of terms in the vector $|*\,e_3\,e_5\,\cdots\,e_p\rangle$ which then gives the ratio as $2^n \cdot 3^{e_3} \cdot 5^{e_5} \cdots p^{e_p}$ where p is the p-th prime number. Each of these

exponents could represent some parameter added to the program. It's worth nothing, here too, this would be called p-limit tuning.

3.1 3D example

Now we use 7-limit tuning with a vector like $|* \, x \, y \, z\rangle$



Thus we get a value of $3^1\dot{5}^1 \cdot 7^-1 \cdot 440\text{Hz} = 942\text{Hz}$ which we can normalise back into the octave with 2^{-1} if we wish this then becomes $\frac{15}{14}$ which is a small interval (the internet tells me it is 'septimal diatonic semitone' whatever that is).

4 Drawbacks

The immediate drawback I can see is that as more parameters are added the less 'strongly consonant' integer-values of the parameters would be. This is mostly a cultural thing, as we are not used to higher-limit tuning, but it's also worth noting that every interval possible in a limit below the currently chosen limit is possible so adding more parameters doesn't shut off the possibility of only exploring one or two of them at a time, creating more obviously consonant intervals.

5 Example Code

The 1px red dots show the points as which the interval is an integer ratio, I have also reduced the frequency to stay within one octave of the base frequency. Thus getting 'sweeps' as you navigate, this is something that can be looked into perhaps?

Navigating to a red dot and being exactly on it, you can hear the consonance vs if you move one unit in any direction, you hear a 'beating' that means you're out of tune.

```
import processing.sound.*;

SinOsc base;
SinOsc variable;
float baseFreq = 440;
// they are in unity to start with
float variableFreq = 440;

float volume = 0.2;
float s_x,s_y,x,y = 0;
float interval;

void settings() {
  size(501, 501);
}

void setup() {
```

```
  base = new SinOsc(this);
  base.play();
  base.freq(baseFreq);
  base.amp(volume);

  variable = new SinOsc(this);
  variable.play();
  variable.amp(volume);
}

void draw() {
  background(255);
  translate(width/2, height/2);
  scale(1, -1);

  // make some axes
  stroke(0);
  strokeWeight(1);
  line(0,height/2,0,-height/2);
  line(width/2,0,-height,0);

  strokeWeight(1);
  point(x,y);

  // mark where the integer values will be, for demo reasons
  stroke(255,0,0);
  strokeWeight(1);
  for(int i = -3; i <= 3; i++) {
    for(int j = -3; j <= 3; j++) {
      point(map(i, -3, 3, -width/2, width/2), map(j, -3, 3, -height/2, height/2));
    }
  }

  // scale x and y to some reasonable numbers -3 -> 3
  s_x = map(x, -width/2, width/2, -3, 3);
  s_y = map(y, -height/2, height/2, -3, 3);
  // now we need to find a way to scale until we're in the same octave
  interval = pow(3,s_x) * pow(5,s_y);

  while (interval < 1 || interval >= 2) {
    if (interval < 1) {
      interval = interval * 2;
    } else if (interval > 2) {
      interval = interval / 2;
    }
  }

  variableFreq = interval * baseFreq;
  variable.freq(variableFreq);
}

void keyPressed() {
  if (key == 'w' || key == 'W')
    y++;

  if (key == 's' || key == 'S')
    y--;

  if (key == 'a' || key == 'A')
    x--;
```

```
    if (key == 'd' || key == 'd')
      x++;
}
```