Approximating Omega (redux)
an alternative version of *Approximating Omega*
(use the original as reference)

The Battery of Sounds:
The original version of Approximating Omega requires a battery of 26 different sounds. As an alternative such that the piece can be performed by fewer people, 24 of the sounds can be produced by playing triples on 3 different instruments using a ternary code to determine the triple. Instruments 0 and 1 should remain an open and closed chime, respectively, as in the original. Then, 2 through 25 can be mapped to {002, 010, 011, 012, 020, 021, 022, 100, 101, 102, 110, 111, 112, 120, 121, 122, 200, 201, 202, 210, 211, 212, 220, 221, 222} where the numbers in each triple indicate the instruments. Instead of playing a unique instrument where the original score indicates 2 through 25, the performer plays a unique ordering of 3 sounds in order specified by the mapping. The triple should be played in rapid succession followed by pause to ensure that they are perceived as a single unit. Other mappings can be explored, for example simultaneously playing combinations of instruments such that each combination is unique and represents one of 2 through 25 in the original.

The Sustained Tones of Section 2:
In the original, the sustained tones are differentiated by octave equivalents of prime harmonics using a D as the fundamental (assuming that the chimes sound a D as well, which, in this version, can also be different). This was to ensure that multiple performers never produce the same pitch when transposing by octaves in order to place the pitch within the range of their respective instruments. It is more important that each tone is unique. Included in this version is a computer program* that automatically generates the sustained tones allowing the original option and an alternate version where the pitches are randomly chosen (assuming that it is highly unlikely that the same pitch will be chosen twice). This reduction along with the one above allows the second section to be performed solo with the computer program. The first section still works better with more performers accompanying the speaker during the reading of the text.

Errata Section 1:
- "...associated list..." in the 4th paragraph should read "...association list..."
- "...because functions with extra arguments..." in the final paragraph should read "...because extra arguments..."

Other Possible Amendments to Section 1:
In the original, the underlining of the word "Symbol" indicates that it is not spoken. However, in this version, whether or not the word is spoken is optional. Also, there are a few references to symbols/functions outside their respective definitions/explanations (the paragraphs between "Lamda" and "Append"). In the original, the corresponding sound is to be played with the spoken reference to the function. In this version, the accompaniment is optional.

***The Computer Program:**
Included with this version is a computer program that plays the sustained parts of Section 2 and can be used as an animated score for the performers. The program indicates the current measure number, the sound number in the original version for that measure as well as the two upcoming measures, and the ternary codes for the respective sound numbers (as explained above). The advancement of the measures can be triggered manually by pressing the "forward" button or automatically by pressing the "start" button.

-*michael winter* (la, 2014)

```
//Approximating Omega (redux) by michael winter. Supercollider 3.6.6. gpl.

(
var measure, ins, code, measureText, insText1, insText2, insText3, codeText1, codeText2, codeText3, gotoText,
forwardButton, backButton, startButton, stopButton, forward, back, updateText, metronome,
primes, powers, oscBank, win, harm, addSine, freeSine, routine, countOff;

SynthDef(\sine, { |num = 1, den = 1, amp = 1.0|
    var freq, fund = 73; //set this to some low octave equivalent of the chimes
    freq = Rand(fund, fund * pow(2, 4)); //for original version, use definition below
    //freq = fund * (num / den) * pow(2, Rand(0, 4).trunc);
    Out.ar([0, 1], SinOsc.ar(fund * (num / den) * pow(2, Rand(0, 4).trunc), 0, Lag.kr(amp)/17))
}).add;

ins = [0, 0, 4, 0, 14, 0, 17, 1, 0, 0, 4, 0, 14, 0, 18, 1, 0, 0, 4, 0, 14, 0, 19, 1, 0, 0, 4, 0,
    14, 0, 20, 1, 0, 20, 0, 4, 0, 21, 1, 1, 1, 1, 1, 0, 4, 0, 14, 0, 25, 1, 0, 9, 3, 0, 9, 5, 0,
    19, 0, 17, 25, 0, 1, 1, 25, 1, 1, 1, 1, 1, 1, 1, 1, 0, 4, 0, 14, 0, 22, 23, 1, 0, 11, 0, 5,
    22, 1, 0, 11, 0, 5, 23, 1, 0, 1, 0, 9, 3, 0, 19, 22, 0, 8, 23, 1, 1, 1, 1, 0, 15, 0, 19, 0,
    8, 22, 1, 0, 8, 23, 1, 1, 0, 9, 0, 7, 22, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 4, 0, 14, 0,
    22, 23, 24, 1, 0, 11, 0, 5, 22, 1, 0, 11, 0, 5, 23, 1, 0, 11, 24, 0, 4, 0, 2, 1, 1, 0, 1, 1,
    0, 18, 0, 4, 0, 3, 1, 1, 23, 24, 1, 1, 0, 11, 0, 5, 23, 1, 0, 18, 22, 0, 4, 0, 3, 1, 1, 24, 1,
    0, 9, 0, 6, 0, 7, 22, 1, 0, 6, 0, 7, 23, 1, 24, 1, 1, 0, 18, 0, 8, 22, 1, 0, 8, 23, 1, 0, 11,
    0, 7, 22, 1, 0, 11, 0, 7, 23, 1, 2, 24, 1, 0, 11, 0, 7, 23, 1, 24, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 0, 4, 0, 14, 0, 22, 23, 1, 0, 11, 0, 5, 22, 1, 0, 11, 0, 5, 0, 7, 0, 13, 25, 0, 4, 0, 12,
    0, 16, 1, 1, 1, 23, 1, 1, 1, 0, 1, 0, 4, 0, 2, 1, 1, 1, 0, 18, 0, 17, 0, 8, 22, 1, 0, 9, 3, 23,
    1, 1, 0, 17, 0, 8, 22, 1, 0, 9, 2, 23, 1, 1, 3, 1, 1, 1, 1, 1, 1];

code = ["0", "1", "002", "010", "011", "012", "020", "021", "022","100", "101", "102", "110", "111",
    "112", "120", "121", "122", "200", "201", "202", "210", "211", "212", "220", "221", "222"];

primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59];
powers = [2, 2, 4, 4, 8, 8, 16, 16, 16, 16, 16, 32, 32, 32, 32, 32, 32];

oscBank = Array.newClear(17);
win = Window.new("Approximating Omega", Rect(128, 64, 1200, 800));
measure = -1;
harm = -1;

measureText = StaticText(win, Rect(10, -200, 600, 600)).string_(measure).align_(\left).font_(Font("Courier-Bold", 200));
insText1 = StaticText(win, Rect(10, 0, 1300, 600)).string_(measure).align_(\left).font_(Font("Courier-Bold", 100));
insText2 = StaticText(win, Rect(410, 0, 1300, 600)).string_(measure).align_(\left).font_(Font("Courier-Bold", 100));
insText3 = StaticText(win, Rect(810, 0, 1300, 600)).string_(measure).align_(\left).font_(Font("Courier-Bold", 100));
codeText1 = StaticText(win, Rect(10, 200, 1300, 600)).string_(measure).align_(\left).font_(Font("Courier-Bold", 150));
codeText2 = StaticText(win, Rect(410, 200, 1300, 600)).string_(measure).align_(\left).font_(Font("Courier-Bold", 150));
codeText3 = StaticText(win, Rect(810, 200, 1300, 600)).string_(measure).align_(\left).font_(Font("Courier-Bold", 150));
```

```
addSine = { harm = harm + 1; oscBank[harm].free; oscBank[harm] = Synth(\sine, [\num, primes[harm], \den, powers[harm]]) };
freeSine = { oscBank[harm].set(\amp, 0); oscBank[harm].free; harm = harm - 1 };
updateText = { measureText.string = measure;
    codeText1.string = code[ins[measure]]; codeText2.string = code[ins[measure+1]]; codeText3.string = code[ins[measure+2]];
    insText1.string = ins[measure]; insText2.string = ins[measure+1]; insText3.string = ins[measure+2] };
forward = { measure = measure + 1; if(ins[measure] == 0, addSine); if(ins[measure] == 1, freeSine); updateText.value };
back = { if(ins[measure] == 1, addSine); if(ins[measure] == 0, freeSine); measure = measure - 1; updateText.value };
countOff = Routine({var delta = 1; metronome.value = 0; delta.yield; metronome.value = 1; delta.yield});

metronome = Button(win, Rect(900, 10, 50, 50)).states_([
    ["", Color.black, Color.red], ["", Color.black, Color.yellow], ["", Color.black, Color.green],]);
backButton = Button(win, Rect(20, 700, 340, 50)).states_([["back", Color.black, Color.red]]).action_(back);
forwardButton = Button(win, Rect(620, 700, 340, 50)).states_([["forward", Color.black, Color.red]]).action_(forward);
startButton = Button(win, Rect(500, 10, 100, 50)).states_([["start", Color.black, Color.red]]).action_({AppClock.play(routine)});
stopButton = Button(win, Rect(700, 10, 100, 50)).states_([["stop", Color.black, Color.red]]).action_({AppClock.clear;
routine.reset});

//this does not work for sure!
gotoText = TextField(win, Rect(470, 700, 40, 30)).action_({ arg text;
    while({ measure < text.value.asInteger }, forward);
    while({ measure > text.value.asInteger }, back)}).string_("go to");

routine = Routine({
    var delta = 1;
    metronome.value = 0; delta.yield;
    metronome.value = 1; delta.yield;
    loop {metronome.value = (metronome.value + 1) % 3; if(metronome.value == 2, forward); delta.yield}
});

win.front;
)
```