Dynamic Smoothing Using Self Modulating Filter

© Andrew Simper, Cytomic, 2014, andy@cytomic.com
public release: 6th Dec 2016

This technical paper outlines a robust and inexpensive dynamic smoothing algorithm based on using the bandpass output of a 2 pole multimode filter to modulate its own cutoff frequency. The bandpass signal is a meaure of how much the signal is “changing” so is useful to increase the cutoff frequency dynamically and allow for faster tracking when the input signal is changing more. The absolute value of the bandpass signal is used since either a change upwards or downwards should increase the cutoff.

The core of the filtering algorithm consists of two one pole low pass forward euler filters. These two filters are in cascade to form a 2 pole low pass filter, and their difference is used to form the band-pass response, which is identical to an SVF bandpass output with damping 2.

This algorithm can be useful for in a variety of situations including “de-noising” an analog potentiometer, or smoothing out the stepped input of a 7-bit MIDI continous controller. It has sufficiently low cpu and numerical stability to be used in fixed precision on micro-controllers, and at a lower than audio sample rate, but care needs to be taken to ensure full range is acheived if using low bit depth computation.

```
Psuedo code efficient
opcount: 6+-, 3*

init:
  basefreq = 2 hz (fine tune to suit)
  sensitivity = 0.5 (fine tune to suit)
  wc = basefreq/samplerate
  gc = tan(pi*wc)
  g0 = 2*gc/(1 + gc)
  sense = sensitivity*4 (efficient linear cutoff mapping)

clear:
  low1 = 0
  low2 = 0

tick:
  low1z = low1;
  low2z = low2;
  bandz = low1z - low2z
  g = min (g0 + sense*abs(bandz), 1)
  low1 = low1z + g*(in   - low1z)
  low2 = low2z + g*(low1 - low2z)
  output = low2
```

```
Psuedo code full (more accurate linear frequency modulation)
opcount: 10+-, 8*

init:
  basefreq = 2 hz (fine tune to suit)
  sensitivity = 2 (fine tune to suit)
  wc = basefreq/samplerate (cubic cutoff mapping done inside algorithm)

clear:
  low1 = 0
  low2 = 0
  inz = 0

tick:
  low1z = low1
  low2z = low2
```

```
bandz = low1z - low2z
wd = wc + sensitivity*abs(bandz)
g = min (wd * (5.9948827 + wd * (-11.969296 + wd * 15.959062)), 1)
low1 = low1z + g*(0.5*(in   + inz  ) - low1z)
low2 = low2z + g*(0.5*(low1 + low1z) - low2z)
inz = in
output = low2
```

Approximation of cutoff gain

$$\text{Simplify}\left[\frac{2\,\text{Tan}[\pi\,\text{wc}]}{\sqrt{(1+\text{Tan}[\pi\,\text{wc}])^2}},\ \text{wc} > 0 \&\& \text{wc} < 1/2\right]$$

```
FullSimplify[%]
```

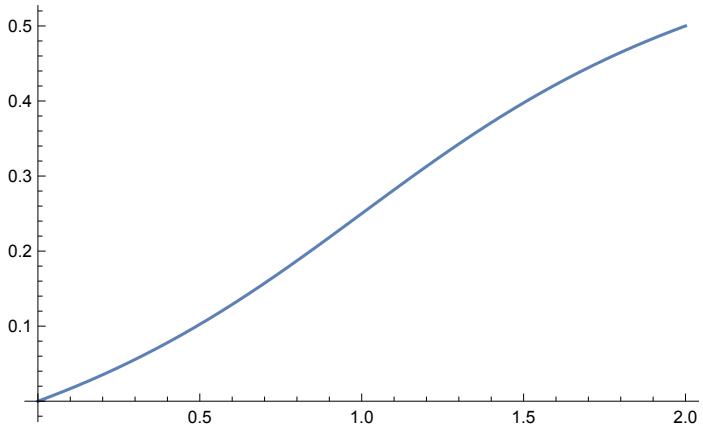
$$\frac{2\,\text{Tan}[\pi\,\text{wc}]}{1+\text{Tan}[\pi\,\text{wc}]}$$

$$2-\frac{2}{1+\text{Tan}[\pi\,\text{wc}]}$$

$$\text{ActualCutoffGain}[\text{wc\_}] := 2-\frac{2}{1+\text{Tan}[\pi\,\text{wc}]};$$

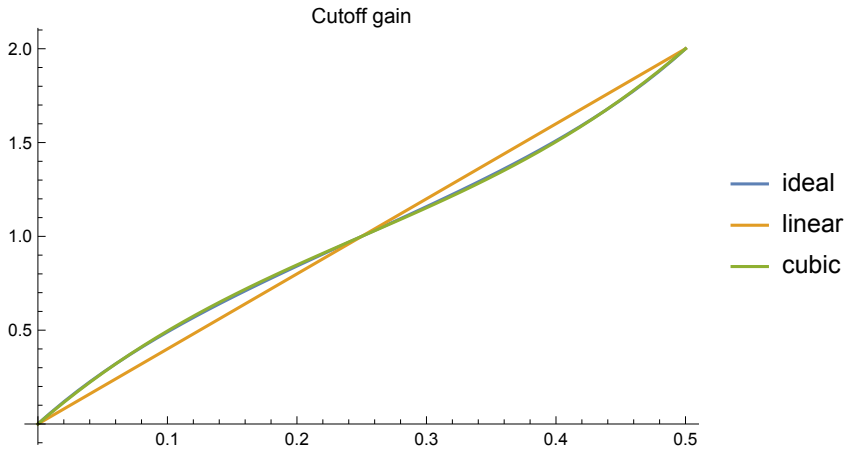
$$\text{CutoffGainToWc}[g\_] = \frac{\text{ArcTan}\left[\frac{g}{2-g}\right]}{\pi};$$

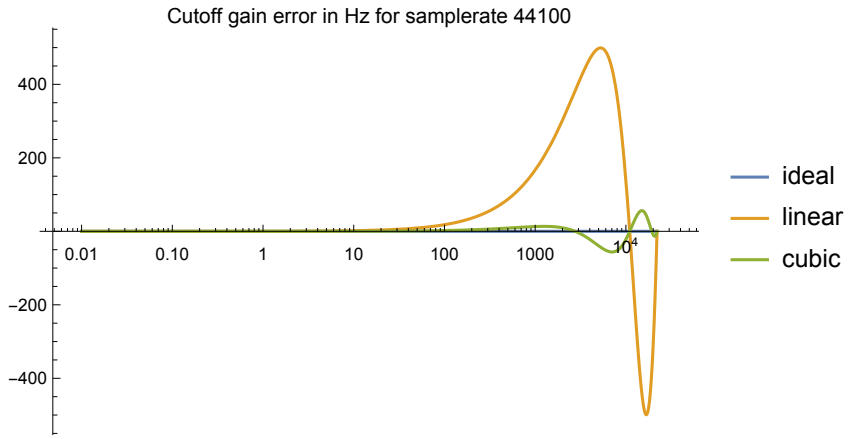
```
Plot[CutoffGainToWc[g], {g, 0, 2}]
f[wc_] := ActualCutoffGain[wc];
g[wc_] := a1 * (wc) + a2 * (wc)^2 + a3 * (wc)^3
s =
 Solve[{f[0] == g[0], f[1 / 16] == g[1 / 16], f[7 / 16] == g[7 / 16], f[1 / 2] == g[1 / 2]},
    {a1, a2, a3}][[1]] // N
ApproximateCutoffGain[wc_] := g[wc] /. s
SetPrecision[ApproximateCutoffGain[wc], 8]

Plot[{ActualCutoffGain[wc], 4 wc, ApproximateCutoffGain[wc]}, {wc, 0, 0.5},
 PlotLabel → "Cutoff gain", PlotLegends → {"ideal", "linear", "cubic"}]
LogLinearPlot[ {0, 22 050
    (CutoffGainToWc[ActualCutoffGain[wc / 44 100]] - CutoffGainToWc[4 wc / 44 100]),
  22 050 (CutoffGainToWc[ActualCutoffGain[wc / 44 100]] -
      CutoffGainToWc[ApproximateCutoffGain[wc / 44 100]])}, {wc, 0.01, 22 050},
 PlotLabel → "Cutoff gain error in Hz for samplerate 44100",
 PlotRange -> All, PlotLegends → {"ideal", "linear", "cubic"}]
```



$$\{a1 \to 5.99488,\ a2 \to -11.9693,\ a3 \to 15.9591\}$$

$$5.9948827\ wc - 11.969296\ wc^2 + 15.959062\ wc^3$$





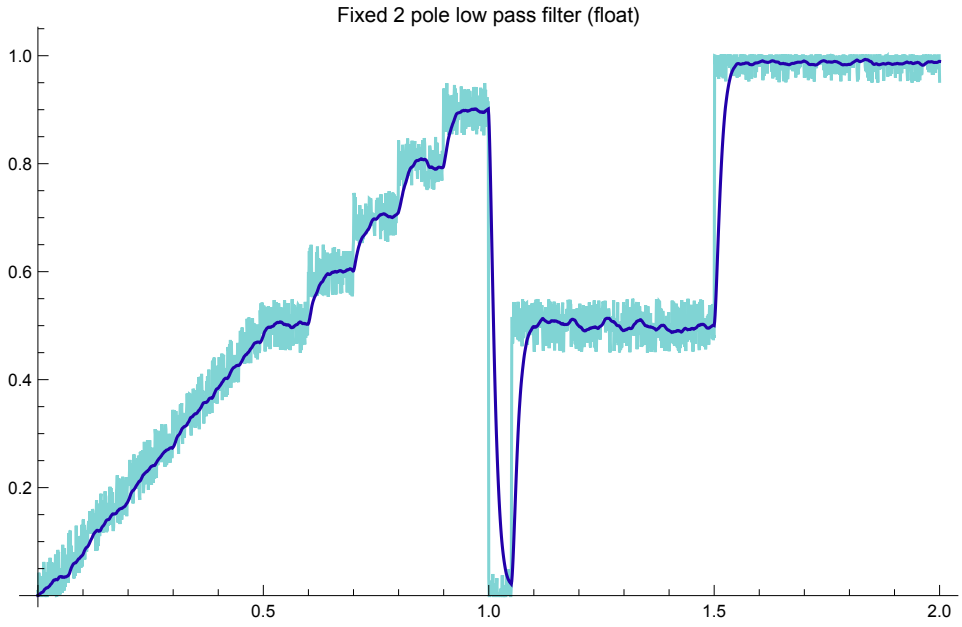
Implementation check

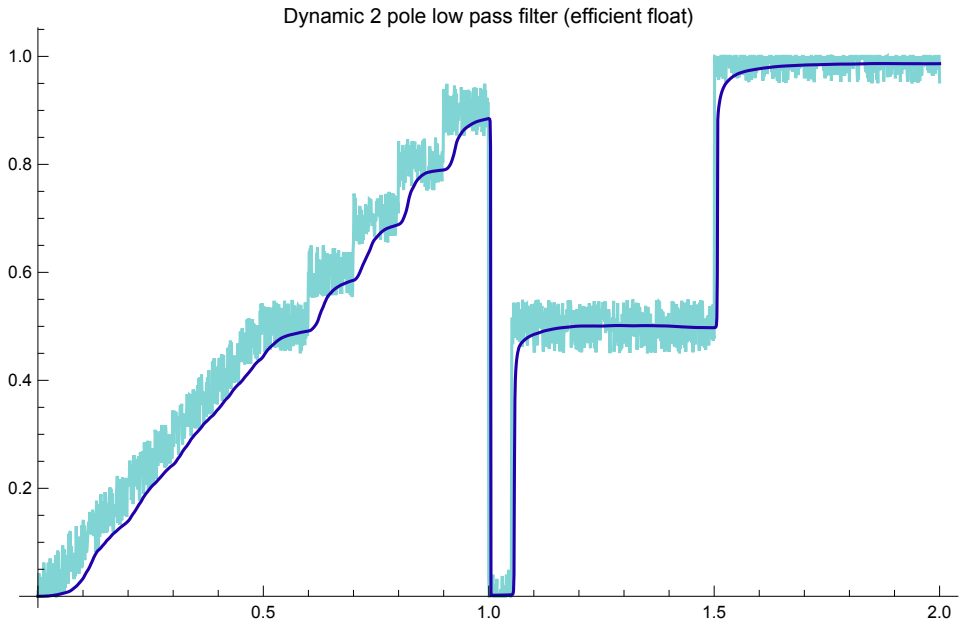
```
Reset[] := Block[{}, band = 0;
    low = 0; low1 = 0; low2 = 0; inz = 0;
    gz = 0];
DynamicSmootherEcon[t_, in_, g0_, sensitivity_] := Block[{g, f, bandz, lowz},
    low1z = low1;
    low2z = low2;
    bandz = low2z - low1z;
    g = Min[g0 + 4 sensitivity * Abs[bandz], 1.0];
    low1 = low1z + g * (in - low1z);
    low2 = low2z + g * (low1 - low2z);
    Return[{t, in, low2, g}];
  ];
DynamicSmootherEcon2[t_, in_, g0_, sensitivity_] := Block[{g, f, bandz, lowz},
    low1z = low1;
    low2z = low2;
    bandz = low2z - low1z;
    g = Min[g0 + 75 sensitivity * bandz * bandz, 1.0];
    low1 = low1z + g * (in - low1z);
    low2 = low2z + g * (low1 - low2z);
    inz = in;
    Return[{t, in, low2, g}];
  ];
```

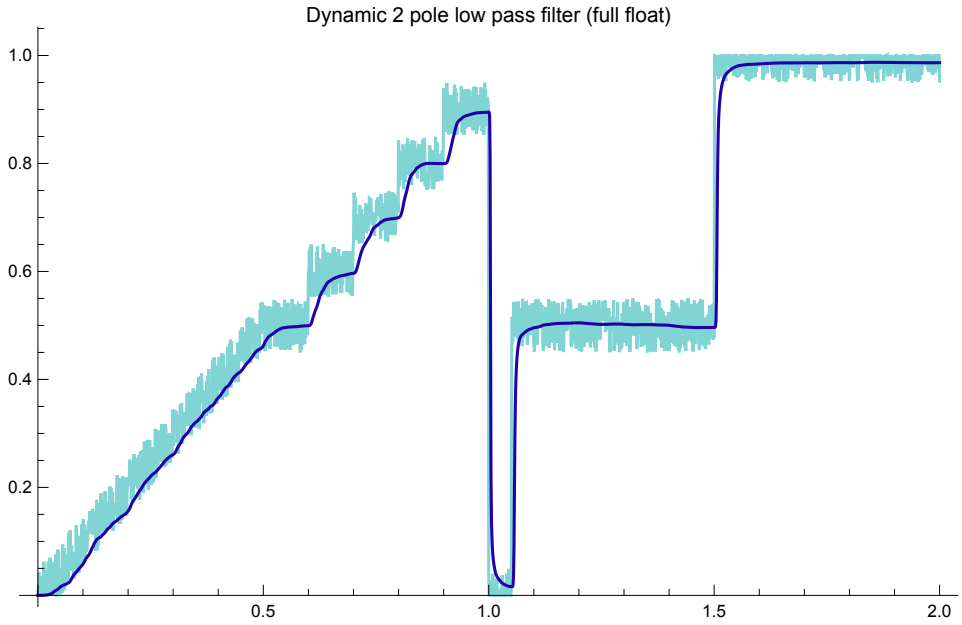
```
DynamicSmoothherFull[t_, in_, w0_, sensitivity_] := Block[{g, f, bandz, lowz},
   low1z = low1;
   low2z = low2;
   bandz = low2z - low1z;
   wd = w0 + sensitivity * Abs[bandz];
   g = Min[wd * (5.9948827 + wd *
         (-11.969296395805415045288100372999906539 92`8. + wd * 15.959062)), 1];
   low1 = low1z + g * (0.5 * (in + inz) - low1z);
   low2 = low2z + g * (0.5 * (low1 + low1z) - low2z);
   inz = in;
   gz = g;
   Return[{t, in, low2, g}];
  ];
fx = 2^7;
fx2 = BitShiftRight[fx, 1];
ToInt[x_] := Floor[fx x];
FromInt[x_] := (x) / fx;
ScaleMult[x_] := Floor[x / fx];
DynamicSmoothherEconInt[t_, inf_, g0_] :=
  Block[{in, g, low1z, low2z, bandz},
   in = ToInt[inf];
   low1z = low1;
   low2z = low2;
   bandz = low2z - low1z;
   g = Min[ToInt[g0] + Abs[bandz], ToInt[1.0]];
   low1 = low1z + ScaleMult[g * (in - low1z)];
   low2 = low2z + ScaleMult[g * (low1 - low2z)];
   Return[{t, FromInt[in], FromInt[ScaleMult[low2 * ToInt[1.1]]], FromInt[g]}];
  ];
Reset[];
sr = 1000.0;
srinv = 1.0 / sr;
Signal[t_] := Clip[
  If[t > 1.5, 1.0, If[t > 1.05, 0.5, If[t > 0.5, Floor[10 * SawtoothWave[t]] / 10,
      SawtoothWave[t]]]] + RandomReal[{-0.05, 0.05}], {0, 1}]
input = Table[{t, Signal[t]}, {t, 0, 2.0, srinv}];
len = Length[input];

SetOptions[ListPlot, Joined → True,
  PlotStyle → {Darker[Opacity[0.5, Hue[0.5]]],
    Darker[Opacity[1.0, Hue[0.7]]], Darker[Opacity[0.5, Hue[0.9]]]}];

DynamicSmoothherFull[0, 1, 0.1, 0]
Reset[];
g0 = ActualCutoffGain[20 srinv];
tp1 =
 Table[DynamicSmoothherEcon[input[[i, 1]], input[[i, 2]], g0, 0], {i, 1, len}];
ListPlot[{tp1[[All, {1, 2}]], tp1[[All, {1, 3}]]},
 PlotLabel → "Fixed 2 pole low pass filter (float)"]

Reset[];
g0 = ActualCutoffGain[2 srinv];
tp2 = Table[
   DynamicSmoothherEcon2[input[[i, 1]], input[[i, 2]], g0, 0.5], {i, 1, len}];
ListPlot[{tp2[[All, {1, 2}]], tp2[[All, {1, 3}]]},
 PlotLabel -> "Dynamic 2 pole low pass filter (efficient float)"]

Reset[];
```

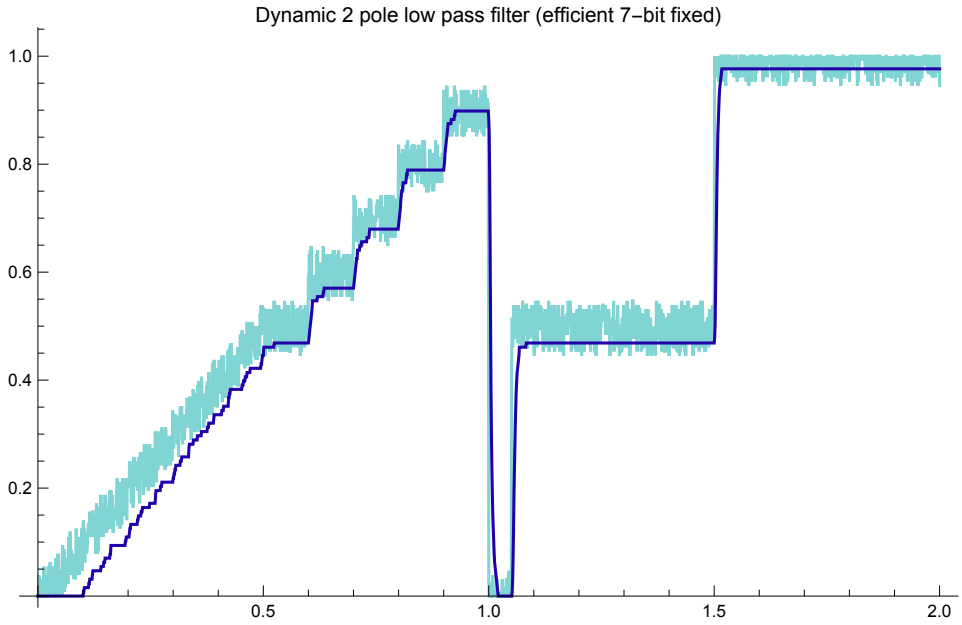
```
w0 = 2 srinv;
tp3 = Table[
   DynamicSmootherFull[input[[i, 1]], input[[i, 2]], w0, 0.5], {i, 1, len}];
ListPlot[{tp3[[All, {1, 2}]], tp3[[All, {1, 3}]]},
 PlotLabel -> "Dynamic 2 pole low pass filter (full float)"]

Reset[];
g0 = ActualCutoffGain[15 srinv];
tp4 = Table[
   DynamicSmootherEconInt[input[[i, 1]], input[[i, 2]], g0], {i, 1, len}];
ListPlot[{tp4[[All, {1, 2}]], tp4[[All, {1, 3}]]},
 PlotLabel -> "Dynamic 2 pole low pass filter (efficient 7-bit fixed)"]
ListPlot[{tp1[[All, {1, 4}]], tp2[[All, {1, 4}]], tp3[[All, {1, 4}]]},
 PlotLabel → "Dynamic and fixed cutoff gain factor", PlotStyle → Automatic,
 PlotRange → All, PlotLegends → {"fixed", "dynamic efficient", "dynamic full"}]
```

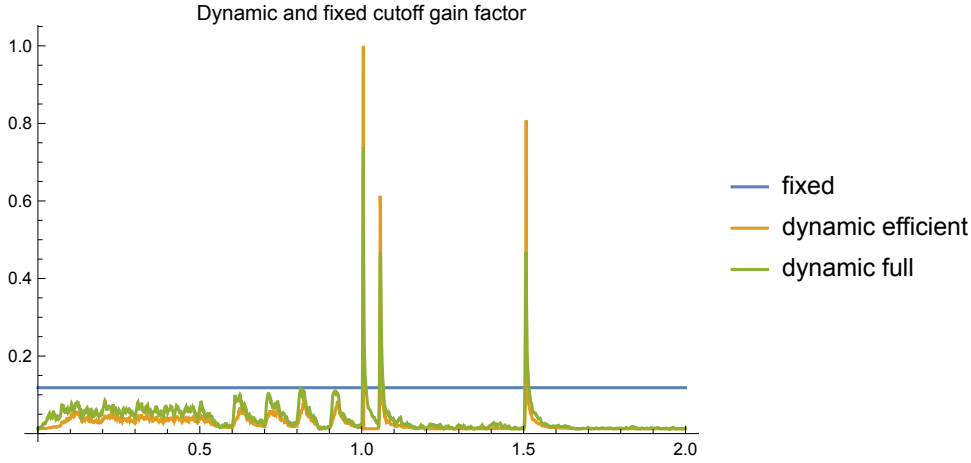
```
{0, 1, 0.0614431, 0.495754}
```





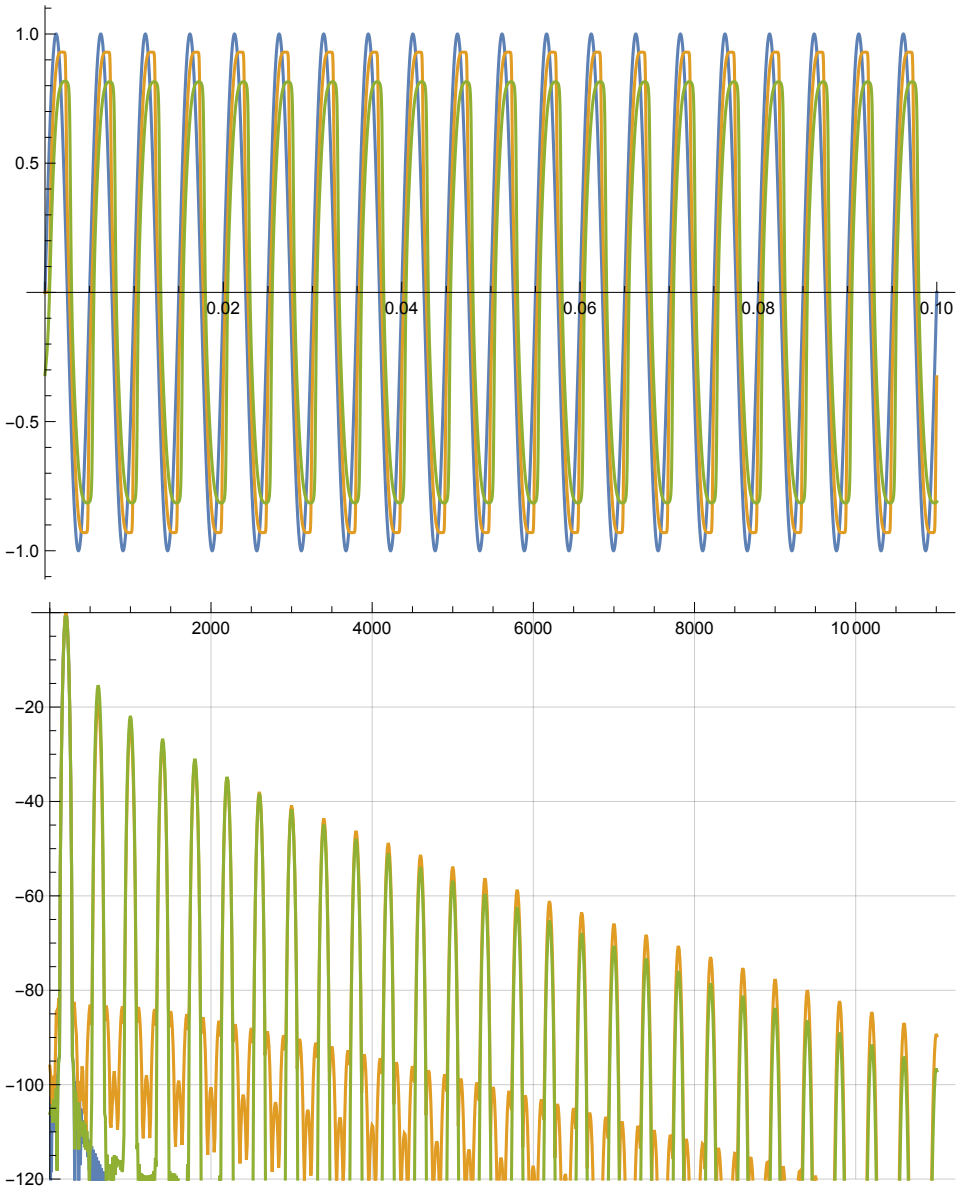







```
ClearAll["Global`*"];
xz = 0;
```

```
y = 0;
low1 = 0;
low2 = 0;

Tick1[t_, x_] := Block[{low1z, low2z, bandz, g},
    low1z = low1;
    low2z = low2;
    bandz = low2 - low1;
    g = Min[g01 + sense1 Abs[bandz], 1];
    low1 = low1z + g (x - low1z);
    low2 = low2z + g (low1 - low2z);
    Return[{t, x, low2}];
  ];
Tick2[t_, x_] := Block[{low1z, low2z, bandz, g},
    low1z = low1;
    low2z = low2;
    bandz = low2 - low1;
    g = Min[g02 + sense2 bandz bandz, 1];
    low1 = low1z + g (x - low1z);
    low2 = low2z + g (low1 - low2z);
    Return[{t, x, low2}];
  ];
g01 = 0.001;
g02 = 0.01;
sense1 = 1;
sense2 = 1;


t1 = 0;
t2 = 0.1;
sr = 44 100.0;
h = 1.0 / sr;

dB[x_] := 20 Log[10, Abs[x] + 1*^-40];
KaiserWindowArray[len_, beta_] :=
  Table[KaiserWindow[(2 * x + 1) / (len), beta], {x, -len / 2, len / 2 - 1}];
SetOptions[ListPlot, Joined → True, ImageSize → Scaled[0.8],
  PlotStyle → Automatic];

infreq = 200;
tp1 = Table[Tick1[t, 1 Sin[2 π t infreq]], {t, t1, t2, h}];
tp2 = Table[Tick2[t, 1 Sin[2 π t infreq]], {t, t1, t2, h}];
len = Length[tp1];
ListPlot[{tp1[[All, {1, 2}]], tp1[[All, {1, 3}]], tp2[[All, {1, 3}]]}]
ftp0 =
  dB[Take[Fourier[tp1[[All, 2]] KaiserWindowArray[len, 12]], Floor[len / 4]]];
ftp1 = dB[Take[Fourier[tp1[[All, 3]] KaiserWindowArray[len, 12]],
    Floor[len / 4]]];
ftp2 = dB[Take[Fourier[tp2[[All, 3]] KaiserWindowArray[len, 12]],
    Floor[len / 4]]];
m0 = Max[ftp0];
ftp0 -= m0;
ftp1 -= m0;
ftp2 -= m0;
freqs = Table[sr n / len, {n, 0, Floor[len / 4] - 1}];
ftp0 = MapThread[List, {freqs, ftp0}];
ftp1 = MapThread[List, {freqs, ftp1}];
ftp2 = MapThread[List, {freqs, ftp2}];
ListPlot[{ftp0, ftp1, ftp2}, Joined → True,
 PlotRange → {-120, 1}, ImageSize → Scaled[0.8], GridLines → Automatic]
```


1.0
0.5
0.02
0.04
0.06
0.08
0.10
−0.5
−1.0
2000
4000
6000
8000
10000
−20
−40
−60
−80
−100
−120