source: http://www.hydrogenaudio.org/forums/index.php?showtopic=86116&st=25

written by pbelkner

QUOTE (benski @ Jan 17 2011, 20:11)

Correct.

V are gain values and have no relation to sampling frequency
Q is a "magic number" that effects the shape of the filter.

Fc stays constant - it's the nominal cutoff frequency. ω is tan(fc/fs *π) [it has been typo'd as Ω in the paper, ω/Ω are lowercase/uppercase pairs]. That is, it's the cutoff frequency as a percentage of the sampling rate, and "pre-warped" with tan() to match the frequency warping done by the bilinear transform. k is often used for ω in source code.

I've finally managed (thanks to the pointers provided by Raiden) to find a closed solution to the re-quantization problem for digital biquad filters as it appears in the context of (but not restricted to) BS.1770.

Assume youv'e given the coefficients a1, a2, b0, b1, b2, b3 of a digital biquad filter for a particular sample frequency Fs (cf. e.g. http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt).

The re-quantization problem consists in calculating the coefficents a1', a2', b0', b1', b2', and b3' of a digital biquad filter with the same characteristics as the given one, but for another sample frequency Fs'.

The key to the solution are the pointers provided by Raiden:

1. Brian Neunaber: Parameter Quantization in Direct-Form Recursive Audio Filters
2. Raiden: ITU-R BS.1770-1 filter specifications, January 15,

2011

According to [1] the coeffients a1, a2, b0, b1, b2 of the given digital biquad filter can be mapped to the parameters Fc, Q, Vl, Vb, and Vh of the corresponding analog filter by the following equations (using the notation from [2]):

CODE

```
(1)    (1 + K / Q + K^2) * a1  =  2 * (K^2 - 1)
(2)    (1 + K / Q + K^2) * a2  =  1 - K / Q + K^2
(3)    (1 + K / Q + K^2) * b0  =  Vh + Vb * K / Q + Vl * K^2
(4)    (1 + K / Q + K^2) * b1  =  2 * (Vl * K^2 - Vh)
(5)    (1 + K / Q + K^2) * b2  =  Vh - Vb * K / Q + Vl * K^2
```

with

CODE

```
(6) K = tan(pi * Fc / Fs).
```

In order to solve the above stated re-quantization problem we do the following:

1. Solve the eqs (1) - (5) for Fc, Q, Vl, Vb, and Vh.
2. Use the equations

CODE

```
(1')    (1 + K' / Q + K'^2) * a1'  =  2 * (K'^2 - 1)
```
3. `(2')    (1 + K' / Q + K'^2) * a2'  =  1 - K' / Q + K'^2`
4. `(3')    (1 + K' / Q + K'^2) * b0'  =  Vh' + Vb' * K' / Q + Vl * K'^2`
5. `(4')    (1 + K' / Q + K'^2) * b1'  =  2 * (Vl * K'^2 - Vh)`
6. `(5')    (1 + K' / Q + K'^2) * b2'  =  Vh - Vb *`

```
K' / Q + Vl * K^2
```

with

CODE

```
(6') K' = tan(pi * Fc / Fs')
```

in order to calculate the coefficents a1', a2', b0', b1', and b2' for a digital biquad filter with the same characteristcs Fc, Q, Vl, Vb, and Vh as the given one but for a different sample frequency Fs'.

We start by solving eqs. (1) and (2) for K^2 and K/Q, respectively. By substituting

CODE

```
x11  =  a1 - 2
x12  =  a1
x1  =  -a1 - 2

x21  =  a2 - 1
x22  =  a2 + 1
x2  =  -a2 + 1

DX = x22 * x11 - x12 * x21
```

and using well known methods we arrive at

CODE

```
(6)  K^2  =  (x22 * x1 - x12 * x2) / DX
(7)  K/Q  =  (x11 * x2 - x21 * x1) / DX
```

Next we solve eqs. (3), (4), and (5) for Vh, Vb, and Vl. Introducing

CODE

```
(8) a0  =  1 + K / Q + K^2
```

and reordering eqs. (3), (4), and (5) they read

CODE

```
(3a)       Vh + K/Q * Vb +       K^2 * Vl  =  b0 * a0
(4a)  -2 * Vh            + (2 * K^2) * Vl  =  b1 * a0
(5a)       Vh - K/Q * Vb +       K^2 * Vl  =  b2 * a0
```

Now it's not hard any longer to find the solutions:

CODE

```
             a0 * (b0 - b2)
(9)  Vb  =  --------------
                2 * K/Q

             a0 * (b0 + b1 + b2)
(10) Vl  =  -------------------
                   4 * K^2

             a0 * (b0 - b1 + b2)
(11) Vh  =  -------------------
                     4
```

Finally we observe from eqs. (6) and (6') the following:

CODE

```
(6)     K   =  tan(pi * Fc / Fs)
(6)'    K'  =  tan(pi * Fc / Fs')
(12)   K'  =  tan(atan(K) * Fs / Fs')
```

Eqs. (6) and (7) along with (9), (10), (11), and (12) provide everything we need for re-quantizing any given digital biquad filter. We demonstrate this by the following C code (please note that this code re-quantizes digital biqad filters on the fly, no preprocessing by an external algebra system ist needed).

CODE

```
typedef struct biquad {
  double fs;
  double a1, a2;
  double b0, b1, b2;
} biquad_t;

typedef struct biquad_ps {
  double k;
  double q;
  double vb;
  double vl;
  double vh;
} biquad_ps_t;

void biquad_get_ps(biquad_t *biquad, biquad_ps_t *ps)
{
  double x11 = biquad->a1 - 2;
  double x12 = biquad->a1;
  double x1 = -biquad->a1 - 2;

  double x21 = biquad->a2 - 1;
  double x22 = biquad->a2 + 1;
  double x2 = -biquad->a2 + 1;

  double dx = x22*x11 - x12*x21;
  double k_sq = (x22*x1 - x12*x2)/dx;
  double k_by_q = (x11*x2 - x21*x1)/dx;
  double a0 = 1.0 + k_by_q + k_sq;

  ps->k = sqrt(k_sq);
  ps->q = ps->k/k_by_q;
  ps->vb = 0.5*a0*(biquad->b0 - biquad->b2)/k_by_q;
```

```
  ps->vl = 0.25*a0*(biquad->b0 + biquad->b1 + biquad->b2)/k_sq;
  ps->vh = 0.25*a0*(biquad->b0 - biquad->b1 + biquad->b2);
}

biquad_t *biquad_requantize(biquad_t *in, biquad_t *out)
{
  if (in->fs==out->fs)
    return in;
  else {
    biquad_ps_t ps;
    double k, k_sq, k_by_q, a0;

    biquad_get_ps(in, &ps);
    k=tan((in->fs/out->fs)*atan(ps.k));
    k_sq = k*k;
    k_by_q = k/ps.q;
    a0 = 1.0 + k_by_q + k_sq;

    out->a1 = (2.0*(k_sq - 1.0))/a0;
    out->a2 = (1.0 - k_by_q + k_sq)/a0;
    out->b0 = (ps.vh + ps.vb*k_by_q + ps.vl*k_sq)/a0;
    out->b1 = (2.0 * (ps.vl*k_sq - ps.vh))/a0;
    out->b2 = (ps.vh - ps.vb*k_by_q + ps.vl*k_sq)/a0;

    return out;
  }
}
```

The following code demonstrates how to re-quantize the 48 kHz BS.1770 pre-filter to it's 44.1 kHz equivalent using the above functions.

CODE

```c
int main(int argc, char **argv)
{
  int i;

  biquad_t pre48000={
    .fs=48000,
    .a1=-1.69065929318241,
    .a2=0.73248077421585,
    .b0=1.53512485958697,
    .b1=-2.69169618940638,
    .b2=1.19839281085285
  };

  biquad_t pre44100={ .fs=44100 };

  biquad_requantize(&pre48000, &pre44100);
  printf("a1: %f, %f\n", pre48000.a1, pre44100.a1);
  printf("a2: %f, %f\n", pre48000.a2, pre44100.a2);
  printf("b1: %f, %f\n", pre48000.b0, pre44100.b0);
  printf("b2: %f, %f\n", pre48000.b1, pre44100.b1);
  printf("b3: %f, %f\n", pre48000.b2, pre44100.b2);
}
```