A result concerning the zeros of the second derivative of a particular transfer function

Dave Gamble

31st Jan 2013

Placing a simple result into the public domain.

This is also a little insight into what I spent my time doing!

The transfer function $H(s) = \frac{s^2+A/Qs+1}{s^2+1/AQs+1}$ is well known to describe the spectral response of a peaking filter, where Q determines the bandwidth, and $A = 10^{gain/40}$.

Assume now a three-parameter transfer function $T(s) = H(s/k)H(sk)$. Observe that $|T(1)| = A^4$ for $k = 1$.

It can be verified that

$$|T(s)|^2_{s=if} = \frac{(1-fk^2)^2 + (A/Q\ fk)^2}{(1-fk^2)^2 + (1/AQ\ fk)^2} \frac{(1-f/k^2)^2 + (A/Q\ f/k)^2}{(1-f/k^2)^2 + (1/AQ\ f/k)^2} \tag{1}$$

and further that:

$$\frac{d^2|T(s)|^2_{s=if}}{df^2}_{f=0} = \frac{\Big(A^2Q^4k^6 + (-A^2 + 2(1+A^4)Q^2 - 9A^2Q^4)k^4 + (16A^2Q^4 - 4(1+A^4)Q^2)k^3 + (-A^2 + 2(1+A^4)Q^2 - 9A^2Q^4)k^2 + A^2Q^4\Big)}{(k^2 + A^2(k^2-1)^2Q^2)^4} \tag{2}$$

Seeking the roots of this equation, we disregard the denominator, and are left with the equation:

$$\begin{aligned} A^2Q^4k^6 + (2(1+A^4)Q^2 - A^2 - 9A^2Q^4)k^4 + (16A^2Q^4 - 4(1+A^4)Q^2)k^3 \\ +(2(1+A^4)Q^2 - A^2 - 9A^2Q^4)k^2 + A^2Q^4 = 0 \end{aligned} \tag{3}$$

Observing symmetry:

$$A^2Q^4(k^6+1) + (2(1+A^4)Q^2 - A^2 - 9A^2Q^4)(k^4+k^2) + (16A^2Q^4 - 4(1+A^4)Q^2)k^3 = 0 \tag{4}$$

The symmetry of this 6^{th} order polynomial allows us to exploit a change of variable. Let: $M = k^2 + k^{-2}$ and divide through by k^3, to yield:

$$A^2Q^4M^3 + ((2(1+A^4)Q^2 - A^2 - 9A^2Q^4) - 3A^2Q^4)M + (16A^2Q^4 - 4(1+A^4)Q^2) = 0 \tag{5}$$

Note the subtraction of $-3A^2Q^4$ from the linear term to compensate for the algebra of the substitution.

This depressed cubic is directly soluble to find M. Given $M = k^2 + k^-2$, we have $k^4 - Mk^2 + 1 = 0$, which is quadratic in k^2.

Sample implementation:

```
void normalised_T(double *out,double gain_db,double Q,double k_scale)
{
// out is an array of 12 doubles which will be filled with two sets
//      of 6 coefficients each describing coefficients of the analogue
//      transfer function H(s), which together, implemented in series,
//      give T(s). (two factored biquad stages)
// gain_db is gain in decibels, e.g. 6
// Q is proportional to bandwidth
// k_scale is a linear multiplier for k, between 0 and 1 inclusive.


        // Compute inputs. Default k to 1.
        double A=pow(10.0,gain/80.0),k=1,M=0,L=0;
        // Lay out unmodified H(s) prototypes.
        for (int i=0;i<12;i++) out[i]=1.0;
        out[1]=out[7]=A/Q,out[4]=out[10]=1/(A*Q);

        // Precompute
        double A2=A*A;
        double Q2=Q*Q;
        double A4=A2*A2;
        double Q4=Q2*Q2;

        // These are the coefficients of the 6th order polynomial.
        // p2=p4, p0=p6, p1=p5=0
        double p6=A2*Q4;
        double p4=-A2+2*(1+A4)*Q2 - 9*A2*Q4;
        double p3=16*A2*Q4-4*(1+A4)*Q2;
        // These are the coefficients of the depressed cubic in M.
        double m3=p6;
        double m1=p4-3*p6;
        double m0=p3;
        // Now we solve the depressed cubic using a numerically
        // stable method. This can be optimised at the expense
        // of stability.
// Solving Cubic:
        double c1=-m1/(3.0*m3);
        double c2=0.5*m0/m3;
        double det=c2*c2-c1*c1*c1;
        if (det<0)
        {
```

```
                double th=acos(c2*pow(c1,-1.5));
                double sq=-2*sqrt(c1);

                M=sq*cos(th/3);
                if (M*M>4)      {
                        M*=0.5;
                        L=M+sqrt(M*M-1);
                        if (L>1) k=sqrt(L);
                }       // These stages retrieve k from M.

                M=sq*cos((th+k2PI)/3);
                if (M*M>4)      {
                        M*=0.5;
                        L=M+sqrt(M*M-1);
                        if (L>1) k=sqrt(L);
                }

                M=sq*cos((th-k2PI)/3);
                if (M*M>4)      {
                        M*=0.5;
                        L=M+sqrt(M*M-1);
                        if (L>1) k=sqrt(L);}
                }
        }
        else
        {
                double t1=pow(fabs(c2)+sqrt(det),1.0/3.0), t2=0;
                if (c2>0) t1*=-1;
                if (t1!=0) t2=c1/t1;
                M=t1+t2;
                if (M*M>4)      {
                        M*=0.5;
                        L=M+sqrt(M*M-1);
                        if (L>1) k=sqrt(L);}
                }
        }

        // We now have k.
        k=1+k_scale*(k-1);
        double k2=k*k;
        // Scale up frequency of first H(s)
        out[1]*=k,out[4]*=k;out[2]=out[5]=k2;
        // Now invert k.
        k=1.0/k;        k2=1.0/k2;
        // And scale down the second H(s)
        out[7]*=k,out[10]*=k;out[8]=out[11]=k2;
}
```