Fast Modulation of Filter Parameters

A practical guide

Ammar Muqaddas
ammar@solostuff.net

April 20, 2019

Version 1.1.1

Acknowledgment

I wish to extend my greatest gratitude to members of the DSP forum in KVRAudio and to a few more people for their help. This work could not have been mature without the useful feedback either recently or over the years from such professionals in the field. Some of which I know their names and some others I've known only by their forum nick name. People including, Vadim Zavalishin, matt42, Andrew Simper, mystran, Jakub Horak, Urs Heckmann and a few others that I might have missed.

Introduction

If you're here you probably know what I'm talking about. So you've tried the famous RBJ cookbook filters, or you've designed filters using pole-zero analyses. All good, but whenever you try to modulate the cutoff at audio rate of any of these filters, it explodes!! Causing a numerical overflow. You recheck your coefficients, stability and all that. You check your code. All seams to be good. And yet, no good.

After long hours, you admit to your self and give up on the math. Now you try to just clip the output of the filter internally and hope some thing musical would come out when it blows. And your ears are barged with those untamable noisy resonations beeping sporadically all over the spectrum. It's just baaaad. Ugly, ugly as hell. Hell would run away if it heard that.

In this article I try to shed some light over the subject of fast modulation of filter parameters. Why do supposedly stable filters explode? And what to do about it. With a bit of experimentation, specially in the first section.

The next three sections describe three different methods for designing filters that can withstand fast modulation without exploding. The fourth section is complementary providing a proof for the stability of the CPS method described in section 2.

Prerequisites

You are expected to know at least basic DSP and filter design concepts to understand this article. At least you should have used filter code before. You know what's a pole and what's a zero. You know whats a Z-plane and an S-Plane. You know what's a transfer function.

1. Zero Delay Feedback (ZDF) filters

Probably the definitive guide for the design of ZDF filters is the book named "The Art of VA Filter Design".

ZDF is actually the popular common name. A more accurate name would be to name it after the methodology defined in the book, which is called Topology Preserving Transform (TPT). But more on that latter.

So lets pinpoint some of the usually said advantages of ZDF (or TPT) filters and what I think really matters the most.

1. **Cut-off and resonance are independent**. True, but good none ZDF filters can do that too. Try RBJ cookbook. For other filters, the claimed increase in resonance that happens usually near high frequencies is caused by the two poles bellow and above nyquist becoming too close together. One can manage that simply by placing zeros at nyquist. Actually some ZDF filters them selves have zeros at nyquist.
2. **Analog Sounding**. Sure enough, ZDF filters are based on idealized analog topologies. Well, It's not accurate component modeling, but still. However, the thing is, as a sound designer or musician, one may like to have that as an option not as a must. It's arguably a flavor, not better or worse.
3. **Stable fast modulation of filter cutoff and resonance.** Yes, this is where I think ZDF really shines. It is an indispensable feature. Many other filters (when done the classical way), just can't cope with fast modulation. We will further show how to fix this in those filters too.

Ok, let me first brief you on the gist of what the above book and this ZDF business is all about.

1. Analog filters use integrators to achieve the filtering. Those integrators are electrical components (mainly capacitors) that integrate the incoming signal. Literally, integrate like in math. So you would pass say a square wave through it and out goes a triangle!!. How cool is that.

2. Connecting those integrators with feedback loops in various ways you can achieve a very wide variety of useful filters.

3. We can digitize these analog filters by replacing those analog integrators with digital equivalents. There are various methods to do so. The book (mentioned above) suggests using some thing called Trapezoidal Integrators.

4. The time domain function for this trapezoidal integrator takes the following form:

$$g = \tan(\pi \cdot w_c) \quad \textbf{(1)}$$

$$out[t] = g \cdot in[t] + m[t-1] \quad \textbf{(2)}$$

$$m[t] = g \cdot in[t] + out[t] \quad \textbf{(3)}$$

where ***m*** is a state (memory) variable that lives through the previous clock tick. ***Out*** and ***in*** are obvious. w_c is $\frac{f_c}{f_s}$ (cutoff frequency over sampling rate) where w_c<0.5. ***g*** is a gain amount that is directly proportional to the w_c frequency chosen.

5. The integrator above approximates what happens with analog integrators. The **tan()** is a fix actually for a warping (squeeze) that happens to the frequency spectrum. Without it, if you shoot for a frequency **fc** the result will be fc minus

something. So it's basically a correction that is required because the integrator above squeezes the whole analog frequency spectrum from 0Hz to ∞Hz and puts it between the digital spectrum 0Hz to nyquist (0 to half the sampling rate). As g=tan(…), One may think of g as a frequency amount. You might actually see g referred to as w_c , w or f in some references or books. It's confusing some times so watch it.

6. Take the output and memory equations directly from the analog topology. The book calls this method a Topology Preserving Transform (TPT). (I'm going to use the TPT name instead of ZDF from now on). Have a look at the following state variable filter topology.

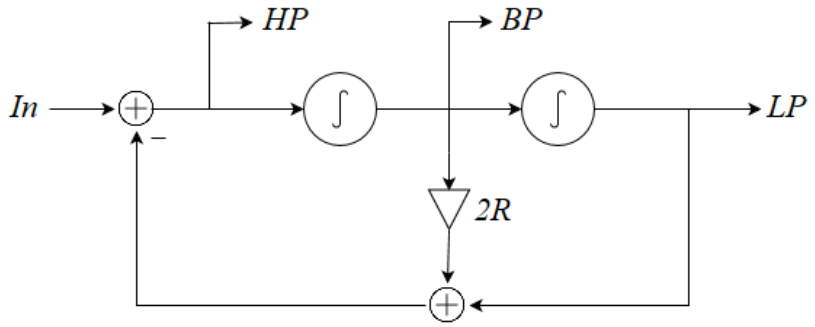

Figure 1

Were R is 1−resonance while resonance is between 0 to 1. In-place of the integrators shown, you put the trapezoidal integrator formula (2) shown above. Then derive LP, HP and BP output directly from the topology. (Circle with a plus is a sum and triangle is a multiply). Doing this you'll get the following:

$$hp[t] = in[t] - (2R \cdot bp[t] + lp[t]) \quad \textbf{(4)}$$

$$bp[t] = g \cdot hp[t] + m1[t-1] \quad \textbf{(5)}$$

$$lp[t] = g \cdot bp[t] + m2[t-1] \quad \textbf{(6)}$$

m1 and ***m2*** here are the state (memory) variables for the two integrators. Both were calculated during the previous clock tick. After using them, we need to calculate their values to be used for the next clock tick. We get those from (3) above.

$$m1[t] = g \cdot hp[t] + bp[t] \quad \textbf{(7)}$$

$$m2[t] = g \cdot bp[t] + lp[t] \quad \textbf{(8)}$$

We're done, but there is a slight problem. Notice in (4) (5) and (6). HP, LP and BP are all expressed in terms of each other (i.e. implicitly). This is the result of the Zero Delay Feedback problem (Yes it's a problem, not actually a name for the filter type. But the public seams to have adopted that as a name). It happens because we have no memory elements in the feedback path between LP and the input (check the topology).

But actually solving this problem is easy. We need to solve the three equations (4) (5) and (6) to evaluate HP in terms of *m1*, *m2*, *R* and *g* ONLY. Once we've done that, we can use (5) to get BP and then (6) to get LP (use your favorite symbolic math tool,

Mathematica, SageMath etc…). After doing all that and summoning everything, you'll get this final working code.

```
g = tan(PI * wc);
hp = (in-(g + 2*R)*m1 - m2)/(g*g + 2*g*R + 1);
bp = g*hp + m1;
lp = g*bp + m2;
m1 = g*hp + bp;
m2 = g*bp + lp;
```

I'm sure you can optimize this a bit further. This filter can handle very fast audio rate modulation of the cutoff (and resonance as far as I can test). If you came here only for this, I don't blame you.

Exposing the guts

All good and dandy. Now here is the thing. All we did above we did in the time domain. Replacing the analog integrators with digital approximations. What if we wanted to do the same thing but in the frequency Z domain instead. That is, figure out the transfer function of the topology in the analog frequency S domain then digitize it to the Z domain. How would that go? Do you think the result would also stand up for fast modulation? It should right? Lets check that. (Experts, Thinking "no, no, no it has to be time invariant". Ok just hold on for moment).

Lets find the transfer function of the topology. An analog integrator in the frequency S domain has the transfer function $\frac{g}{s}$. It has a pole at frequency 0Hz and a zero at ∞Hz (you'd usually see this as $s=j0$ and $s=j\infty$. Weird notation? I know. It's just that frequencies are expressed in the imaginary axis in the S-plane. The Z-plane is different, (don't confuse).

Deriving the HP, BP and LP from the topology will give us:

$$HP = in - 2R \cdot BP - LP \tag{9}$$

$$BP = \frac{g}{s} HP \tag{10}$$

$$LP = \frac{g}{s} BP \tag{11}$$

Eliminate HP and BP and solve for $\frac{LP}{in}$ and you'll get the transfer function of LP.

$$H(s) = \frac{LP}{in} = \frac{g^2}{s^2 + 2R \cdot g \cdot s + g^2} \tag{12}$$

We could also do the same for HP and BP, But lets just stick with LP. Now we'll digitize it. By moving it from the S domain to the Z domain. Doing this, we'll have to approximate the $\frac{g}{s}$ integrator with the Z domain transform of the same trapezoidal

integrator above (1)(2)(3). Without too much detail, we do this by doing the following substitution:

$$s = 2\frac{1-z^{-1}}{1+z^{-1}} \tag{13}$$

This has a name. It's called the Bilinear Transform (Or Tusten in MatLab). Make no mistake about it. It is a direct result of the trapezoidal integrator above. So after the substitution we have:

$$H(z) = \frac{LP}{in} = \frac{g^2}{\left(2\frac{1-z^{-1}}{1+z^{-1}}\right)^2 + 2R \cdot g \cdot 2\frac{1-z^{-1}}{1+z^{-1}} + g^2} \tag{14}$$

Putting it in polynomials forms:

$$H(z) = \frac{LP}{in} = \frac{g^2 \cdot \left(1 + 2z^{-1} + z^{-2}\right)}{4 + 4R \cdot g + g^2 + \left(2g^2 - 8\right) \cdot z^{-1} + \left(4 - 4R \cdot g + g^2\right) \cdot z^{-2}} \tag{15}$$

Multiplying by the denominator and transforming this to the time domain (Inverse Z transform). LPz^{-2} tranforms to *lp[t-2]*. LPz^{-1} tranforms to *lp[t-1]* and LPz^{0} to *lp[t]*. $in \cdot z^{-2}$ to *in[t-2]*, $in \cdot z^{-1}$ to *in[t-1]* and $in \cdot z^{-0}$ to *in[t]*.

$$lp[t] = \frac{g^2 \cdot (in[t] + 2 \cdot in[t-1] + in[t-2]) - \left(2g^2 - 8\right) \cdot lp[t-1] - \left(4 - 4R \cdot g + g^2\right) \cdot lp[t-2]}{4 + 4R \cdot g + g^2} \tag{16}$$

Notice here that this is essentially direct form 1 except that we factored $\frac{1}{4 + 4R \cdot g + g^2}$ out, Direct form 1 is presumably the most numerically stable form of all the well known forms. (If you don't know what I'm talking about, just continue to see where I'm heading with this. The big picture is more important than the details here).

Now we'll put that in code.

```
/* in0 is the current input sample (i.e in[t]).
lp0 is the current output sample (i.e lp[t])
in1, in2, lp1, lp2 are state (memory) variables. (i.e in[t-1],in[t-
2], lp[t-1], lp[t-2]
*/
g = tan(PI * wc);
temp = 4+4*R*g+g*g;
lp0 = (g*g*(in0+2*in1+in2) - (2*g*g-8)*lp1 - (4-4*R*g+g*g)*lp2)/temp;
in2 = in1; in1 = in0;
lp2 = lp1; lp1 = lp0;
```

Try it, it should work. But now try it with fast audio rate modulation. And……It Explodes!! Surprised? I'll be surprised if your not. This code came from the transfer function of the exact same topology that worked before using the exact same integrator. Just done in a different method. Yet one of them is stable the other is not at high rate modulation. How is that possible?! What do you think went wrong here?

The reason the above didn't work is because the transfer function that we derived does not apply here because the system becomes a Time Variant system when modulation is applied; therefore the derivation above is not valid. A Transfer function only works when we have a Linear Time Invariant (LTI) system. A modulated filter is not LTI.

Lets try a different way. What if we do it in the time domain? Completely avoiding the transfer function. In other words, I want to get some thing like this:

$$LP[t] = b_0 in[t] + b_1 in[t-1] + b_2 in[t-2] - a_1 LP[t-1] - a_2 LP[t-2] \quad \textbf{(17)}$$

That is, get the LP output only in terms of current and past input and output, just like Direct form 1. But do it all in the time domain. Is this even possible? Lets get back to our original 5 equations we got straight from the topology (4)(5)(6)(7)(8). I'm going to write the exact same equations three times, one exactly as above, one for the previous time sample and one before it. (i.e. [t], [t-1] and [t-2]). BUT I'm going to also account for changes in cut-off and resonance across the 3 time/samples (i.e. g is going to be g[t], g[t-1], g[t-2] and R is going to be R[t], R[t-1], R[t-2])

$$hp[t] = in[t] - \left(2R[t] \cdot bp[t] + lp[t]\right) \quad \textbf{(18)}$$
$$bp[t] = g[t] \cdot hp[t] + m1[t-1] \quad \textbf{(19)}$$
$$lp[t] = g[t] \cdot bp[t] + m2[t-1] \quad \textbf{(20)}$$
$$m1[t] = g[t] \cdot hp[t] + bp[t] \quad \textbf{(21)}$$
$$m2[t] = g[t] \cdot bp[t] + lp[t] \quad \textbf{(22)}$$

$$hp[t-1] = in[t-1] - \left(2R[t-1] \cdot bp[t-1] + lp[t-1]\right) \quad \textbf{(23)}$$
$$bp[t-1] = g[t-1] \cdot hp[t-1] + m1[t-2] \quad \textbf{(24)}$$
$$lp[t-1] = g[t-1] \cdot bp[t-1] + m2[t-2] \quad \textbf{(25)}$$
$$m1[t-1] = g[t-1] \cdot hp[t-1] + bp[t-1] \quad \textbf{(26)}$$
$$m2[t-1] = g[t-1] \cdot bp[t-1] + lp[t-1] \quad \textbf{(27)}$$

$$hp[t-2] = in[t-2] - \left(2R[t-2] \cdot bp[t-2] + lp[t-2]\right) \quad \textbf{(28)}$$
$$bp[t-2] = g[t-2] \cdot hp[t-2] + m1[t-3] \quad \textbf{(29)}$$
$$lp[t-2] = g[t-2] \cdot bp[t-2] + m2[t-3] \quad \textbf{(30)}$$
$$m1[t-2] = g[t-2] \cdot hp[t-2] + bp[t-2] \quad \textbf{(31)}$$
$$m2[t-2] = g[t-2] \cdot bp[t-2] + lp[t-2] \quad \textbf{(32)}$$

These are 15 equations. Use your favorite symbolic solver to get lp[t] in terms of past lp[], and past and current in[], g[] and R[] eliminating all other terms. You'll get the following hideous thing:

$$lp[t] = -(g[t]^2 g[t-1] in[t] + g[t] g[t-1]^2 in[t-1] - (2g[t] g[t-1] R[t-1] - (2g[t] R[t] + 1) g[t-1] - g[t]) g[t-2]^2 in[t-2] - 2g[t] g[t-1] lp[t-1] R[t-1] + (g[t]^2 in[t] + ((2g[t] R[t] + 1) g[t-1]^2 + 2g[t] g[t-1]) in[t-1] - ((2g[t] R[t] + 1) g[t-1]^2 + 2g[t] g[t-1] - 2g[t] R[t] - 1) lp[t-1] + 2(g[t]^2 g[t-1] in[t] + (2g[t] R[t] + 1) g[t-1] lp[t-1]) R[t-1]) g[t-2] - (g[t] g[t-1]^2 - 2(2g[t] R[t] + 1) g[t-1] - g[t]) lp[t-1] + ((2g[t] g[t-1] R[t-1] - (2g[t] R[t] + 1) g[t-1] - g[t]) g[t-2]^2 + 2g[t] g[t-1] R[t-1] - (2g[t] R[t] + 1) g[t-1] - g[t]) lp[t-2] - 2((2g[t] g[t-1] R[t-1] - (2g[t] R[t] + 1) g[t-1] - g[t]) g[t-2] lp[t-2] + (g[t]^2 g[t-1] in[t] + g[t] g[t-1]^2 in[t-1] - 2g[t] g[t-1] lp[t-1] R[t-1] - (g[t] g[t-1]^2 - 2(2g[t] R[t] + 1) g[t-1] - g[t]) lp[t-1]) g[t-2]) R[t-2]) / (2(g[t]^2 + 2g[t] R[t] + 1) g[t-1] g[t-2] R[t-2] - (g[t]^2 + 2g[t] R[t] + 1) g[t-1] - (2(g[t]^2 + 2g[t] R[t] + 1) g[t-1] R[t-1] + g[t]^2 + 2g[t] R[t] + 1) g[t-2])$$

This is the correct time variant formula that the transfer function method missed. And it actually works at fast rate modulation (well it should). Obviously, no one in his sanity would use such a CPU hog for any thing other than experimentation. But if you insist on trying it for your self here is the code. Just bear in mind here that because of the numerical complexity, this can cause 0/0 division which will cause a NAN to occur.

```
// Initialize to avoid NAN
g1 = 0.5; g2 = 0.5; R1 = 0.5; R2 = 0.5;

// One sample Tick (below)
g = tan(PI * wc);
lp0 = -(g*g*g1*in0 + g*g1*g1*in1 - (2*g*g1*R1 - (2*g*R + 1)*g1 -
g)*g2*g2*in2 - 2*g*g1*lp1*R1 + (g*g*in0 + ((2*g*R + 1)*g1*g1 +
2*g*g1)*in1 - ((2*g*R + 1)*g1*g1 + 2*g*g1 - 2*g*R - 1)*lp1 +
2*(g*g*g1*in0 + (2*g*R + 1)*g1*lp1)*R1)*g2 - (g*g1*g1 - 2*(2*g*R +
1)*g1 - g)*lp1 + ((2*g*g1*R1 - (2*g*R + 1)*g1 - g)*g2*g2 + 2*g*g1*R1
- (2*g*R + 1)*g1 - g)*lp2 - 2*((2*g*g1*R1 - (2*g*R + 1)*g1 -
g)*g2*lp2 + (g*g*g1*in0 + g*g1*g1*in1 - 2*g*g1*lp1*R1 - (g*g1*g1 -
2*(2*g*R + 1)*g1 - g)*lp1)*g2)*R2)/(2*(g*g + 2*g*R + 1)*g1*g2*R2 -
(g*g + 2*g*R + 1)*g1 - (2*(g*g + 2*g*R + 1)*g1*R1 + g*g + 2*g*R +
1)*g2);
if (_isnan(lp0)) lp0=1; // NANs can occur
g2 = g1; g1 = g;
R2 = R1; R1 = R;
in2 = in1; in1 = in0;
lp2 = lp1; lp1 = lp0;
```

Despite it's uselessness. The above huge thing verifies a few points.

1. Stability at fast rate modulation has nothing to do with the filter structure or form used being zero delay feedback. There is an obvious one sample and two sample delay feedback (lp[t-1] and lp[t-2]) above. Yet it's stable at fast rate modulation.
2. TPT is sufficient but not necessary to achieve stability in fast modulation. The complex form above is obviously nothing similar to the original analog structure, although it was derived from it. The integrators and their state/memory variables have been completely removed. Yet it's stable.

3. The really nice thing about the TPT (or ZDF) method is how it completely embeds all past cutoff and resonance (g and R) information in the memory variables (m1 and m2 in our case). Needing nothing but the current g and R for the current time/sample calculation.

Another advantage of TPT filters is that there is a wealth of history behind analog filters that one could bring directly into the digital form without needing to reinvent allot of wheels.

2. Arbitrary filter design using Pole-Zero analysis

The main advantage of Pole-Zero filter design is the clear and direct ability to place every pole and zero exactly where you want in the frequency spectrum. Doing so, you can achieve practically any bespoke filter or phase shifter without relying on analog structures.

In Pole-Zero analyses you start by placing whatever poles and zeros you want on the Z-Plane. Then derive the transfer function from there, and then transform to time domain and so get your code. Any instability problems we have with fast modulation or without modulation are due to the poles not zeros. Without modulation, stability can be insured by simply keeping all poles inside the unit circle. But with modulation, it becomes more complicated. So lets see the simplest 2-pole no-zero filter we can have. (I can do 1-pole but these will have no resonance parameter. It's not very useful for typical usage).

The Z-plane plot and frequency response of a 2-pole no-zero filter is shown below. X shown in the Z-plane denotes the two conjugate pair poles.

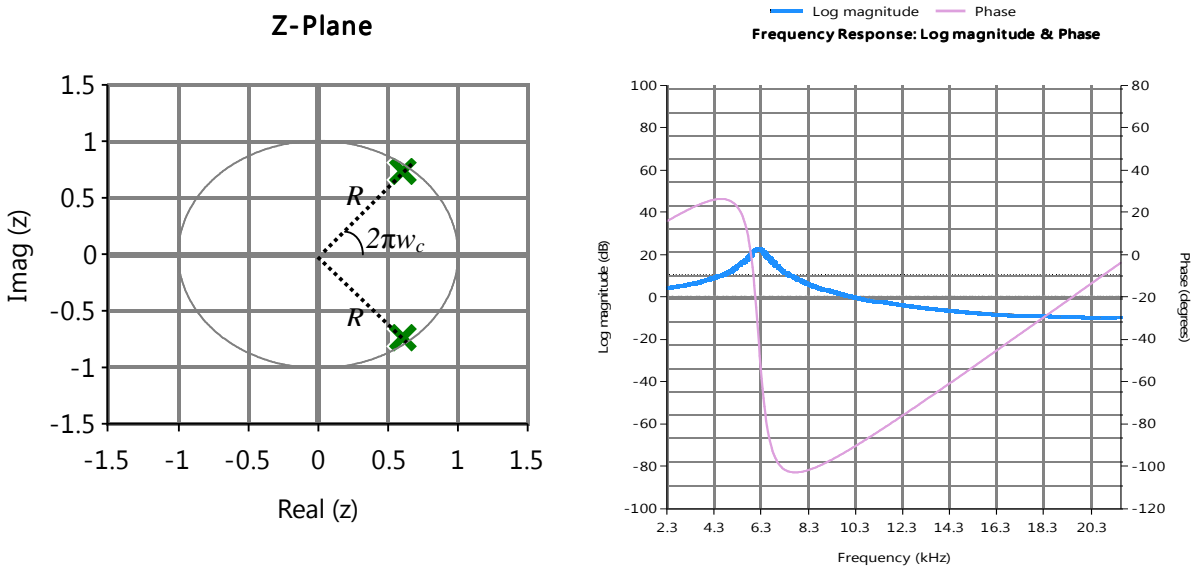

Figure 2

Transfer function:

$$H(z) = \frac{1}{\left(1 - R(x - jy)\cdot z^{-1}\right)} \cdot \frac{1}{\left(1 - R(x + jy)\cdot z^{-1}\right)} \quad \textbf{(33)}$$

Which can be simplified as:

$$H(z) = \frac{1}{1 - 2R\cdot x\cdot z^{-1} + R^2 z^{-2}} \quad \textbf{(34)}$$

R is the resonance or more accurately, the radius in this case (distance from origin to the pole in the Z-plane. See figure.). $x = \cos(2\pi \cdot w_c)$ and $y = \sin(2\pi \cdot w_c)$ where w_c is as defined above. i.e. $\frac{f_c}{f_s}$ (cutoff frequency over sampling rate). I'm going to assume that you know what I'm talking about here.

The above is very raw, it's not normalized nor adapted well for good sound. But it's best for simplicity. Getting the time domain formula (using the inverse Z-transform as we did before).

$$out[t] = in[t] + 2R\cdot x\cdot out[t-1] - R^2 out[t-2] \quad \textbf{(35)}$$

As you would expect if you'd use this filter while modulating the cutoff very fast it'll explode. For the same reason we talked about above. The transfer function is NOT valid while modulating the filter parameters because the system becomes time variant. In otherwords, the formula above should involve current and past cutoff and resonance information (at least for t-1 and t-2). So what can we do about this?

Complex 1-Pole Sections (CPS) Stabilizer

What if we managed to put the time domain system in the following form:

$$out[t] = c_1 \cdot in[t] + \sum_{k=2}^{n} c_k \cdot m_k[t-k+1] \quad \textbf{(36)}$$

Such that all c_k are coefficients expressed only in terms of current cutoff and resonance (R[t], x[t] and y[t]). And all m_k are state/memory variables that embed the complete previous state of the filter including it's past cutoff and resonance information. Similarly:

$$m_i[t] = d_{i,1} \cdot in[t] + \sum_{k=2}^{n} d_{i,k} \cdot m_k[t-k+1] \quad \textbf{(37)}$$

Where all $d_{i,k}$ are coefficients, again expressed only in terms of current cutoff and resonance (R[t], x[t] and y[t]). If we can achieve this, we have solved our time variant system.

As far as I can see, it's a design problem that can have many solutions. (If your head has started to spin. Never mind. The above is just a generalization/formularization of the problem we have). Now one may argue that because this is a second order system, we only need two state/memory variables. Well, yes but that is only guaranteed if we have no restriction on the coefficients (i.e. being expressed only in current cutoff and resonance). Which is not the case here. Hence, we have to extend until *n* number of state variables. Where *n* is something that we need to find out.

There is actually an easy way to resolve the problem. ~~Although I can't see it mentioned or proved anywhere~~[1]. **By splitting the transfer function into it's original cascade of Complex 1-Pole Sections (CPS)**. Exposing it's conjugate pairs. In other words, we'll use (33) instead of (34). Here it is again:

$$H(z) = \frac{1}{\left(1 - R(x - jy)\cdot z^{-1}\right)} \cdot \frac{1}{\left(1 - R(x + jy)\cdot z^{-1}\right)}$$

Getting the time domain form of the first (left) cascade:

$$re_1[t] = in[t] + R(x \cdot re_1[t-1] - y \cdot im_1[t-1]) \quad \textbf{(38)}$$

$$im_1[t] = R(x \cdot im_1[t-1] + y \cdot re_1[t-1]) \quad \textbf{(39)}$$

Where $re_1[t]$ is the real part and $im_1[t]$ is the imaginary part. Feeding it into the second (right) section.

$$re_2[t] = re_1[t] + R(x \cdot re_2[t-1] + y \cdot im_2[t-1]) \quad \textbf{(40)}$$

$$im_2[t] = im_1[t] + R(x \cdot im_2[t-1] - y \cdot re_2[t-1]) \quad \textbf{(41)}$$

But we know already that the imaginary part of the second section MUST be equal to zero since the conjugate pair cancel each other imaginary parts. This leaves us with:

$$re_2[t] = re_1[t] + R(x \cdot re_2[t-1]) \quad \textbf{(42)}$$

$$im_2[t] = 0 \quad \textbf{(43)}$$

The output of the whole system is in $re_2[t]$. So basically we can just write:

$$out[t] = re_1[t] + R(x \cdot out[t-1]) \quad \textbf{(44)}$$

With some manipulation one can easily show that the above form can be expressed in the generalized form proposed in (36) and (37).

Final 2-pole no-zero filter code:

```
// in0 is the current input (in[t]), out0 is the current output
```

[1] Fortunately. it seams that a similar method has actually been suggested and used before by Max Mathews, in his paper "Methods for Synthesizing Very High Q Parametrically Well Behaved Two Pole Filters". Albeit being named differently as "Phasor Filters". A brief discussion of stability can be found in the paper called "Dynamic FM synthesis using a network of complex resonator filters". Named here as "Complex Resonators". At the time of writing this article, I had no knowledge of those papers.

```
// re, im and out0 are state/memory variables.

x = cos(2.0*PI*wc); y = sin(2.0*PI*wc); /* Or we can use: y=sqrt(1.0-
x*x); same thing */
// Section 1
temp = re;
re = in0 + R*(x*re - y*im);
im = R*(x*im + y*temp);
// Section 2
out0 = re + R*(x*out0);
```

This is stable at fast modulation as far as my tests go. I do provide an elaborate proof for its stability in the last section of this article.

Try it, but don't expect good sound. For a properly designed say low pass filter we need some more work which is shown in the next section.

Using the CPS method described above one can stabilize practically any n-pole m-zero filter by cascading multiple CPS sections. Notice that in the special case of strictly real poles, this would be just a special case of the CPS where the imaginary part is y=0.

Low pass 2-pole 2-zero filter

Note: it is very common among designers to omit specifying the number of zeros in a filter. I just specify it here for extra clarity. So don't be confused when you see many filters out there saying like 2-pole or 4-pole. They also usually have zeros. But that's not always true of-course. Which may add to the confusion.

I'm going to briefly describe an LP design that's originally NOT stable at fast modulation then make it stable by applying the simple CPS method I showed above. Notice here that in both TPT and pole-zero you can get high audible gain if the resonance is very high in fast modulation. But they won't explode. Experimenting with soft clipping can be interesting here. In fact, clipping is inherent in analog filters because each power supply has limits to the voltage is can produce.

So here it goes.

1. Looking at the two pole transfer function in (34). We'll use that and add two zeros at nyquist. Why? Because you won't have a low pass without them. You need something to attenuate higher frequencies. This will give you the following:

$$H(z) = \frac{1 + 2z^{-1} + z^{-2}}{1 - 2R \cdot x \cdot z^{-1} + R^2 z^{-2}} \tag{45}$$

2. We'll normalize (force) the gain at 0Hz (z=1) to be 0db. Why? Because if we don't, then the fundamental (lowest) frequency of your input may get amplified or attenuated. You don't want a lowpass to do that right?

$$b_0 = \left(\frac{1 - 2R_c \cdot x + R_c^{\,2}}{4} \right) \tag{46}$$

$$H(z) = b_0 \cdot \frac{1 + 2z^{-1} + z^{-2}}{1 - 2R \cdot x \cdot z^{-1} + R^2 z^{-2}} \tag{47}$$

3. The filter above works but still won't completely close at low cutoff and resonance. I'm going to steal something from RBJ to redefine the resonance. And with some intuition we can remap R to get a better response:

$$c = 100R^3 + 1 \tag{48}$$

$$R_c = \frac{c}{c + y} \tag{49}$$

Where *y* is as defined above. And the final transfer function is:

$$H(z) = b_0 \cdot \frac{1 + 2z^{-1} + z^{-2}}{1 - 2R_c \cdot x \cdot z^{-1} + R_c^{\,2} z^{-2}} \tag{50}$$

In time domain:

$$out[t] = b_0 \left(in[t] + 2in[t-1] + in[t-2] \right) + 2R_c x \cdot out[t-1] - R_c^{\,2} out[t-2] \tag{51}$$

This would work well except at fast modulation. So we'll do as above and split $H(z)$ into a cascade of complex 1-pole sections. Notice that the order of the numerator (zeros) doesn't matter here since those do not cause instability. We can simply put all zeros in the first section:

$$H(z) = b_0 \cdot \frac{1 + 2z^{-1} + z^{-2}}{\left(1 - R(x - jy) \cdot z^{-1}\right)} \cdot \frac{1}{\left(1 - R(x + jy) \cdot z^{-1}\right)} \tag{52}$$

In time domain:

$$re_1[t] = b_0 \left(in[t] + 2in[t-1] + in[t-2] \right) + R_c \left(x \cdot re_1[t-1] - y \cdot im_1[t-1] \right) \tag{53}$$

$$im_1[t] = R_c \left(x \cdot im_1[t-1] + y \cdot re_1[t-1] \right) \tag{54}$$

$$lp[t] = re_1[t] + R_c \left(x \cdot lp[t-1] \right) \tag{55}$$

And from there we get our final code for the 2-pole 2-zero LP filter.

```
// in0, in1, in2 are in[t] in[t-1] and in[t-2] respectively
// lp is the current output lp[t]

x = cos(2.0*PI*wc); y = sin(2.0*PI*wc);
c = 100*R*R*R+1; Rc = c/(c+y);
b0 = (1.0-2*Rc*x+Rc*Rc)/4.0;
temp = re;
re = b0*(in0 + 2*in1 + in2) + Rc*(x*re - y*im);
```

```
im = Rc*(x*im + y*temp);
lp = re + Rc*(x*lp);
in2 = in1; in1 = in0;
```

This is not optimized for CPU usage but rather for clear illustration.

Multimode 2-pole 2-zero filter

Although the main point has been made above. I feel I'd just put a final code of a multimode filter. The LP is almost same as above, while we derive the HP and BP along the way. Notice here that I put the zeros after the second cascade for easier HP and BP derivation. But (for some reason) this results in higher gain at fast modulation (it's still stable).

```
temp = 2.0*PI*wc;
x = cos(temp); y = sin(temp);
c = 100*R*R*R+1; Rc = c/(c+y);
b0 = (1.0-2*Rc*x+Rc*Rc)/4.0;
temp = re;
re = in0 + Rc*(x*re - y*im);
im = Rc*(x*im + y*temp);
int0 = re + Rc*(x*int0);
lp = b0*(int0 + 2*int1 + int2);
hp = lp - in0;
bp = y*(int0 - int2);

int2 = int1; int1 = int0;
```

Arbitrary n-pole m-zero filters

Using a 2-pole 2-zero filter building block. We can stabilize practically any filter of any higher or lower order we like. Suppose we have the following generalized form transfer function.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{56}$$

Assuming that the coefficients a_1 and a_2 do not cause any poles to go outside the unit circle (i.e stable in the time in-variant case). All we need to know to stabilize this using CPS is the roots of the denominator (when equalized to zero). Since this is a quadratic equation. The roots are:

$$p_1 = \frac{-a_1 + \sqrt{a_1^2 - 4a_2}}{2} \quad p_2 = \frac{-a_1 - \sqrt{a_1^2 - 4a_2}}{2}$$

Therefore $H(z)$ can be rewritten as:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{\left(1 - p_1 z^{-1}\right)} \cdot \frac{1}{\left(1 - p_2 z^{-1}\right)}$$

For a 2-pole filter ($a_2 \neq 0$), p_1 and p_2 may or may not be a complex conjugate pair. The final code is:

```
tau = (a1*a1-4*a2);
If (tau>=0)
{     // Two real roots. No imaginary part
      sqtau = sqrt(tau);
      // Section 1
      x = (-a1 + sqtau)/2.0;
      re = b0*in0 + b1*in1 + b2*in2 + x*re;
      // Section 2
      x = (-a1 - sqtau)/2.0;
      out0 = re + x*out0;
}
else
{     // With imaginary parts (Conjugate Pair)
      sqtau = sqrt(-tau);
      x = -a1/2.0; y = sqtau/2.0;
      // Section 1
      temp = re;
      re = b0*in0 + b1*in1 + b2*in2 + x*re - y*im;
      im = x*im + y*temp;
      // Section 2
      out0 = re + x*out0;
}
in2 = in1; in1 = in0;
```

If you know your poles are always real or always complex, you can disregard one part of the **if** statement above.

For a 1-real-pole filter ($a_2 = 0$). The code is simply:

```
// 1 real pole filter
out0 = b0*in0 + b1*in1 + b2*in2 - a1*out0;
in2 = in1; in1 = in0;
```

Notice here that if your design intentionally chooses p_1 and p_2 to be directly controlled by your filter parameters without needing the a_1, a_2 coefficients. Then the code above simplifies because x,y are already known from the real,imaginary parts of p_1 and p_2 without solving for any roots.

A few obvious but rather important notes here. You can stabilize 1-poles or 1-zero filters by zeroing out the a_2 or the b_2 coefficients respectively. And you can stabilize no-poles or no-zeros filters by zeroing out the a_1, a_2 or the b_1 , b_2 coefficients respectively. (Although the no-poles case here is probably useless since such filters would be always stable anyway).You can also stabilize higher order n-pole m-zero filters by cascading multiple CPS filters using this method (i.e. placing them in series).

3. Pole-Zero to TPT Converter

If your not comfortable using complex numbers in the CPS method but still like to do pole-zero design. There is a rather indirect way to convert any Z-domain pole-zero filter to a TPT filter and essentially stabilizing it in fast modulation. Although this

method is a bit lengthy, the most interesting thing about it is that it yields different sounds in fast modulation!!. In fact very different. The sounds you get from TPT are particularly smoother while CPS generates more aggressive and hard tones.

The method is as follows:

1. Apply **inverse** Bilinear Transform to the Z domain transfer function you have to bring to the analog S domain.
2. Equalize the resultant transfer function coefficients with an already known stable analog topology that has the same (or more) number of poles and zeros. Granted we already have digitized the topology (using Trapezoidal Integration) and got the code for it. This would undo the **inverse** Bilinear Transform we did in step one so the filter would stay the same when digitized.
3. Solve the equations you got to get the Z domain to TPT substitutions.

I'm going to do this in a general purpose form for a 2-pole 2-zero transfer function to serve as a building block for higher and lower order filters. Suppose we have the following Z-domain transfer function.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{57}$$

If your wondering why there isn't an a_0 coefficient, it's because you can always eliminate one of the coefficients dividing the numerator and denominator over it. So no need to add it from the beginning.

So first we're going to apply inverse Bilinear Transform. Recall from equation (13). We need to invert that to get z^{-1}. Doing that we get:

$$z^{-1} = \frac{2 - s}{2 + s} \tag{58}$$

Substituting in (57) to get it in the analog S domain:

$$H(z) = \frac{b_0 + b_1\left(\frac{2-s}{2+s}\right) + b_2\left(\frac{2-s}{2+s}\right)^2}{1 + a_1\left(\frac{2-s}{2+s}\right) + a_2\left(\frac{2-s}{2+s}\right)^2} \tag{59}$$

Simplifying and rewriting it in polynomials form:

$$H(s) = \frac{(b_0 - b_1 + b_2)s^2 + 4(b_0 - b_2)s + 4(b_0 + b_1 + b_2)}{(1 - a_1 + a_2)s^2 + 4(1 - a_2)s + 4(1 + a_1 + a_2)} \tag{60}$$

I'm going to divide both stories over $(1 - a_1 + a_2)$ to eliminate s^2 coefficient in the denominator. We'll call it τ. You'll see why in a moment.

$$\tau = (1 - a_1 + a_2) \tag{61}$$

$$H(s)=\frac{\frac{1}{\tau}(b_0-b_1+b_2)s^2+\frac{4}{\tau}(b_0-b_2)s+\frac{4}{\tau}(b_0+b_1+b_2)}{s^2+\frac{4}{\tau}(1-a_2)s+\frac{4}{\tau}(1+a_1+a_2)} \quad \textbf{(62)}$$

Now step 2. We are going to equalize the coefficients here to the coefficients of a TPT structure of the same order that we know is stable. We can actually use the same topology we used in Figure 1. The transfer functions of *HP, BP* and *LP* happens to be the following:

$$HP(s)=\frac{s^2}{s^2+2Rgs+g^2} \quad \textbf{(63)}$$

$$BP(s)=\frac{gs}{s^2+2Rgs+g^2} \quad \textbf{(64)}$$

$$LP(s)=\frac{g^2}{s^2+2Rgs+g^2} \quad \textbf{(65)}$$

We know (from LTI linearity) that when we sum transfer functions together, the result is simply the mix (sum in time domain) of those three filters. So what if we sum all three while scaling them to our liking. So we can define a new transfer function *G(s)*

$$G(s)=c_0HP(s)+c_1BP(s)+c_2LP(s) \quad \textbf{(66)}$$

$$G(s)=c_0\frac{s^2}{s^2+2Rgs+g^2}+c_1\frac{gs}{s^2+2Rgs+g^2}+c_2\frac{g^2}{s^2+2Rgs+g^2} \quad \textbf{(67)}$$

Therefore:

$$G(s)=\frac{c_0s^2+c_1gs+c_2g^2}{s^2+2Rgs+g^2} \quad \textbf{(68)}$$

(A very similar approach to (66) (67) and (68) can be found in the book section 4.7)

We want *G(s) = H(s)*. Ding so, we have to equalize all coefficients in (62) with coefficients in (68) and solve for g, R, c_0, c_1 and c_2. Without too much detail, the final substitution we get:

From denominator:

$$\tau=(1-a_1+a_2),\quad g=\pm\sqrt{\frac{4}{\tau}(1+a_1+a_2)},\quad R=\frac{2}{g\tau}(1-a_2)$$

From numerator:

$$c_0 = \frac{1}{\tau}(b_0 - b_1 + b_2), \quad c_1 = \frac{4}{g\tau}(b_0 - b_2), \quad c_2 = \frac{4}{g^2\tau}(b_0 + b_1 + b_2)$$

Notice here that there is actually no need to do the frequency pre-warp (tan() thing) as before. Because what ever warping that has been done with the time domain trapezoidal integration, would have already been undone with the inverse Bilinear Transform in the frequency domain.

A few obvious but rather important notes here. You can do 1-poles or 1-zero filters by zeroing out the a_2 or the b_2 coefficients respectively. And you can do no-poles or no-zeros filters by zeroing out the a_1, a_2 or the b_1 , b_2 coefficients respectively. (Although the no-poles case here is probably useless since such filters would be always stable anyway).You can also do higher order n-pole m-zero filters by cascading multiple filters using this stabilizer method.

A few things to watch for. Your coefficients should not cause τ to equal to zero otherwise you'd have a division over zero. Also your coefficients should not cause what's under the square root in $g = \pm\sqrt{\frac{4}{\tau}(1 + a_1 + a_2)}$ to be negative, unless you specifically want to deal with imaginary numbers for a certain reason.

Final code:

```
/* in, out: Input and Output
a1, a2, b0, b1, b2 are the coefficients of your Z-domain transfer
function that you want to stabilize using TPT
m1, m2 are state/memory variables
All other variables are just temporary storage
*/

tau = 1-a1+a2;
g = sqrt((4/tau)*(1+a1+a2));
R = 2*(1-a2)/(g*tau);
c0 = (b0-b1+b2)/tau;
c1 = 4*(b0-b2)/(g*tau);
c2 = 4*(b0+b1+b2)/(g*g*tau);

g = 0.5*g;
hp = (in-(g + 2*R)*m1 - m2)/(g*g + 2*g*R + 1);
bp = g*hp + m1;
lp = g*bp + m2;
m1 = g*hp + bp;
m2 = g*bp + lp;
out = c0*hp + c1*bp + c2*lp;
```

I'll leave CPU optimization for you.

4. CPS BIBO Stability

Here we present a proof for the stability of the Complex 1-Pole Section system under fast modulation of filter parameters.

Requirements

A signal is bounded if a finite $B \geq 0$ exists such that the signal magnitude never exceeds B. In otherwords:

$$\left|y[t]\right| \leq B \quad \forall t \tag{69}$$

We have to show that the discrete time variant CPS is BIBO stable (Bounded Input Bounded Output stable). A sufficient condition for this is to prove that every bounded input produces a bounded output given the system has zero initial conditions (or at least bounded initial conditions).

We know that:

$$\left|y[t]\right| \leq \sum_{k=-\infty}^{\infty} \left|y[k]\right| \tag{70}$$

If we prove the summation to be bounded, we have proved y[t] to be bounded. It so happens that this is easier to prove than directly proving boundedness of y[t] itself. Essentially, We have to show that:

$$\sum_{k=-\infty}^{\infty} \left|y[k]\right| \leq B < \infty \tag{71}$$

In other words, we have to prove that y[k] is **absolutely sumable**. Note: If this was an LTI system. Then it would have been a bit easier. We only had to prove that the impulse response is absolutely sumable. But this is not the case here.

Definitions and assumptions

The input signal is x[t] and output is y[t]. xre[t], xim[t] are the real and imaginary parts of x[t] respectively. re[t], im[t] are the real an imaginary parts of y[t] respectively

The output of the CPS is defined as:

$$re[t] = xre[t] + R[t] \cdot \left(\cos(w[t]) \cdot re[t-1] - \sin(w[t]) \cdot im[t-1]\right) \tag{72}$$

$$im[t] = xim[t] + R[t] \cdot \left(\cos(w[t]) \cdot im[t-1] + \sin(w[t]) \cdot re[t-1]\right) \tag{73}$$

R[t] is the time-variant radius (resonance). Assuming $0 \leq R[t] < 1$. And w(t) is the time-variant angle (cutoff). We also define $h[t]$ as the **scaled** impulse response and define $hre[t]$ and $him[t]$ as the real and imaginary parts of $h[t]$ respectively. To get the scaled impulse response, we set the input x[t] to an arbitrary complex scaled impulse as follows:

$$x[t] = (r + jm)\delta[t] = \begin{cases} (r + jm) & t = 0 \\ 0 & t \neq 0 \end{cases} \tag{74}$$

In other words:

$$xre[t] = r\delta[t] = \begin{cases} r & t = 0 \\ 0 & t \neq 0 \end{cases} \qquad xim[t] = m\delta[t] = \begin{cases} m & t = 0 \\ 0 & t \neq 0 \end{cases}$$

From (72) and (73) we can see that the only necessary initial conditions to calculate $hre[0]$ and $him[0]$ (and hence $h[0]$) are $hre[-1]$ and $him[-1]$. Therefore, we assume the system has finite arbitrary initial conditions at $hre[-1]$ and $him[-1]$. For $t < -1$ we assume the system has zero initial conditions (or at least bounded initial conditions).

For $t > 0$. From (72) and (73):

$$hre[t] = R[t] \cdot (\cos(w[t]) \cdot hre[t-1] - \sin(w[t]) \cdot him[t-1]) \tag{75}$$

$$him[t] = R[t] \cdot (\cos(w[t]) \cdot him[t-1] + \sin(w[t]) \cdot hre[t-1]) \tag{76}$$

Proof

The magnitude of the scaled impulse response $h[t]$ is

$$|h[t]| = |hre[t] + j \cdot him[t]| \tag{77}$$

$$|h[t]| = \sqrt{hre[t]^2 + him[t]^2} \tag{78}$$

Expanding:

$$|h[t]| = \sqrt{(R[t] \cdot (\cos(w[t]) \cdot hre[t-1] - \sin(w[t]) \cdot him[t-1]))^2 + (R[t] \cdot (\cos(w[t]) \cdot him[t-1] + \sin(w[t]) \cdot hre[t-1]))^2}$$

Simplifying:

$$|h[t]| = \sqrt{R[t]^2 \left(hre[t-1]^2 (\cos(w[t])^2 + \sin(w[t])^2) + him[t-1]^2 (\cos(w[t])^2 + \sin(w[t])^2)\right)}$$

Since $\cos(w[t])^2 + \sin(w[t])^2 = 1$

$$|h[t]| = R[t]\sqrt{hre[t-1]^2 + him[t-1]^2} \tag{79}$$

Side note: the scaled impulse response magnitude is independent of w(t) (cutoff) changes

Notice, that the square root term on the right is essentially the magnitude of $h[t-1]$. Check (78). So we can rewrite (79) as:

$$\left|h[t]\right| = R[t] \cdot \left|h[t-1]\right| \tag{80}$$

We can expand this back to h[0].

$$\left|h[t]\right| = \left(\prod_{i=1}^{t} R[i]\right) \cdot \left|h[0]\right| \tag{81}$$

It is easy to see from here that:

$$\lim_{t\to\infty} \prod_{i=1}^{t} R[i] = 0 \qquad 0 \le R[i] < 1 \tag{82}$$

But for the sake of clarity, we define R_m to be the maximum of R[t] for all t. So $\prod_{i=1}^{t} R[i] \le R_m{}^{t}$. Therefore, from (81)

$$\left|h[t]\right| = \left|h[0]\right| \cdot \left(\prod_{i=1}^{t} R[i]\right) \le \left|h[0]\right| R_m{}^{t} \tag{83}$$

From here, we know that:

$$\lim_{t\to\infty} \left(\left|h[0]\right| R_m{}^{t}\right) = 0 \qquad 0 \le R_m < 1 \tag{84}$$

Therefore, using (83) and (84):

$$\lim_{t\to\infty} \left|h[t]\right| = 0 \tag{85}$$

And since $\left|h[0]\right|$ is finite because it is a function of the finite initial condition $h[-1]$ from (80). So using (83), $h[t]$ will also be finite for all $t > 0$ since $R_m{}^{t}$ is finite. Notice that $h[t]$ has also been assumed zero (or bounded) for all $t < -1$. These three conclusions along with (85) essentially guarantee that $\left|h[t]\right|$ is sumable. Which means that there exist B such that

$$\sum_{k=-\infty}^{\infty} \left|h[k]\right| \le B < \infty \tag{86}$$

This proves that the output signal $h[t]$ is bounded. Which proves that the output signal $y[t]$ is bounded when the input is a scaled impulse $(r + jm)\delta[t]$ assuming the CPS system has finite initial conditions at $y[-1]$ and zero (or bounded) initial conditions for $t < -1$.

Now, any input $x[t]$ can be expressed as a sum of scaled impulses as follows:

$$x[t] = \sum_{k=-\infty}^{\infty} x[k] \cdot \delta[t-k] \tag{87}$$

For every *t*. A scaled impulse of $x[t] \cdot \delta[0]$ is input while the CPS system has an initial condition of $y[t-1]$. In other words, $x[t]$ is nothing but repeated differently scaled impulses. We've already showed that the response of the CPS system for such impulses is bounded. Which means that $y[t]$ is bounded and it is used as an initial condition for $y[t+1]$ which is bounded as well. And is in turn used as initial condition for $y[t+2]$ and so on until $y[t+\infty]$. Therefore, there exist B such that

$$\left|y[t]\right| \leq \sum_{k=-\infty}^{\infty} \left|y[k]\right| \leq B < \infty \qquad \forall t \tag{88}$$

Which is what we wanted to prove.