

# *IIR filter design for audio DSP programmers*

IT'S FOR PROGRAMMERS, NOT MATH STUDENTS/TEACHER

© 2013 MAIK MENZ

## Contents

1. Introduction .....	1
2. Infinite impulse response (IIR) filter.....	1
2.1 Sample processing .....	1
3. Design process .....	2
3.1 Filter response .....	2
3.2 z-Plane.....	2
3.2.1 Poles and Zeros .....	3
3.2.2 Frequencies .....	3
3.3 Placing poles and zeros .....	4
3.4 From complex poles and zeros to real coefficients .....	4
3.4.1 From Complex to real.....	5
3.5 Filter normalization.....	6
3.5.1 Insert the scale factor .....	7
3.5.2 Calculating the scale factor .....	8
3.5.3 Simplified scale factor for common filter types.....	9
4. Higher order filter .....	10
4.1 Cascade and parallel filter for programmers .....	11

## 1. Introduction

Filter design has always been a mystery for me, before I finally found my way into it. It looks very complicated and involves a lot of higher math that I already forgot after learning it the hard way 15 years ago in school. The intention of this document is to show that IIR filter design isn't black magic and I'll try to describe the important steps as detailed as possible.

As you might have noticed, English is not my native language. I haven't studied the stuff in this document, I just learned it on my own. Be patient!

## 2. Infinite impulse response (IIR) filter

As you might know, there are two different types of filters: Finite impulse response (FIR) filters and Infinite impulse response (IIR) filters. The difference between those two is, that when you give a FIR filter an input of a fixed length, then the output has a fixed length, too. The output of an IIR filter can be infinitely long. The advantage of an IIR filter is, that you can get very good results in less processing steps, than with a FIR filter. That's the main reason for using it in audio DSP.

This document only covers IIR filters, but most of the described functions and calculation steps also apply to FIR.

### 2.1 Sample processing

The sample processing in an IIR filter is always done by one and the same formula, but it can be modified to fit your needs, e.g. by changing the number of coefficients. In math, this formula is called "Difference Equation" and the name tells nothing about what it actually does. Here it is:

$$y[n] = \sum_{k=0}^M b_k x[n-k] - \sum_{k=1}^N a_k y[n-k]$$

This already looks badass? Yes! But when you unroll it, you might get a clue, what it means:

$$y[n] = (b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \dots) - (a_1 y[n-1] + a_2 y[n-2] + a_3 y[n-3] + \dots)$$

In this equation **x** is the array of input samples, **y** is the array of output samples, **n** is the number of the current sample that's getting processed. The variables **a** and **b** are called coefficients, where **b** is the coefficient for input samples and **a** is the coefficient for previous output samples. In some text you'll notice that **a** and **b** are interchanged, there's nothing wrong, but you have to take care.

In the filter design process, the only thing you do is calculating those coefficients, to pass it into the equation above. For real time processing, where you might not have an output array, you can replace **y[n]** by your output variable, but you have to store all previous outputs for processing the next output sample. As you can see from the equation, the length of the array of output samples, you need to store, depends on the amount of coefficients you use for your filter. To keep it simple, in the further text I'll only use 3 input coefficients ( **b0**, **b1**, **b2** ) and 2 output coefficients ( **a1**, **a2** ). This is a very common limitation, the resulting filter equation is called a biquad.

The complete biquad equation is this:

$$y[n] = (b_0x[n] + b_1x[n - 1] + b_2x[n - 2]) - (a_1y[n - 1] + a_2y[n - 2])$$

Or simplified:

$$y[n] = b_0x[n] + b_1x[n - 1] + b_2x[n - 2] - a_1y[n - 1] - a_2y[n - 2]$$

For real time processing, this function looks like this in code:

```
y = b0*x + b1*x1 + b2*x2 - a1*y1 - a2*y2
y2 = y1
y1 = y
x2 = x1
x1 = x
```

In this code **x** is the current input and **y** is the current output. Be careful, when you put this into a function, the variables **y1**, **y2**, **x1** and **x2** are reassigned for the next run, so you can't put them into the function scope, unless you declare them static.

### 3. Design process

OK, now you know how to calculate the samples, but where do we get the coefficients **a** and **b** from? Well, you can search in the internet, and test some precalculated ones, but this is not the goal of this document.

To calculate those coefficients yourself, you need a desired filter response that you can represent in a very simple way, with the least amount data.

#### 3.1 Filter response

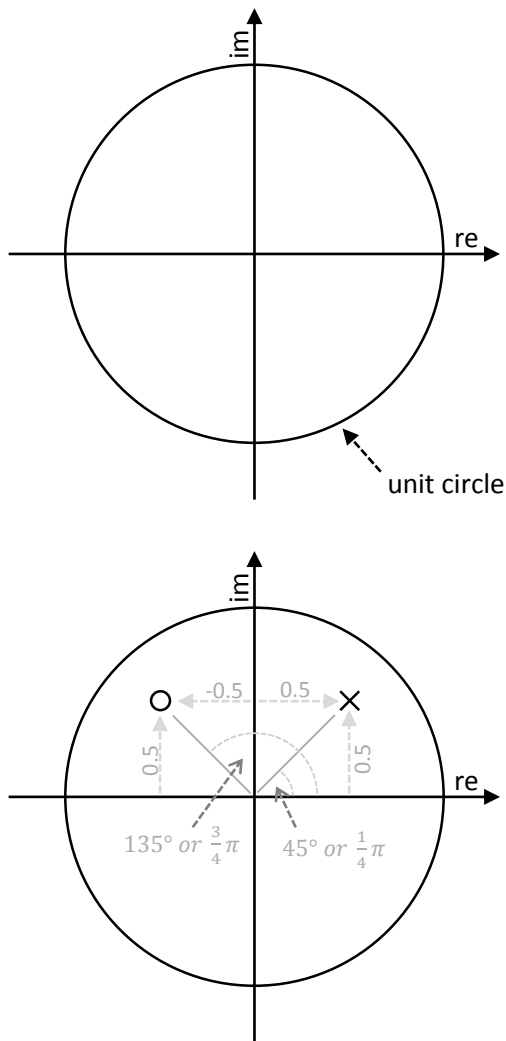
The filter response tells, what the filter does with a specific input frequency. The main properties of the filter response are magnitude and phase. The magnitude response tells us how much the input frequency gets amplified or attenuated by the filter. The phase response of the filter tells us, how the phase of the input signal gets changed by the filter for a specific frequency.

There are also other properties like impulse response, step response or group delay, but these go beyond this documentation, and for audio DSP they are not that important. These additional properties can be evaluated by either deriving it from the phase/magnitude response or applying an input signal to the filter itself.

#### 3.2 z-Plane

The z-Plane is a graphical representation of a filter response. You basically draw in your desired filter response into a diagram, and use some basic measurements to get the data for your calculations.

The z-Plane looks like this:



You might think, it's a diagram for complex numbers with a circle drawn into it. And you are almost right! But the circle has a special role.

The radius of the circle is 1, therefore it is called the "unit circle". Everything lying on the circle is 1 unit away from the center.

### 3.2.1 Poles and Zeros

There are two things that you can draw into the diagram: Poles and Zeros. Poles are represented as crosses and zeros are represented as circles. Poles amplify frequencies and zeros attenuate.

In this diagram you see one pole and one zero. The zero is at the coordinates **-0.5** on the real axis and **0.5** on the imaginary axis, or in short **[-0.5, 0.5]**. Its complex value is **-0.5+i0.5**.

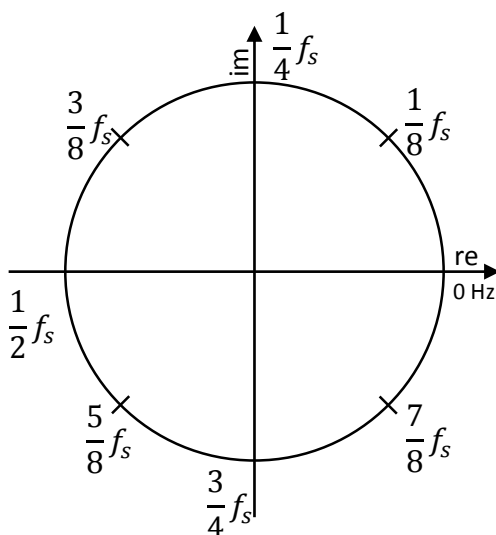
The zero has a magnitude, which is just the distance to the origin of the coordinates. In DSP related documents you'll often find the Greek letters, and I don't want to make a difference. The Greek letter for the magnitude is the small Sigma  $\sigma$  and you can calculate it with the rearranged Pythagorean Theorem:

$$\sigma = \sqrt{real^2 + imag^2}$$

In the example above, the magnitude of the zero is **0.7071067811865475**.

The angle at which the zero is placed is **135°** or  **$\frac{3}{4}\pi$**  in radians. As you may notice, the angle is measured in relation to the real axis. The Greek letter for the angle is the small Omega:  $\omega$  And just for completeness, you can calculate  $\omega$  this way:

$$\omega = \text{atan2}(\text{imag}, \text{real})$$



### 3.2.2 Frequencies

In the z-Plane, angles are the most important things, because they represent frequencies. Wrapped around the unit circle you have all your frequencies. You can convert angles into frequency and vice versa:

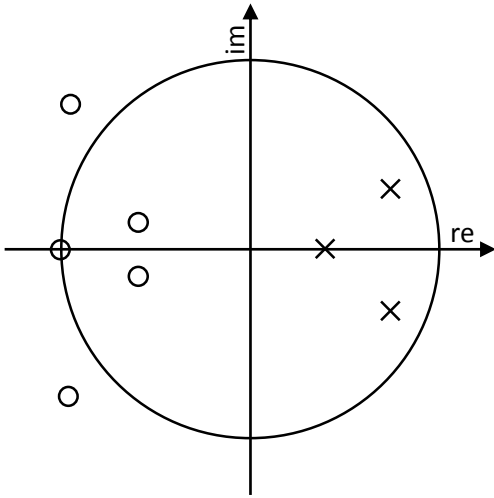
$$f = \frac{\omega}{2\pi} f_s \quad \omega = 2\pi \frac{f}{f_s}$$

$f_s$  is your sampling frequency.

Side note: The top half of the z-Plane covers all frequencies below the Nyquist frequency.

### 3.3 Placing poles and zeros

Poles have to be placed inside the unit circle. When you place them outside the unit circle or even on it, the filter can get unstable. Zeros can get placed everywhere.



The distances of poles and zeros to the unit circle have influence to their effects. The closer a pole is to the unit circle, the more the nearby frequencies on the unit circle get amplified. The closer a zero is to the unit circle, the more the nearby frequencies on the unit circle get attenuated.

When you place poles and zeros on the z-Plane, you have to do it in pairs. These pairs of poles and zeros are called conjugate pairs. So when you place something in the top half of the z-Plane, then you have to place the same object with an inverted imaginary coordinate to the bottom half, too. When you place something on the real axis (where the imaginary coordinate is 0), you don't have to add a second object at the same position.

This conjugate pair placement is not a general rule, it is just because we require it, to keep the rest of the design process as simple as possible.

### 3.4 From complex poles and zeros to real coefficients

After you placed your poles and zeros on the z-Plane, you somehow have to convert them into the real coefficients **a** and **b**. This can be done with another function which is called "transfer function", this is how it is often presented:

$$H(z) = \frac{B}{A} = \frac{\sum_{k=0}^M b_k z^{-k}}{1 + \sum_{k=1}^N a_k z^{-k}}$$

Basically, this function gives the filter response in complex numbers, where **z** is the frequency on the unit circle as a complex number. You don't really have to calculate with this, for filter design. You just need to know how it looks like. You can unroll the sums, to make it more readable. For a biquad filter it looks like this:

$$H(z) = \frac{b_0 z^0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad \text{or simplified:} \quad H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

There is also another representation for the transfer function, which looks like this:

$$H(z) = \frac{\prod_{k=1}^M (z - z_k)}{\prod_{k=1}^N (z - p_k)}$$

In this version you have  $z_k$  which are the complex coordinates of your zeros in the z-Plane and  $p_k$  which are the complex coordinates of your poles in the z-Plane. A biquad has 2 poles and 2 zeros maximum, therefore the unrolled version looks like this:

$$H(z) = \frac{(z - z_1)(z - z_2)}{(z - p_1)(z - p_2)}$$

OK, now you have one formula in two representations. One has all your known poles and zeros in it, the other has all your unknown coefficients in it. The next step is clear: let's morph one formula into the layout of the other.

Let's start by expanding the pole-zero version:

$$H(z) = \frac{z^2 - z_1z - z_2z + z_1z_2}{z^2 - p_1z - p_2z + p_1p_2}$$

Next step: partially factor out  $z$  and  $p$

$$H(z) = \frac{z^2 + (-z_1 - z_2)z + z_1z_2}{z^2 + (-p_1 - p_2)z + p_1p_2}$$

This already looks very close to the coefficient representation. The exponents of  $z$  are not negative, but this can be changed by dividing the whole equation by the highest power of  $z$ , in this case  $z^2$ .

$$H(z) = \frac{1 + (-z_1 - z_2)z^{-1} + z_1z_2z^{-2}}{1 + (-p_1 - p_2)z^{-1} + p_1p_2z^{-2}}$$

Now let's compare this version, with the coefficient representation:

$$H(z) = \frac{1 + (-z_1 - z_2)z^{-1} + z_1z_2z^{-2}}{1 + (-p_1 - p_2)z^{-1} + p_1p_2z^{-2}} \quad H(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2}}{1 + a_1z^{-1} + a_2z^{-2}}$$

They look identical. You can read from this representation directly what the filter coefficients are:

$$\begin{aligned} b_0 &= 1 \\ b_1 &= -z_1 - z_2 \\ b_2 &= z_1z_2 \\ a_1 &= -p_1 - p_2 \\ a_2 &= p_1p_2 \end{aligned}$$

### 3.4.1 From Complex to real

But there is still one problem, the poles and zeros are complex numbers while the  $a$  and  $b$  coefficients are real numbers. This is where the conjugate pair placement rule comes in very handy. Let's look closely at  $b_1$ :

$$\begin{aligned} b_1 &= -z_1 - z_2 \\ b_1 &= -(z_{1re} + iz_{1im}) - (z_{2re} + iz_{2im}) \\ b_1 &= -z_{1re} - iz_{1im} - z_{2re} - iz_{2im} \\ b_1 &= (-z_{1re} - z_{2re}) - i(z_{1im} + z_{2im}) \end{aligned}$$

When we have only one zero, then it is on the real axis, and there is no imaginary part, and the other zero coordinates are  $0$ , so  $b_1$  is a real number. When we have two zeros, there are two chances, either both are on the real axis or they are a conjugate pair. When they are on the real axis, then there is no imaginary part. When they are conjugate pairs then  $z_{2im} = -z_{1im}$  and  $z_{2re} = z_{1re}$  as you can see, the imaginary part cancels out, and you end up with  $b_1 = -2z_{1re}$ . Now we take a look at  $b_2$ :

$$\begin{aligned} b_2 &= z_1z_2 \\ b_2 &= (z_{1re} + iz_{1im})(z_{2re} + iz_{2im}) \\ b_2 &= z_{1re}z_{2re} + z_{1re}iz_{2im} + z_{2re}iz_{1im} + i^2z_{1im}z_{2im} \end{aligned}$$

Here is one rule for complex numbers that you should already know:  $i^2 = -1$ . With this and the factoring out of  $i$  we finally get:

$$b_2 = (z_{1re}z_{2re} - z_{1im}z_{2im}) + i(z_{1re}z_{2im} + z_{1im}z_{2re})$$

For one zero, the summand in the imaginary parts get 0 and the whole imaginary part, too. The same happens with two zeros on the real axis. When we have a conjugate pair of zeros, then again  $z_{2im} = -z_{1im}$  and  $z_{2re} = z_{1re}$ . With this in the equation we get:

$$b_2 = (z_{1re}z_{1re} - z_{1im}(-z_{1im})) + i(z_{1re}(-z_{1im}) + z_{1im}z_{1re})$$

$$b_2 = (z_{1re}z_{1re} + z_{1im}z_{1im}) + i(-z_{1re}z_{1im} + z_{1im}z_{1re})$$

$$b_2 = z_{1re}z_{1re} + z_{1im}z_{1im}$$

As you can see, the imaginary part is gone, too. Since we don't have to care about the imaginary part, we can write out coefficients like this:

$$b_0 = 1$$

$$b_1 = -z_{1re} - z_{2re}$$

$$b_2 = z_{1re}z_{2re} - z_{1im}z_{2im}$$

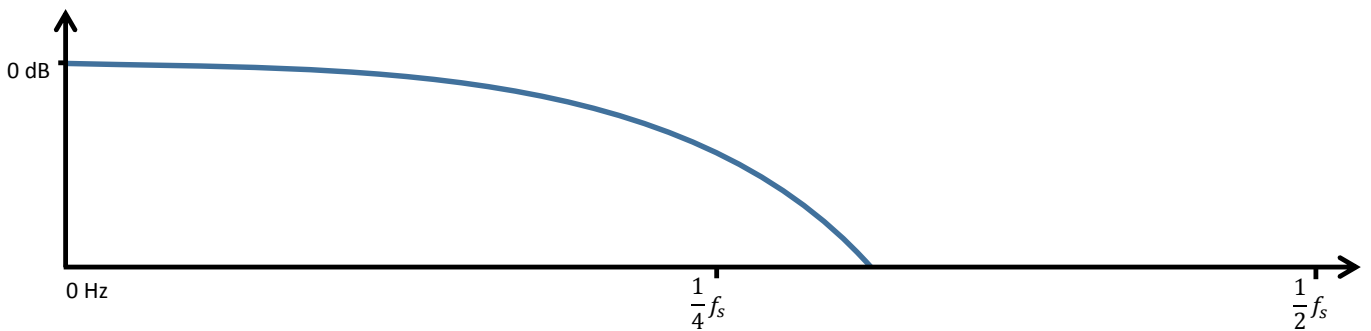
$$a_1 = -p_{1re} - p_{2re}$$

$$a_2 = p_{1re}p_{2re} - p_{1im}p_{2im}$$

Now you might ask, why do we need  $b_0$  when it is always 1? The answer is: normalization. Right now we took the poles and zeros from the z-Plane, as they are. But there is one thing that the z-Plane doesn't show and that is the gain. The z-Plane tells you what frequencies get amplified and what frequencies get attenuated. But you have to decide what your reference level is.

### 3.5 Filter normalization

Right now you have converted a relative pole/zero filter response into coefficients. In the normalization process, you make the relative response absolute. To do this you need a reference point, from which you know the magnitude of your filter response. This is in most cases fairly simple. Let's say you design a Lowpass filter, with a magnitude response in a shape like this:



In this case you can already see your reference level: it is **0dB** at **0Hz**, because **0Hz** is the lowest frequency and in a Lowpass filter, this will always pass through as the name suggests it. For a Highpass filter it is similar, then your reference level is **0db** at  $\frac{1}{2}f_s$  because this is the highest frequency in your signal. In a Bandpass filter it is a little bit different, because your **0dB** level is not at a fixed point, but you can determine that, too. It is just in the center of your passband.



So back to the Lowpass filter, we know the magnitude of our filter is **0dB** at **0Hz**, so let's write it down:

$$|H(z_0)| = 0dB = 1$$

Or in general:

$$|H(z_r)| = 1$$

In this case  $z_r$  is our reference frequency as complex number, where our filter response needs to be **0dB**. Mostly your complex reference is on the unit circle ( $\sigma = 1$ ), this implies that the magnitude of your reference input frequency is **0dB**.

### 3.5.1 Insert the scale factor

All we have to do now is, to scale our filter response to fit this requirement. As you might recall,  $H(z)$  for our biquad in pole/zero representation was this:

$$H(z) = \frac{1 + (-z_1 - z_2)z^{-1} + z_1z_2z^{-2}}{1 + (-p_1 - p_2)z^{-1} + p_1p_2z^{-2}}$$

And there are no variables that we could scale. So we need to introduce a scaling variable, let's call it  $c$ :

$$H(z) = c \frac{1 + (-z_1 - z_2)z^{-1} + z_1z_2z^{-2}}{1 + (-p_1 - p_2)z^{-1} + p_1p_2z^{-2}}$$

And now we add our requirements:

$$1 = \left| c \frac{1 + (-z_1 - z_2)z_r^{-1} + z_1z_2z_r^{-2}}{1 + (-p_1 - p_2)z_r^{-1} + p_1p_2z_r^{-2}} \right| = c \left| \frac{1 + (-z_1 - z_2)z_r^{-1} + z_1z_2z_r^{-2}}{1 + (-p_1 - p_2)z_r^{-1} + p_1p_2z_r^{-2}} \right|$$

When we rearrange that, we get  $c$ :

$$c = \frac{1}{\left| \frac{1 + (-z_1 - z_2)z_r^{-1} + z_1z_2z_r^{-2}}{1 + (-p_1 - p_2)z_r^{-1} + p_1p_2z_r^{-2}} \right|}$$

The final step now is to insert  $c$  into our pole/zero representation, and make it look like the coefficient representation:

$$H(z) = c \frac{1 + (-z_1 - z_2)z^{-1} + z_1z_2z^{-2}}{1 + (-p_1 - p_2)z^{-1} + p_1p_2z^{-2}}$$

$$H(z) = \frac{c + (-z_1 - z_2)cz^{-1} + z_1z_2cz^{-2}}{1 + (-p_1 - p_2)z^{-1} + p_1p_2z^{-2}}$$

Inserting  $c$  completely looks very complicated, and it doesn't really help. When you read out all of our coefficients the final results you get are:

$$b_0 = c$$

$$b_1 = (-z_{1re} - z_{2re})c$$

$$b_2 = (z_{1re}z_{2re} - z_{1im}z_{2im})c$$

$$a_1 = -p_{1re} - p_{2re}$$

$$a_2 = p_{1re}p_{2re} - p_{1im}p_{2im}$$

### 3.5.2 Calculating the scale factor

OK, for someone who knows how to deal with complex numbers it wouldn't be a problem to calculate the variable  $c$ . For everyone else, this is a nearly impossible task. There are several ways to calculate this, they depend on how you represent complex numbers. For my explanation, I use the polar form and some of the properties of our poles and zeros that we already know. The polar form of a complex number looks like:

$$z = \sigma(\cos\omega + i \sin\omega)$$

As mentioned earlier  $\sigma$  is the magnitude, which is the distance to the origin of the coordinates. And  $\omega$  is the angle in relation to the real axis.

As you know, our frequencies are on the unit circle, so for our frequency variable  $z_r$ , we already know that  $\sigma = 1$ . So we got:

$$z_r = \cos\omega_r + i \sin\omega_r$$

With  $\omega_r$  as our reference frequency expressed as angle.

From the calculation of our coefficients we know that they are all real numbers. I have added a  $u$  to the index of our coefficients to show that these are the unscaled coefficients:

$$b_{u_0} = 1$$

$$b_{u_1} = -z_1 - z_2 = -z_{1re} - z_{2re}$$

$$b_{u_2} = z_1 z_2 = z_{1re} z_{2re} - z_{1im} z_{2im}$$

$$a_{u_1} = -p_1 - p_2 = -p_{1re} - p_{2re}$$

$$a_{u_2} = p_1 p_2 = p_{1re} p_{2re} - p_{1im} p_{2im}$$

We can insert this into the formula of  $c$  and get:

$$c = \frac{1}{\left| \frac{1 + b_{u_1} z_r^{-1} + b_{u_2} z_r^{-2}}{1 + a_{u_1} z_r^{-1} + a_{u_2} z_r^{-2}} \right|}$$

Now, to deal with the exponents of  $z_r$ , we can use another rule for complex numbers that says:

$$z^n = r^n (\cos(n\omega) + i \sin(n\omega))$$

Applied to  $z_r$ , we get it even shorter, because  $r$  is 1 in this case:

$$z_r^n = \cos(n\omega_r) + i \sin(n\omega_r)$$

Let's insert this into  $c$ :

$$c = \frac{1}{\left| \frac{1 + b_{u_1} (\cos(-\omega_r) + i \sin(-\omega_r)) + b_{u_2} (\cos(-2\omega_r) + i \sin(-2\omega_r))}{1 + a_{u_1} (\cos(-\omega_r) + i \sin(-\omega_r)) + a_{u_2} (\cos(-2\omega_r) + i \sin(-2\omega_r))} \right|}$$

We can simplify that by using symmetry of trigonometric functions, that is:

$$\sin(-\omega) = -\sin\omega \quad \text{and} \quad \cos(-\omega) = \cos\omega$$

Applied to  $c$  we get:

$$c = \frac{1}{\left| \frac{1 + b_{u_1} (\cos\omega_r - i \sin\omega_r) + b_{u_2} (\cos(2\omega_r) - i \sin(2\omega_r))}{1 + a_{u_1} (\cos\omega_r - i \sin\omega_r) + a_{u_2} (\cos(2\omega_r) - i \sin(2\omega_r))} \right|}$$

This needs a little rearrangement, so first we factor out the brackets:

$$c = \frac{1}{\left| \frac{1 + b_{u1}\cos\omega_r - ib_{u1}\sin(\omega_r) + b_{u2}\cos(2\omega_r) - ib_{u2}\sin(2\omega_r)}{1 + a_{u1}\cos\omega_r - ia_{u1}\sin(\omega_r) + a_{u2}\cos(2\omega_r) - ia_{u2}\sin(2\omega_r)} \right|}$$

Next step is collecting the real and the imaginary parts:

$$c = \frac{1}{\left| \frac{1 + b_{u1}\cos\omega_r + b_{u2}\cos(2\omega_r) - i(b_{u1}\sin\omega_r + b_{u2}\sin(2\omega_r))}{1 + a_{u1}\cos\omega_r + a_{u2}\cos(2\omega_r) - i(a_{u1}\sin\omega_r + a_{u2}\sin(2\omega_r))} \right|}$$

When you look closer now, you see, that there is only one complex number with  $b_u$  coefficients and one complex number with  $a_u$  coefficients. This makes the next step a lot easier, which is getting the magnitude of the main denominator. As mentioned before, the magnitude of a complex number can be calculated with this:

$$\sigma = \sqrt{\text{real}^2 + \text{imag}^2}$$

Our  $c$  can be written as:

$$c = \frac{1}{\sqrt{\frac{\text{real}_1^2 + \text{imag}_1^2}{\text{real}_2^2 + \text{imag}_2^2}}}$$

With the root identities, we can transform that into:

$$c = \frac{1}{\frac{\sqrt{\text{real}_1^2 + \text{imag}_1^2}}{\sqrt{\text{real}_2^2 + \text{imag}_2^2}}} \quad \text{or simplified:} \quad c = \frac{\sqrt{\text{real}_2^2 + \text{imag}_2^2}}{\sqrt{\text{real}_1^2 + \text{imag}_1^2}} \quad \text{or even better:} \quad c = \sqrt{\frac{\text{real}_2^2 + \text{imag}_2^2}{\text{real}_1^2 + \text{imag}_1^2}}$$

When we apply this to our formula of  $c$ , we get our final result that is an absolute real value calculated from real numbers and an angle:

$$c = \sqrt{\frac{\left(1 + a_{u1}\cos\omega_r + a_{u2}\cos(2\omega_r)\right)^2 + \left(a_{u1}\sin\omega_r + a_{u2}\sin(2\omega_r)\right)^2}{\left(1 + b_{u1}\cos\omega_r + b_{u2}\cos(2\omega_r)\right)^2 + \left(b_{u1}\sin\omega_r + b_{u2}\sin(2\omega_r)\right)^2}}$$

I skip the step of inserting  $c$  into our biquad coefficients, because this would make it unnecessary difficult to read. In a program you just calculate the unscaled coefficients first, then you calculate  $c$  and after that you can calculate the scaled coefficients.

### 3.5.3 Simplified scale factor for common filter types

For some filter types, the scale factor  $c$  can be simplified a lot, because of the reference frequency that is commonly used. For a Lowpass the reference frequency is  $0\text{Hz}$ , so we get:

$$\omega_r = \omega_0 = 0$$

Applied to  $c$ , we get  $c_L$  for the Lowpass:

$$c_L = \frac{1 + a_{u1} + a_{u2}}{1 + b_{u1} + b_{u2}}$$

For a Highpass the reference frequency is the Nyquist frequency which is  $\frac{f_s}{2}$ , so we get:

$$\omega_r = \omega_N = \pi$$

And  $c_H$  for the Highpass:

$$c_H = \frac{1 - a_{u1} + a_{u2}}{1 - b_{u1} + b_{u2}}$$

For a bandstop or notch filter, typically you choose 0Hz or  $\frac{f_s}{2}$  as reference frequency, depending on which one has the lowest scale factor:

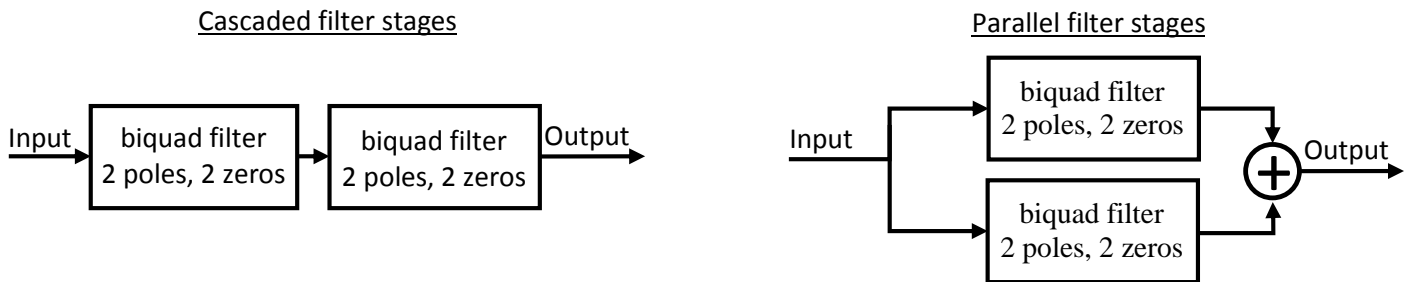
$$c_{BS} = \begin{cases} c_L, & c_L < c_H \\ c_H, & c_L \geq c_H \end{cases}$$

## 4. Higher order filter

In chapter 3 I described the z-Plane design process for a biquad filter. A biquad filter has an order of 2 that means it has 2 poles and 2 zeros maximum. You can do the whole design process for higher order filter too, but sometimes it is not practical to do this, just because the filter order is not fixed.

The solution for n-order filters is straight forward, you design several lower order filters, and put them in series (called “cascading”) or put them in parallel.

For a 4 pole and 4 zero filter, you just calculate 2 biquad filters with 2 poles and 2 zeros each and combine these stages.



That’s the easiest way to do it. Another way to get the same result with a little less sample processing is to combine both filter responses  $H_1(z)$  and  $H_2(z)$  into one response and then pass the resulting coefficients into the sample processing procedure.

For cascading stages you have to multiply  $H_1(z)$  with  $H_2(z)$  :

$$H(z) = \frac{B}{A} = \frac{B_1}{A_1} \times \frac{B_2}{A_2} = \frac{B_1 B_2}{A_1 A_2}$$

You end up with a  $H(z)$  with 5 **b** coefficients and 4 **a** coefficients, that you can use in the processing procedure.

For parallel stages, you have to add  $H_1(z)$  and  $H_2(z)$ , that is a little bit more complex:

$$H(z) = \frac{B}{A} = \frac{B_1}{A_1} + \frac{B_2}{A_2} = \frac{B_1 A_2 + B_2 A_1}{A_1 A_2}$$

You’ll also end up with a  $H(z)$  with 5 **b** coefficients and 4 **a** coefficients.

### 4.1 Cascade and parallel filter for programmers

Cascading filters in a program is actually very easy. You don't have to care about the exponents of  $z$  or even  $H(z)$ . Here is an example in pseudo code:

```
// this function can be used for cascading and parallel:
function multiplyCoeffs(c1, c2)
    c3 = new array with length (lengthOf(c1) + lengthOf(c2) - 1)
    fill c3 with 0
    for i=0 to lengthOf(c1)-1
        for j=0 to lengthOf(c2)-1
            c3[i+j] = c3[i+j] + c1[i]*c2[j]
        next
    next
    return c3
end

// a1,a2 and b1,b2 are arrays and hold all your coefficients
// a1[0] and a2[0] is always 1

b3 = multiplyCoeffs(b1, b2)
a3 = multiplyCoeffs(a1, a2)

// the new coefficients are now in a3 and b3
```

For parallel filters you can use the “multiplyCoeffs” function from the code above:

```
// a1,a2 and b1,b2 are arrays and hold all your coefficients
// a1[0] and a2[0] is always 1

b1a2 = multiplyCoeffs(b1, a2)
b2a1 = multiplyCoeffs(b2, a1)

// b1a2 and b2a1 can have different length
b3 = new array with length (max(lengthOf(b1a2), lengthOf(b2a1)))
for i=0 to lengthOf(b3)-1
    if i>=lengthOf(b1a2)
        b3[i] = b2a1[i]
    else if i>=lengthOf(b2a1)
        b3[i] = b1a2[i]
    else
        b3[i] = b2a1[i] + b1a2[i]
    end
next

a3 = multiplyCoeffs(a1, a2)

// the new coefficients are now in a3 and b3
```

... may be continued later...