SECTION 1.

Getting Acquainted

SECTION 1 1-1
1 Forward. 1-3
1.1 Tools for the Journey. 1-3
1.2 Determining coefficients. 1-4
1.2.1 First Example – Coefficients for a low pass filter. 1-4
1.2.1.1 Step 1 – Define the system in the frequency domain. 1-4
1.2.1.2 Step 2 – Perform an inverse Fourier Transform. 1-5
1.2.1.3 Step 3 – Translate to the discrete time domain. 1-5
1.2.1.4 Step 4 – perform z – transform. 1-6
1.2.1.5 Step 5 – Write result in "Normal Regenerative Form" 1-7
1.2.1.6 Step 6 - Solve for output G(z). 1-7
1.2.1.7 Step 7 – Inverse z – transform. 1-8
1.2.1.8 Step 8 – Check results. 1-8
1.3 Summary of Process. 1-11
1.4 Next Steps 1-11

1 Forward.

From the outside looking in, the topic of digital control loops can look like a complex task. Perhaps best left for expert mathematicians with expensive computing programs to feed their output to a masters level programming engineer to implement.

I think this view deserves to be challenged. With a little bit of effort, and a rudimentary understanding of z – transforms and the C programming language, I believe a typical engineer will be able to transfer their analog control design experience into the digital domain.

There are many texts available that do an outstanding job covering the topic of z – transforms and complex filters and related equations. While they give the readers great experience and challenges, they may be in excess of what is required to begin working in this discipline.

Furthermore, there are few texts that take the reader from a basic understanding through to an application. This treatment is a journey which attempts to take us from the basics through to an application and beyond.

1.1 Tools for the Journey.

I like to pack light. All you really need is a pencil and a piece of paper and a good understanding of algebra. But why push a car when you can drive it? It is best to get some good math processing software. I recommend MathCad®. This software is very easy to learn and relatively powerful. The input and output of the software is easy to read and understand and lends itself to be easily annotated.

Another tool I will be referring to is a Microchip® demo board. In this case it will be the Synchronous Buck demo board. This board has a microcontroller with a modest on-board DSP engine. To compliment the demo-board, we will use the in-circuit development programming tool, currently called ICD3® and we will also use their C30 compiler.

All of these tools represent a highly cost effective way to enter the field digital control. They may be substituted with whatever tools the reader has which are readily available, however these tools will be referred to in this development.

Two critical tasks are involved in designing a digital control loop. The first is determining the coefficients for the control loop while the second is determining how to practically implement the mathematical results. The first sections of this treatment will focus on determining the coefficients.

1.2 Determining coefficients.

Yes, I've brought up some scary concepts like z – transform, and I will be bringing up equally scary terms like Fourier transform. These areas will be tread on lightly. The reader should have some familiarity with these terms and understand why they exist.

1.2.1 First Example – Coefficients for a low pass filter.

First, let us start with a simple example. We start by saying we wish to gather coefficients for a simple low pass filter. I will arbitrarily select 10 kHz as a break point. The first decision we will have to make is what the desired "Tack" time is. This is the rate at which we will be processing the math. I'm going to call this time Tm which is one over the math frequency, Fm. It is desired to keep Fm much higher than any frequency we are interested in filtering. This is required since the digital process breaks down at frequencies approaching the math frequency. For this case I will select 1 MHz as the math frequency.

1.2.1.1 Step 1 – Define the system in the frequency domain.

The frequency domain for the LPF system is defined as follows…

$$H(\omega) = \alpha \cdot \frac{1}{\alpha + j \cdot \omega} \quad \text{where} \quad \alpha = 2\pi f_b \quad \text{and} \quad f_b = 10\text{kHz}$$

This should now look like an old familiar friendly definition for a single pole low pass filter. If not, maybe its time to break-off and go back to review those old textbooks. You can see at low frequencies, the gain will be 1, the break point will be where $\alpha = \omega$, or 10 kHz, and for high frequencies the gain will be ever decreasing and equal to $-j\alpha/\omega$.

1.2.1.2 Step 2 – Perform an inverse Fourier Transform.

Moving forward, the next step would be to get the inverse Fourier transform for the system described above. Some of us can do this by inspection, others can use Fourier lookup tables, while others may want to use MathCad®. I recommend getting comfortable with the software, if you are serious about designing digital control loops, the math may get a little difficult and time consuming.

By inspection, this example transforms to…

$$h(t) = \alpha \cdot e^{-\alpha \cdot t} \cdot u(t)$$

To use MathCad® instead of the manual method, the following function is typed in.

$$f_b = 10000 \qquad \alpha = 2\pi f_b$$

$$H(\omega) = \alpha \cdot \frac{1}{\alpha + j \cdot \omega}$$

Using their "Symbolic" toolbar, the following result is obtained.

$$H(\omega)\ \text{invfourier}, \omega \ \rightarrow 20000 \cdot \pi \cdot \exp(-20000 \cdot \pi \cdot t) \cdot \Phi(t)$$

Noting that $\alpha = 20000 * \pi$, this is consistent with the manual method.

1.2.1.3 Step 3 – Translate to the discrete time domain.

The reason we need the time domain representation of the system is because the digital system that we can implement must operate in the discrete time domain. That being said, the next step required is to translate the time domain system into the discrete time domain. To do this formally and correctly we would have to invoke convolution theorem, generally this is not difficult, but can get involved at times. Luckily, for most practical examples, this will boil down to replacing the time domain variable t with n*Tm, then scaling the whole result by Tm. Applying this simple approach results in the following.

$h(n) = T_m \cdot \alpha \cdot e^{-n \cdot \alpha \cdot T_m} \cdot u(n \cdot T_m)$, where u() is the unit step function.

1.2.1.4 Step 4 – perform z – transform.

So, how do we write a computer program to implement e-n ? The answer is we don't. One more transform is still necessary. We need to translate this expression to the z domain. The z domain is analogous to frequency domain, except that it relates to discrete time signals instead of continuous time. We work in that domain to simplify the e-n term which will be common in many of our problems.

Using z transform tables, or by inspection, the z transform is as follows.

$$H(z) = T_m \cdot \alpha \cdot \frac{1}{1 - e^{-\alpha \cdot T_m} \cdot z^{-1}}$$

Alternatively, MathCad® gives the result

$$h(n)\ \text{ztrans}, n \rightarrow \frac{1}{50} \cdot \pi \cdot \frac{z}{\left(z - \exp\left(\frac{-1}{50} \cdot \pi\right)\right)}$$

Again, these results agree with each other. We prefer the form of the first result. I will call this form the "normal regenerative form" or just the "normal" form. This form is desirable for the next step of processing. We want the denominator to start with a leading 1. It will be seen shortly that this leading term will be a coefficient for the result, so naturally we want it to be 1. We also need to have all the values of z raised to a negative integer since this represents a delay which can be easily implemented on a microprocessor or other digital device. A positive exponent for z would indicate some point in the future which may be a little difficult to implement.

The good news is that the finish line is in sight with a few more substitutions.

1.2.1.5 Step 5 – Write result in "Normal Regenerative Form"

In this example, in the previous step the results were already in the "normal" form. Once the equation is in this form, the coefficients are easily identified. After we have some practice, we can stop here and use the coefficients directly. For this example, we will be demonstrating the significance of the normal form.

The normal is defined as follows.

$$H(z) = \frac{b_0 + b_1 \cdot z^{-1} + \cdots + b_n \cdot z^{-n}}{1 - a_1 \cdot z^{-1} - \cdots - a_m \cdot z^{-m}}$$

1.2.1.6 Step 6 - Solve for output G(z).

We are ultimately trying to build a machine that will give us a series of numbers as an output g(n). This series is a result of out system h(n) operating on an input series f(n).

Represented mathematically,

g(n) = f(n) * h(n) , where * is the convolution function.

We choose to work in the discrete frequency domain, because like the continuous frequency domain, the convolution of the time domain translates to simple multiplication.

Here, $G(z) = F(z) \cdot H(z)$

We can transpose this as follows.

$$\frac{G(z)}{F(z)} = H(z)$$

We'll add to the mix a few definitions to clean up the math.

$$a_1 = e^{-\alpha \cdot T_m} \qquad b_0 = T_m \cdot \alpha$$

Here are the substitutions implemented.

$$\frac{G(z)}{F(z)} = \frac{b_0}{1 - a_1 \cdot z^{-1}}$$

Re-arranged, we get…

$$G(z) = b_0 \cdot F(z) + a_1 \cdot G(z) \cdot z^{-1}$$

1.2.1.7 Step 7 – Inverse z – transform.

We now need to get back to the discrete time domain, which is where we can process digital instructions. Fortunately, we can all do this by inspection at this point if we recognize that z-i is simply a delay of i steps. Applying this we get…

gn = b0 fn – a1 gn-1

There it is, finished, not too bad in terms of effort. Well, there are a few more things. First, we need to check the results and second we need to implement them on real hardware. I'm going to save the implementation for a slightly more complex example later on, the checking we can do now.

1.2.1.8 Step 8 – Check results.

It is very important to check the results. A small mistake in calculations could resulting hours of wasted time at the implementation phase. Once a checking process is set up, it can take only a few minutes to review the results.
We always need to check results for at least convergence, if not also for performance. The method I like to use is to apply a test pulse and see that the result converges to a value. While this will not unconditionally prove the system is stable for every input, it is a strong indicator. You should also note that the accuracy of the coefficients can at times affect the convergence of the system. When possible, you should use numbers you can resolve on the target digital processor.

Several tools can be used to implement the checking that I've proposed here. One can use MathCad®, Excel® or some other math program. I prefer MathCad®, because it is easy to use and easy to set up to provide a visual check.

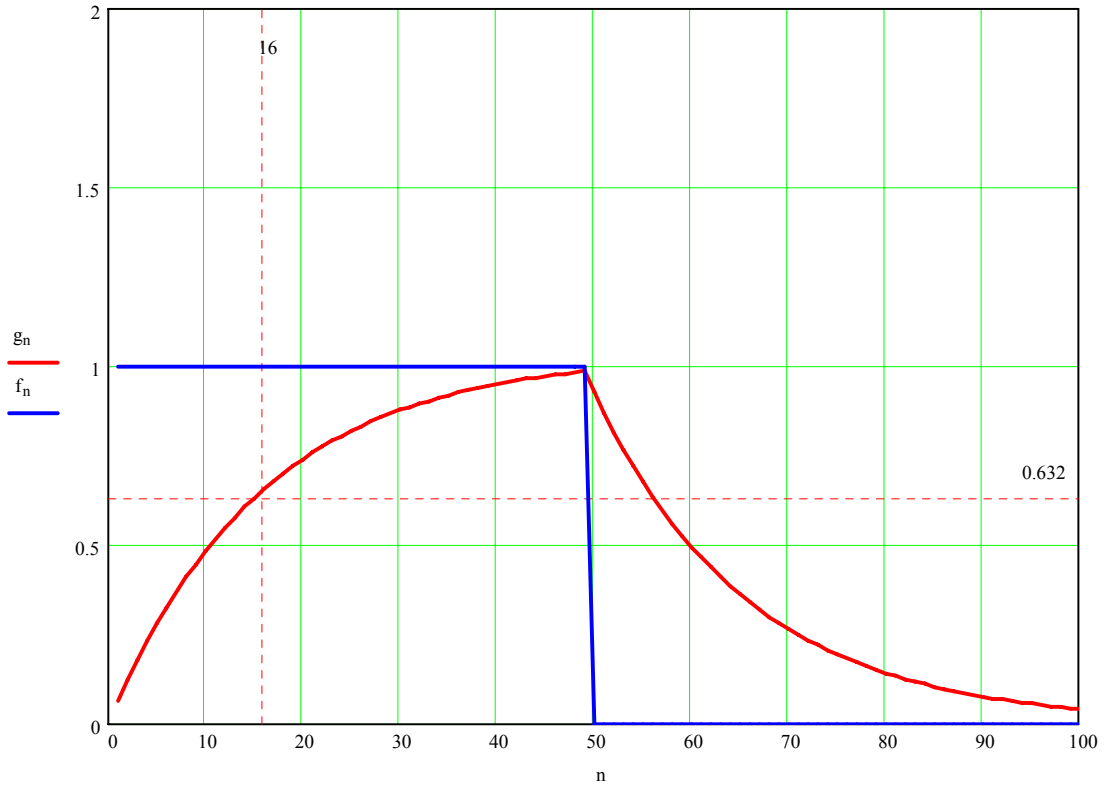
We'll choose to calculate 100 steps. Since the tack time is 1uS, this will represent 100 uS. The input will be a 50 step (uS) pulse with an amplitude of 1.

$n = 1..100 \qquad 100 \cdot T_m = 100 \times 10^{-6}$

$f_n = u(n) - u(n - 50) \qquad g_0 = 0$

$g_n = b_0 \cdot f_n - a_1 \cdot g_{n-1}$

The digital domain response is plotted.



We can see the result converges to zero. We can also observe the time-constant for the filter is about 16uS. This is exactly what we should expect for a 10 kHz LPF. The test so far looks good.

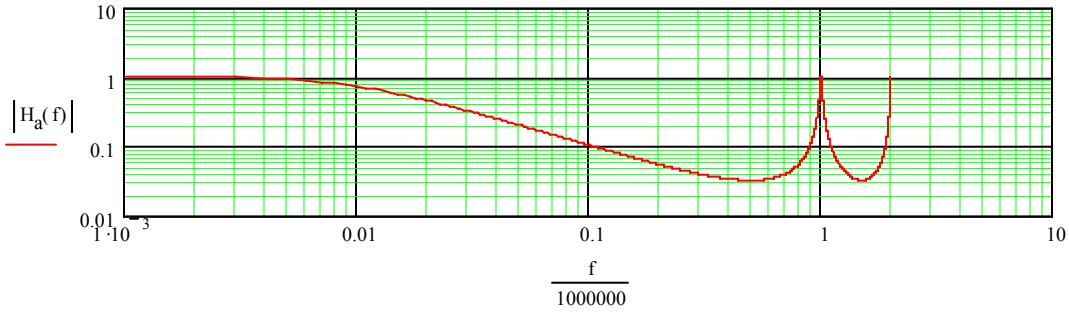
It would also be interesting to see what these coefficients represent in the continuous frequency domain. Some mathematically inclined individuals would argue that it is better to check in the z domain. Sure we can plot the z domain and contemplate the significance of the unit circle and right half plane zeros. We do, however, live in the frequency domain. Our instruments give us frequency domain data, and the initial input for this, and probably every filter we design, was in the frequency domain, and so it makes sense that a final check is performed in that domain. There are several ways this can be implemented in software, but the easiest method is to perform a simple substitution and graph.

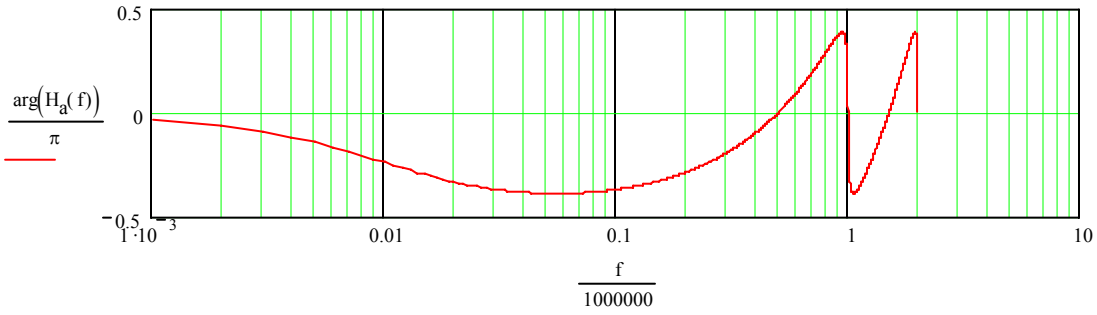
Given that we already have H(z), we can determine its analog equivalent, Ha(f) by substituting z with ej2πfTm. Considering that we are using a math program, we don't even have to perform the manipulations, we can just plug the value in.

The frequency domain response is plotted.

$$H_a(f) = H\left(e^{j\cdot 2\pi\cdot f\cdot T_m}\right)$$

$$f = \frac{f_b}{10}, \frac{f_b}{5} .. 200\cdot f_b$$





We expect a low-pass filter with an initial breakpoint of 10 kHz, the gain should fall off with a constant slope. The phase should start out at 0, at the break frequency it should be (-π/4), then it should asymptotically approach -π/2. In these graphs we observe the gain behaving as predicted out to ~ 500 kHz (Tm / 2), the phase follows as desired out to ~ 60 kHz. If better performance is needed, we would need to change the filter or increase the math rate.

1.3 Summary of Process.

Here are the steps we've walked through so far.

Process	Description	Why / Comments
Step 1	Describe the system in the frequency domain.	Define what you would like the end result to represent. Engineers are most familiar with frequency domain representations of systems.
Step 2	Perform an inverse Fourier transform to get the system definition into the time domain.	This is required for following steps. Engineers are typically not familiar with time domain representations of systems
Step 3	Translate to the discrete time domain by inserting substitutions and scaling.	Required since digital devices can only work in the discrete time domain.
Step 4	Perform z-transform bringing problem into the discrete frequency domain (z - domain).	Simplify math.
Step 5	Perform algebra to express denominator with a leading 1, and terms of z raised to a negative integer.	Necessary for simplification.
Step 6	2nd part of algebra to solve for the desired output G(z).	Necessary for simplification.
Step 7	Simple inverse z- transform to get back to discrete time domain.	Provides the end result.
Step 8	Check for convergence and evaluate performance.	Save time by clearly understanding the results before implementing them.

1.4 Next Steps

An accompanying MathCad sheet has been prepared to follow these steps.

Section 2 will introduce some more complex examples and a much simpler process to derive coefficients. That process builds on the steps outlined here.