Digital Control Loop Design using the Construct Method

Abstract

This paper introduces foundational concepts in digital signal processing (DSP) with a focus on designing control algorithms for power supplies. The material is broken into digestible, intuitive segments to support individual learning and gradual skill development. Practical shortcut methods are presented to reduce complexity and accelerate progress. Accompanying Mathcad worksheets are included to encourage hands-on exploration and deeper understanding.

A key contribution of this tutorial is the introduction of a novel approach that uses a small set of fundamental ***constructs*** to assemble digital controllers. When paired with a concise set of rules, these constructs provide a clear and repeatable path to successful implementation. While DSP is a broad field, this targeted method makes it accessible to Power Engineers who traditionally rely on analog control techniques. The goal is not to replace formal education, but to fill in practical gaps, connect theory to real-world applications, and offer a streamlined process from concept to working solution.

Notices

COPYRIGHT NOTICE

Disclaimer of Liability

The information, observations, and conclusions provided in this document are offered as a free informational resource. All materials and information are provided "AS IS" and do not constitute professional or legal advice. No representation or warranty of any kind, express or implied, is made regarding the accuracy, adequacy, validity, reliability, availability, or completeness of the information contained in this document. Users should not use this information without independent verification and assume all risks associated with its use. Under no circumstances shall any liability be incurred for any loss or damage resulting from the use of or reliance on the information provided herein. Use of this document and reliance on its contents is solely at the user's own risk.

Table of Contents

List of Equations

List of Figures

1 Introduction

Digital control loops are often perceived as a complex and specialized domain — one seemingly reserved for advanced mathematicians and engineers with access to powerful computational tools. However, this perception can be misleading.

With a foundational understanding of Z-transforms and the C programming language, engineers with experience in analog control can effectively transition their skills into the digital realm. This transition does not require mastery of abstract mathematics or elaborate simulation software, but rather a practical mindset and a structured approach.

While many excellent texts delve deeply into the theory of Z-transforms and digital filter design, they can sometimes overwhelm those looking for a more application-oriented path. This resource is designed to bridge that gap — guiding the reader from essential principles to real-world implementation.

The goal is to provide a clear, practical journey from basic understanding to working digital control systems, with enough context to build confidence and competence along the way.

1.1 Tools for the Journey

This exploration begins with a modest set of technical requirements. At a minimum, a working knowledge of algebra is sufficient to follow the underlying concepts. However, to streamline analysis and improve clarity, the use of advanced mathematical software is strongly recommended. Mathcad®, in particular, offers significant advantages due to its intuitive interface and superior documentation capabilities, producing worksheets that are both readable and easy to annotate.

Designing a digital control loop involves two primary tasks: first, determining the coefficients that define the loop behavior, and second, implementing those coefficients in a practical, real-time system. This section focuses on the first task — deriving the coefficients — while laying a foundation for implementation strategies discussed later.

While the individual steps involved in coefficient calculation are important, the most significant contribution of this work lies in the introduction of constructs — modular, reusable building blocks for digital control design. These constructs offer a powerful framework that enables engineers to assemble a wide variety of control networks with efficiency and clarity.

1.2 Euler's Formula

Euler's formula is said to be the cornerstone of signal processing in general.

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

Derivations will not be the focus here; however, this formula originates from evaluating the infinite Taylor series expansion of the exponential function and extending it to the imaginary axis.

The important take away for engineers is that $e^{j\theta}$ represents a phasor.

1.3 Laplace Transform

The Laplace Transform is a special case of the Fourier Transform designed to solve ordinary differential equations (ODE) effectively.

Laplace Transform and Reverse Transform

$$F(s) = \int_0^{\infty} f(t)e^{-st}dt \qquad f(t) = \frac{1}{j2\pi}\int_{\sigma-j\infty}^{\sigma+j\infty} e^{st}F(s)ds$$

This topic will not be developed in detail either; however, it is important to recognize it as a valuable tool that enables the use of algebraic techniques in the frequency domain as a substitute for more complex time-domain mathematics. It is common practice to use lookup tables or computational software to solve such equations. For the purposes of this document, a short set of transform pairs will be required.

Description	$f(t)$	$F(s)$
Step	$1\,u(t)$	$\frac{1}{s}$
Decay	$e^{-at}u(t)$	$\frac{1}{s+a}$
Charge	$(1-e^{-at})u(t)$	$\frac{a}{s\,(s+a)}$
Frequency Shift	$e^{at}\,f(t)u(t)$	$F(s-a)$
Time shift	$f(t-a)u(t-a)$	$e^{-as}\,F(s)$
Convolution	$\int_0^t h(\tau)f(t-\tau)d\tau$	$\mathrm{H}(s)F(s)$
Impulse	$\delta(t-t_0)$	$e^{-s\,t_0}$

Figure 1-1: Laplace Transform Pairs

1.3.1 Low Pass Filter Example using Laplace Transform

The discussion begins with a simple example to demonstrate the usefulness of the Laplace transform. A single-pole low-pass filter, shown in ***Figure 1-2*** , will serve as the illustrative case.

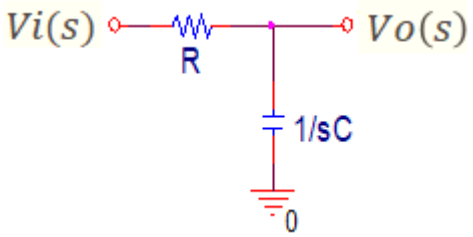


Figure 1-2: Low Pass Filter Schematic

Performing a nodal analysis yields…

$$\frac{Vo(s)-Vi(s)}{R}+\frac{Vo(s)}{\frac{1}{sC}}=0$$

If we create a term α such that $\alpha=\frac{1}{RC}$, the equation for the gain of the system becomes…

$$H(s)=\frac{G(s)}{F(s)}=\frac{Vo(s)}{Vi(s)}=\frac{\alpha}{s+\alpha}$$

Equation 1-1: Gain of a Single Pole Low Pass Filter

This form is written to make the 'breakpoint' readily identifiable. In this case the breakpoint isα.

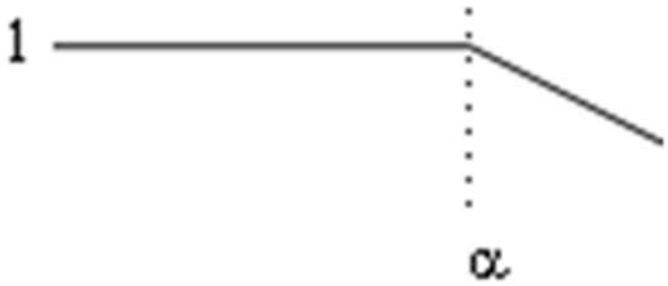


Figure 1-3: Single Pole Low Pass Filter Transfer Function H(s)

There is a "Breakpoint" at $|s|=\alpha$

The analysis will remain in the frequency domain, as both the problem statement and the resulting expressions are currently intuitive and rely solely on algebraic manipulation. To illustrate the relative difficulty of working in the time domain, a brief comparison is provided. Using the Laplace transform table shown in ***Figure 1-1*** , the corresponding time-domain transfer function is: $h(t) = \alpha\, e^{-\alpha t}$

In order to determine the LPF output in the time domain it is required to use convolution.

$$\mathrm{g(t)} = \int_0^t \mathrm{h}(\tau)\mathrm{f}(t-\tau)d\tau$$

Instead of using integration, it is better to stay in the frequency domain. The problem simplifies to multiplication.

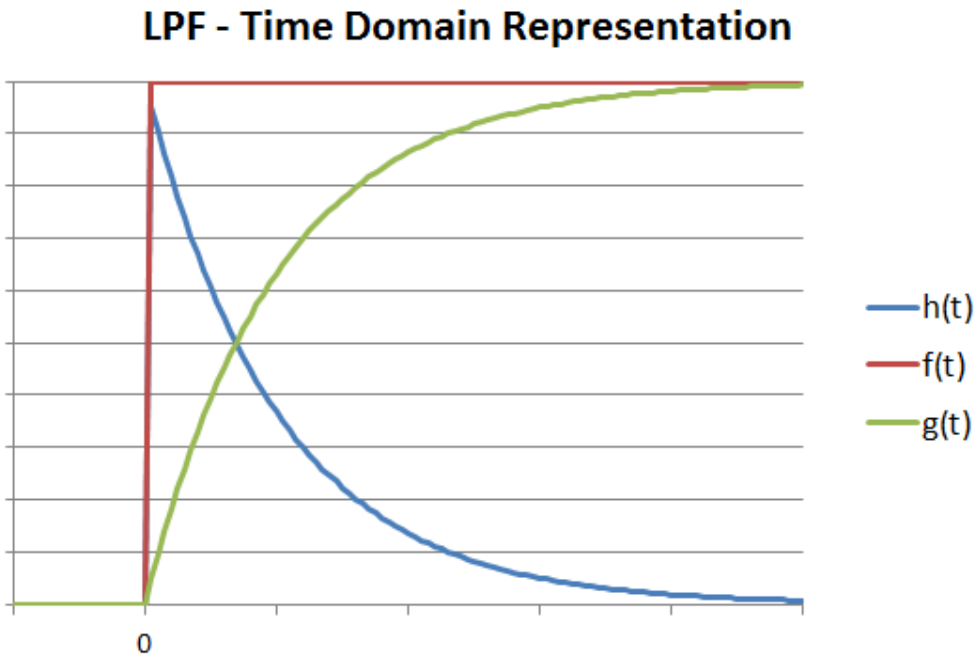
$$\mathrm{G(s)} = H(s)F(s)$$

The filter's performance can be analyzed by applying a step function as the input.

$$\mathrm{G(s)} = H(s)F(s) = \frac{\alpha}{s+\alpha}\ \frac{1}{s}$$

Now the transform table is used to get the time domain solution.

$$g(t) = (1 - e^{-\alpha t})u(t)$$



1.4 Z – Transform

The Z transform is a discrete frequency domain function. It is analogous to the Laplace transform in the discrete domain. It was created as a tool to solve difference equations by allowing the use of algebra in the Z domain as opposed to convolution and integration in the discrete time domain.

$\boldsymbol{X(z) = \sum_{n=0}^{\infty} x[n] z^{-n}}$ Z Transform

$\boldsymbol{x[n] = \frac{1}{j2\pi} \int X(z) z^{n-1} dz}$ Inverse Z Transform

	Time Domain	z-Domain
Step	$u[n]$	$\frac{1}{1-z^{-1}}$
Decay	$e^{-anT}u[n]$	$\frac{1}{1-e^{-aT}z^{-1}}$
Approach	$(1-e^{-anT})u[n]$	$\frac{1-e^{-aT}}{(1-z^{-1})(1-e^{-aT}z^{-1})}$
Time Shift	$x[n-k]$	$z^{-k}X(z)$
Convolution	$h[n] * f[n]$	$H(z)F(z)$
Impulse	$\delta[n-n_0]$	$e^{-j\alpha n_o}$

Figure 1-4: Z Transform Table

2 The Cookbook Process

The following outlines several steps (up to 8) to arrive at a process to take an analog frequency domain problem and translate it into a discrete time domain solution.

The process begins with a specified $H(s)$. Translate $H(s)$ to the z domain to get $H(z)$. Then translate $H(z)$ to the discrete time domain arriving at $g(n)$.

Microprocessors and FPGAs operate in the discrete-time domain. The equation for g(n) can be entered directly into their respective environments for execution.

While the process may appear straightforward, successful implementation requires expressing $H(z)$ in a normalized recursive form upon reaching the z-domain. Specifically, the denominator should have a leading coefficient of 1, as shown below.

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

Equation 2-1: H(z) Normal Recursive Form

This is important because $\frac{G(z)}{F(z)} = H(z)$. By cross multiplying ***Equation 2-1***, an expression for G(z) is obtained, which can be solved algebraically. The resulting expression can then be translated to the desired g(n) form by inspection.

The ***unique twist*** is that the process will be used to develop 4 basic constructs. Once developed, the resulting equations will remain the same, only the variable of where to place the breakpoints will change.

More complex designs can be implemented by combining any combination of the 4 constructs, and then perform algebra to arrive at the final discrete time equations.

2.1 Step 1 – Specify the requirement

The analysis begins with the Low Pass filter example. A numeric value of 10 kHz is assigned to the desired breakpoint since Mathcad will be used for demonstration purposes.

$H(s) = \frac{\alpha}{s+\alpha}$ where $\alpha = 2\pi f_b$ and $f_b = 10kHz$

2.2 Step 2 – Translate to the Time Domain

Using Tables, Mathcad, or just by inspection, translate this 'requirement' to the time domain.

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

2.3 Step 3 – Determine h(n)

The term for $h(n)$ is now needed. This is straight forward. Replace t with nT_M , where n is the sample number and T_M is the rate at which math is executed. Then it is necessary to scale by T_M . The result is written below.

$h(n) = T_M \alpha e^{-n\alpha T_M} u(nT_M)$ For this example, the math frequency, T_M is set to 1µS.

Scaling is needed to normalize the result. Consider an integrator which is summing samples of a data stream that is always positive. If it is sampling once per second, after 10 seconds it will sum 10 samples of the data stream and get to a certain result. Next consider the same integrator sampling at once per microsecond. In 10 seconds it would have sampled and added the same data stream 10 million times. The result will be 1 million times higher than the first. If each result is scaled by the sample rate, each case will be made to provide a similar result.

2.4 Step 4 – Translate to the z Domain

Using the Z transform table, Mathcad, or by inspection, the result is transformed to the z domain.

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

2.5 Step 5 – Format the Result

The result must now be formatted to the normal recursive form consistent with ***Equation 2-1***. In this case it is already in that format.

2.6 Step 6 – Solve for G(z)

As stated earlier, $\frac{G(z)}{F(z)} = H(z)$.

In this case…

$$\frac{G(z)}{F(z)} = \frac{\alpha T_M}{1 - e^{-\alpha T_M} z^{-1}}$$

Equation 2-2: Solving for G(z)

Now the leading 1 in the denominator becomes useful.

$G(z) = b_0 F(z) + a_1 G(z) z^{-1}$, where $b_0 = T_M \alpha_1$, and $a_1 = e^{-\alpha T_M}$.

2.7 Step 7 – Perform an Inverse z Transform

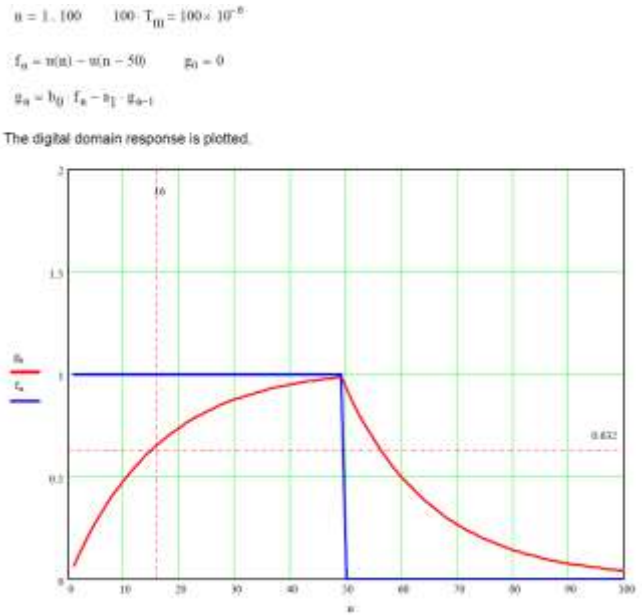
By using the time shift theorem in table of ***Figure 1-4***, we get the difference formula.

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

The first of four constructs Completed!

2.8 Step 8 – Check Your Results

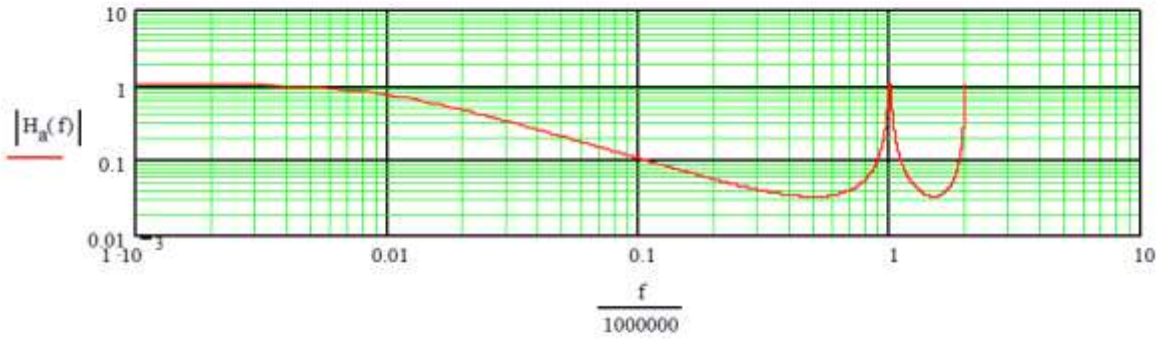
Not quite done. It is always best to check results. This can be done in actual hardware, excel, or Mathcad. It is best to use Mathcad to get time domain and frequency domain results.

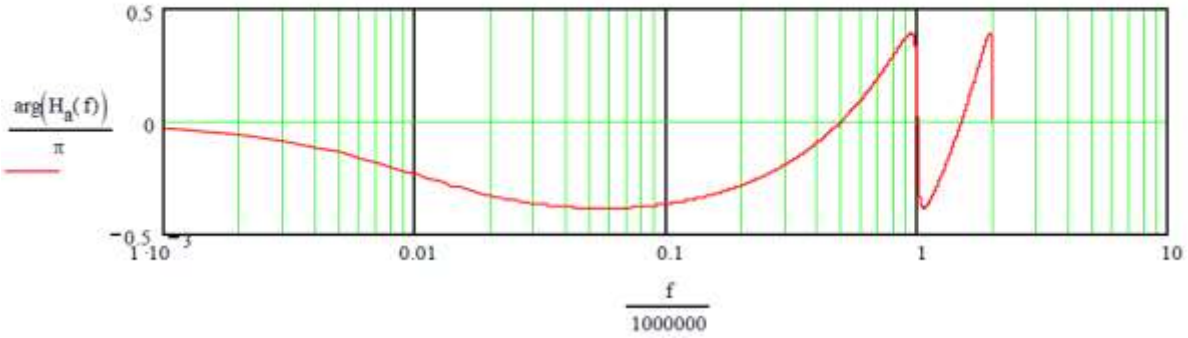


Frequency Domain Plot of the Discrete LPF

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

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





To create the graph, z is replaced with $e^{j2\pi fT_m}$. After all, that is what z is, a phasor.
There are a few interesting points to note. First is that at 10 kHz, the gain is -3db and the phase is $\frac{-\pi}{4}$. Therefore, the problem statement is solved and mission accomplished.

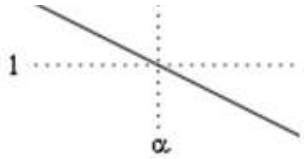
Next, notice that the filter stops working at some point. When fT_M is at $\frac{1}{2}$, z is at π.
At this frequency, the filter has its greatest attenuation. Then as z moves further counter-clockwise, from π to 2π, the filter produces the complex conjugate of the attenuation values that it produced in the first half of the sweep.

For purposes of attenuation, we should not count on the filter anywhere above ½ the math frequency.

3 The Four Constructs

The details of the remaining constructs are covered on an accompanying Mathcad worksheet. The results are provided in the following paragraphs.

3.1 Integrator Construct

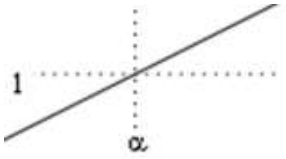


$$H(z) = \frac{b_0}{1 - z^{-1}} \quad \text{where} \quad b_0 = \alpha T_M$$

3.2 Single Pole (LPF) Construct

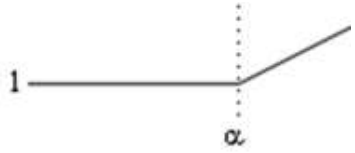
$$H(z) = \frac{b_0}{1 - a_1 z^{-1}} \quad \text{where} \quad b_0 = \alpha T_M \text{ and } a_1 = e^{-\alpha T_M}$$

3.3 Differentiator Construct



$$H(z) = b_0 + b_1 z^{-1} \quad \text{where} \quad b_0 = \frac{1}{\alpha T_M} \text{ and } b_1 = \frac{-1}{\alpha T_M}$$

3.4 Single Zero Construct



$$H(z) = b_0 + b_1 z^{-1} \quad \text{where} \quad b_0 = \frac{1}{\alpha T_M} \text{ and } b_1 = \frac{-e^{-\alpha T_M}}{\alpha T_M}$$

4 Applying Constructs

Armed with the constructs, there is no need to use transforms, but place more focus more on algebra.

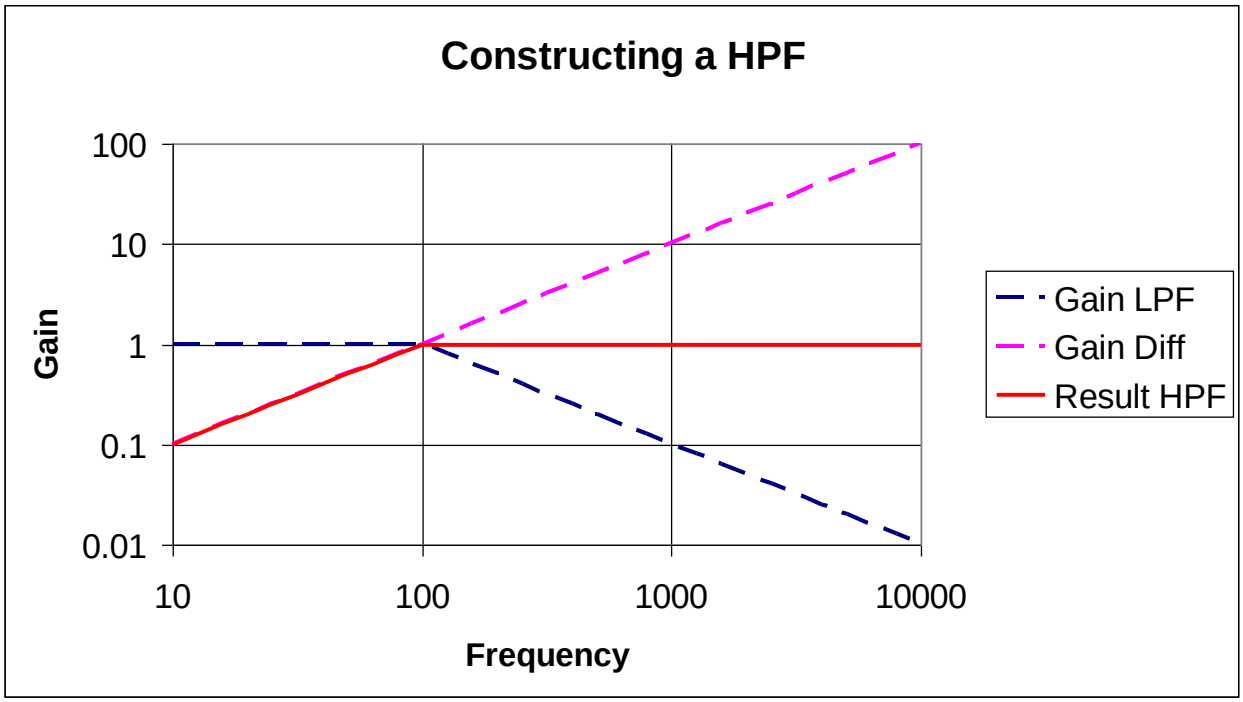
Many filter configurations can be assembled by multiplying various constructs, then only perform algebra to get the resulting equation into the normal recursive form. Once in that form, the coefficients drop out by inspection.

Notice that the Single Zero construct and the Single Pole construct were made to have a value of 1 at frequencies below the breakpoint. This was done intentionally, so that when multiplying these with other constructs, they will have no effect at frequencies below their breakpoints.

Likewise, for the Integrator and Differentiator constructs, the critical frequency where the gain equals 1 is identified, so that when multiplied with other constructs, the critical frequency will be preserved. This has the effect of adding a slope if required and setting the total gain with a single coefficient.

4.1 Constructing a High Pass Filter

The first example will be to construct a Single Pole High Pass Filter. In the frequency domain, it is desired to start with a single slope increasing, then at some breakpoint, the gain will flatten and remain flat.
To achieve this, the process begins with a Differentiator Construct, this gives us the positive slope, and then follow that with a LPF Construct which will introduce a pole and flatten out the gain.



For purposes of demonstration, a break frequency of 100 Hz is selected and the math update time of 25µS is selected.

$f_b = 100Hz \qquad T_M = 25\mu S$

The s domain breakpoint is calculated as $\alpha_b = 2\pi f_b$

Now two of the constructs from section 3 are used.

$H_{Diff}(z) = \frac{1-z^{-1}}{T_M \alpha_b} \qquad H_{LPF}(z) = T_M \alpha_b \frac{1}{1 - e^{-T_M \alpha_b} z^{-1}}$

These two constructs are in series. To get the resulting gain, they are multiplied.

$$H_{HPF}(z) = H_{Diff}(z)\, x\, H_{LPF}(z) = \frac{1 - z^{-1}}{1 - e^{-T_M \alpha_b} z^{-1}}$$

The coefficients are visible by inspection.

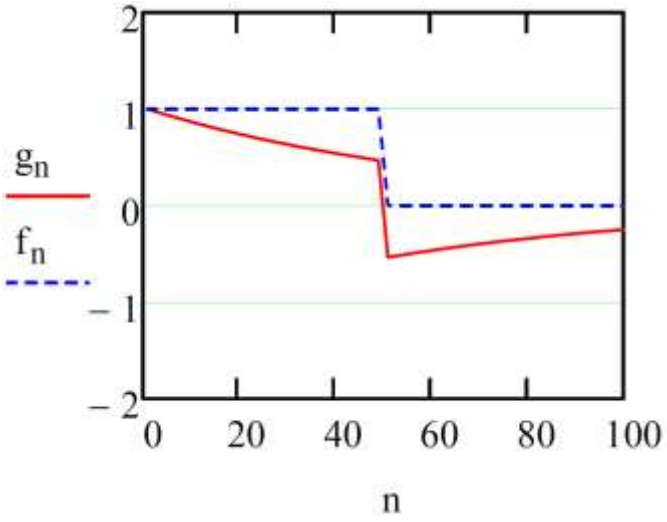
$b_0 = 1 \quad b_1 = -1 \quad a_1 = e^{-T_M \alpha_b}$

Done! Only algebra and a few short steps were needed.

4.1.1 High Pass Filter Evaluation

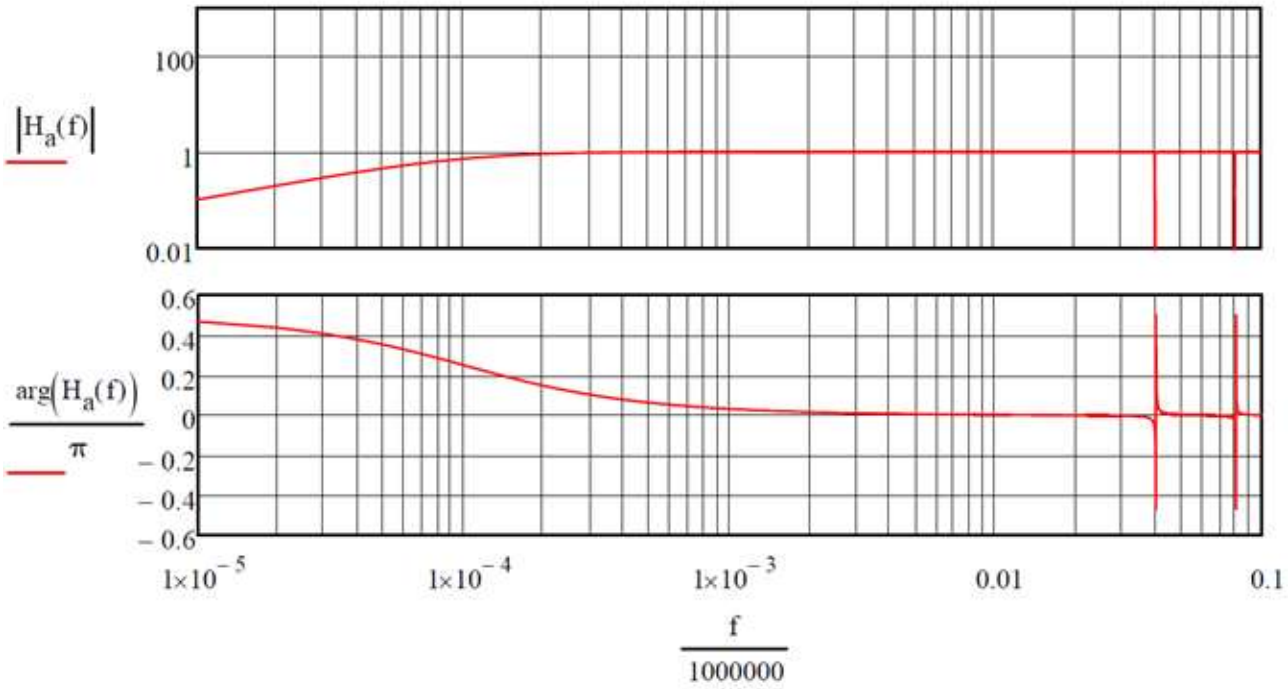
The equation can be entered in Mathcad for evaluation. In the discrete time domain…

$$g_n = f_n - f_{n-1} + a_1 g_{n-1}$$



Then in the frequency domain,

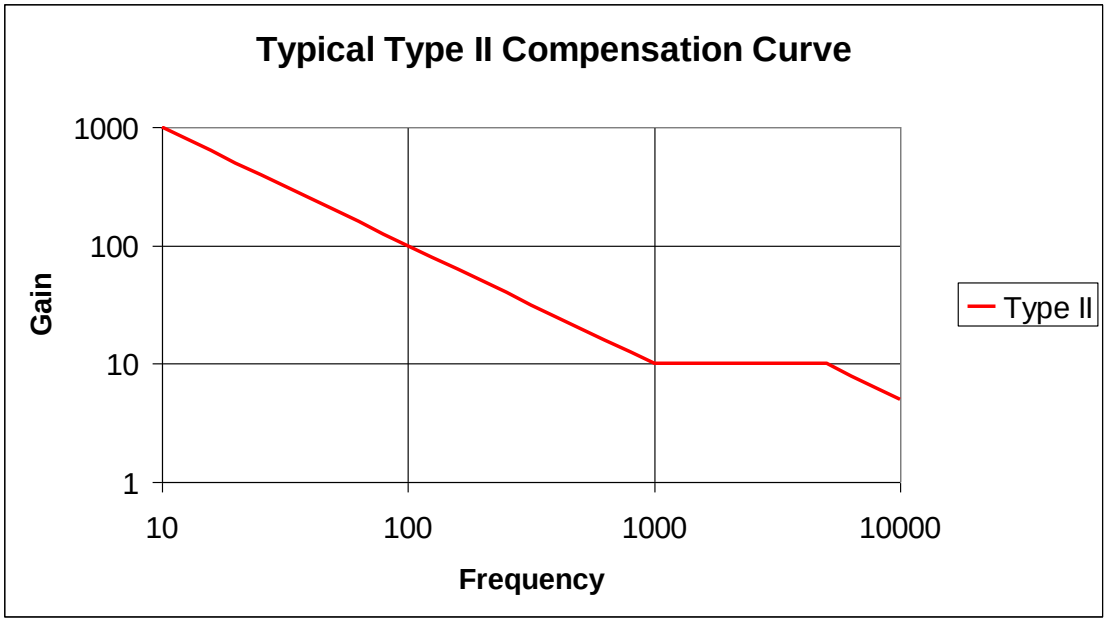
$$H_a(f) = H_{HPF}(e^{j2\pi f T_M})$$



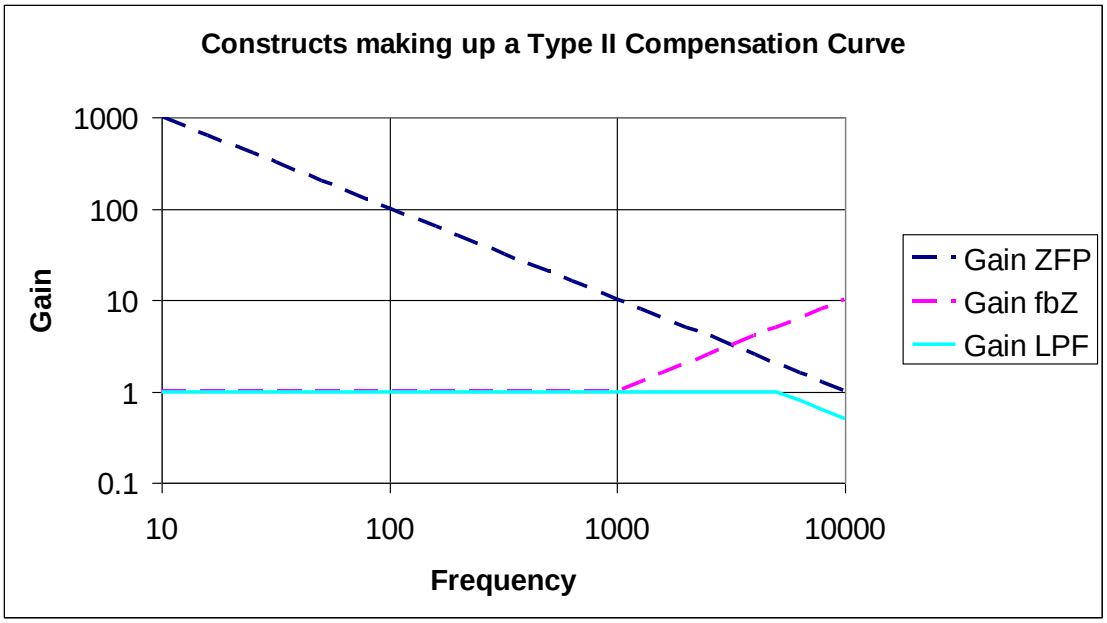
Here it is shown that the gain is increasing with a positive slope of 1 until the frequency reaches the breakpoint of 100 Hz. At that point, the gain is -3db and the slope if flattening.

4.2 Construct Example to Build a Type II Network

A Type II network starts as an integrator then a zero comes in at some breakpoint followed by a pole at the next breakpoint. A gain greater than 1 in introduced as well.



It should be obvious that the problem is solved by starting with an integrator, and then introduce a single zero at the first breakpoint, then a pole at the second breakpoint. The 3 constructs are shown below.



To set the gain as required, the integrator cross over must be 10 kHz, the breakpoint for the zero is at 1 kHz, and the final pole is at 5 kHz.

$f_{b1} = 10kHz$ $f_{b2} = 1kHz$ $f_{b3} = 5kHz$

The same math processing rate of 25µS will be used.

$\alpha_1 = 2\pi \cdot f_{b1}$ $\alpha_2 = 2\pi \cdot f_{b2}$ $\alpha_3 = 2\pi \cdot f_{b3}$

Now the constructs from section 3 are used.

$$H_{Int}(z) = \frac{\alpha_1 \cdot T_m}{1 - z^{-1}} \quad H_{Zero}(z) = \frac{1}{\alpha_2 \cdot T_m} + \frac{-e^{-\alpha_2 T_m}}{(\alpha_2 \cdot T_m)} \cdot z^{-1} \quad H_{LPF}(z) = \frac{\alpha_3 \cdot T_m}{1 - e^{-\alpha_3 T_m} \cdot z^{-1}}$$

The gain for the Type 2 network will be the multiplication of these 3 terms.

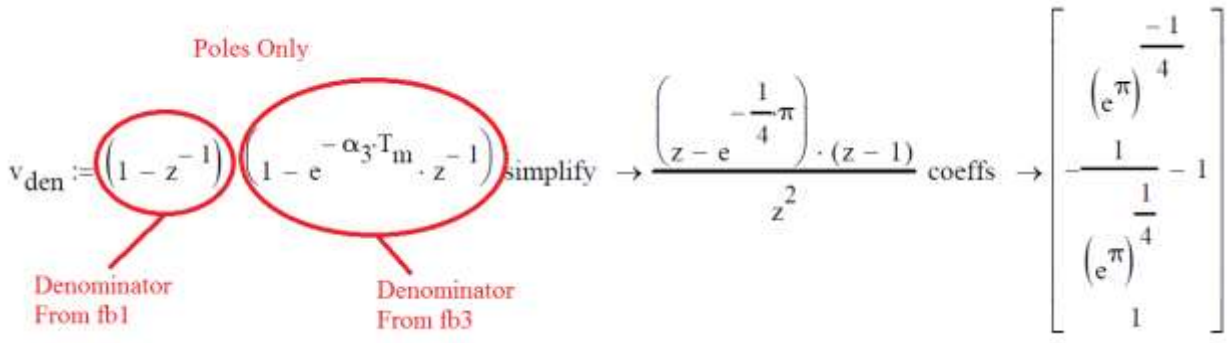
$$H_{T2}(z) = -H_{Int}(z)\, H_{Zero}(z)\, H_{LPF}(z)$$

Use of the negative sign is optional. It needs to be applied if the feedback signal is generated by taking the measurement then subtracting the set-point ($V_{FB} = V_{Meas} - V_{SP}$). Many digital controllers have library routines that generate V_{FB} in the opposite direction ($V_{FB} = V_{SP} - V_{Meas}$). The required inversion is already built into the library function. They have done this to match the legacy analog ICs which use positive values only so that a single positive Vcc voltage would be required.

As promised, only algebra is required. Unlike the previous example, which was intentionally made to have terms cancel out, there is no cancelation here. There remains a tedious exercise to find the coefficients.

Luckily, this only needs to be worked out once, then the heavy lifting can be done using Mathcad, Excel, Java etc. Mathcad is the best for documentation purposes, it is shown here.

First, the denominators will be combined and the coefficients extracted.



It looks complicated, but only copy and paste is required to place these into the equation.

A few minor adjustments are required. First because Mathcad is producing coefficients for z as opposed to z^{-1}. To address this we need to reverse the order. Second, when we solve for G(z) the a_n values are moved to the opposite side, so we will need the additive inverse. (See ***Equation 2-2***)

$p := 0 .. 2$

$a_p := \left[\left(-v_{den}\right)_{2-p}\right]$ $\quad a = \begin{pmatrix} -1 \\ 1.4559 \\ -0.4559 \end{pmatrix}$

The next step involves determining the b coefficients for the numerator. Writing a term for the numerator is not as immediately visible as was the denominator. To simplify the effort, the total term, $H_{T2}(z)$, is multiplied by the denominator, which was visible. Then let Mathcad do the work.

$v_{num} := H_{T2}(z) \cdot \left[\left(1 - z^{-1}\right) \cdot \left(1 - e^{-\alpha_3 \cdot T_m} \cdot z^{-1}\right)\right]$ simplify $\rightarrow \frac{5 \cdot \pi \cdot e^{-\frac{\pi}{20}}}{2 \cdot z} - \frac{5 \cdot \pi}{2}$ coeffs $\rightarrow \begin{bmatrix} \frac{5 \cdot \pi}{2} \\ 2 \cdot \left(e^{\pi}\right)^{\frac{1}{20}} \\ -\frac{5 \cdot \pi}{2} \end{bmatrix}$

Complete Term — Denominator

Now for the final extraction, the order needs to be reversed as before,. The inverse is not needed since the b_n terms are already on the correct side of the equation.

$p := 0 .. 1$

$b_p := \left[\left(v_{num}\right)_{1-p}\right]$ $\quad b = \begin{pmatrix} -7.854 \\ 6.7123 \end{pmatrix}$

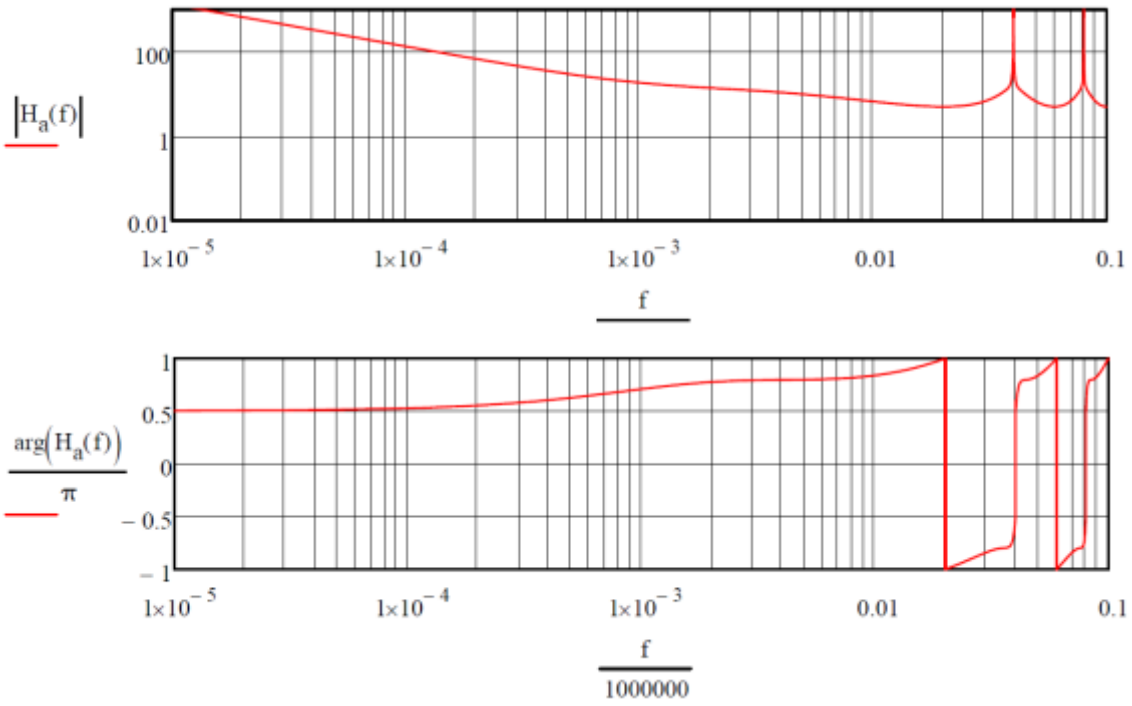
We now have the answer in the z domain and the discrete time domain.

$$H_{T2}(z) = \frac{b_0 + b_1 \cdot z^{-1}}{1 - a_1 \cdot z^{-1} - a_2 \cdot z^{-2}}$$

$$g_n = b_0 \cdot f_n + b_1 \cdot f_{n-1} + a_1 \cdot g_{n-1} + a_2 \cdot g_{n-2}$$

Notice – There are 2 poles (a_1 and a_2) and 1 zero (b_1). Terms a_0 and b_0 do not create any frequency dependent terms. The term a_0 is always equal to 1 and is incorporated into g_n.

4.2.1 Result Check for the Type II Network



The frequency domain is plotted using the same substitution as before. The resulting gain and breakpoints are in line with the problem statement.

5 IIR vs. FIR

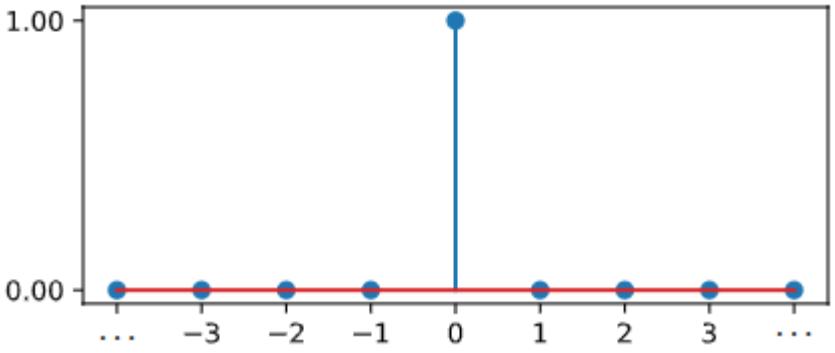
When working in the discrete time domain, it is best to understand these terms and the baggage that comes with each of them. We'll dissect these terms in reverse order.

5.1 Impulse Response

This part of the term refers to the frequency domain response when a time domain impulse is applied. This is true for continuous and discrete situations. We will focus on the discrete case.

In discrete time, an impulse, or delta function is one for n=0 and 0 for all other n.

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$



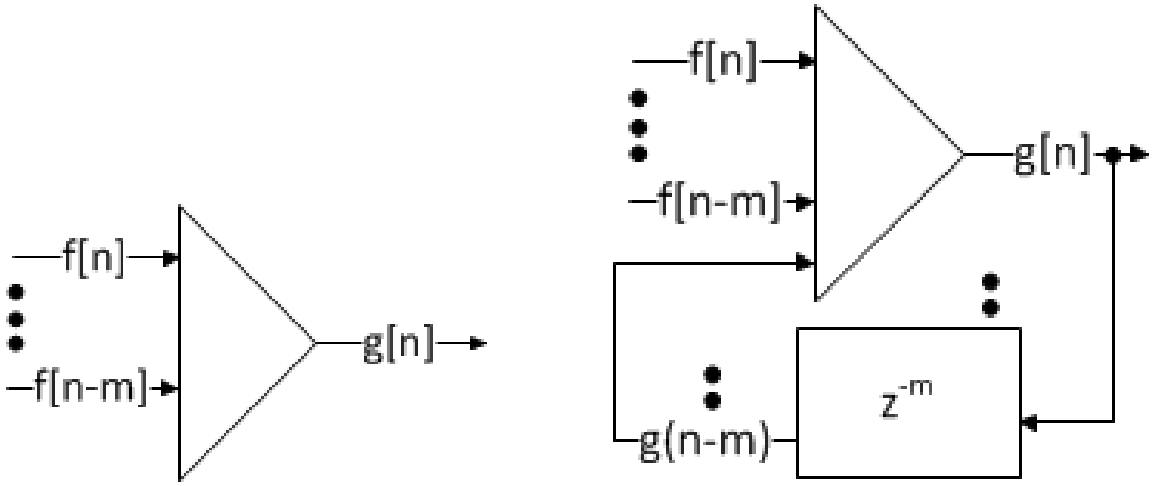
Performing a Discrete Fourier Transform on the delta function yields a flat spectrum with amplitude of 1 and no phase shift for all frequencies.

$$x[\alpha] = \sum_{n=-\infty}^{\infty} \delta[n]\, e^{-j\alpha n} = 1 \qquad \text{for all } \alpha$$

Therefore, if a time domain impulse is applied to the input, the response will be the gain of the system.

5.2 Significance of Finite vs. Infinite

FIR **IIR**



The FIR uses present and past values of the input to produce the output. It does not have memory of its current state or any past states. An IIR also uses present and past values of the input, but also uses past values of the output. This system has memory. Theoretically, the output will continue forever since past values of output are used as input.

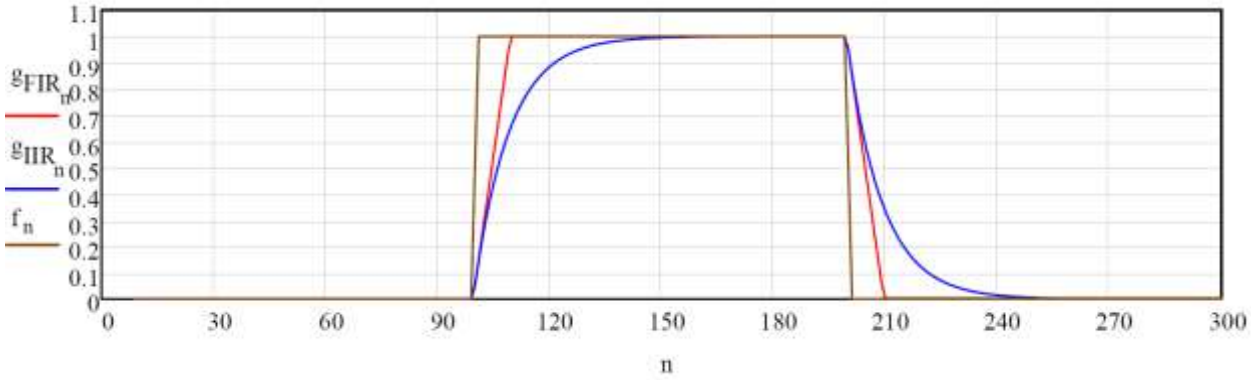
5.3 Examples of FIR and IIR

One example of a FIR is a moving average filter. This is perhaps the easiest digital filter to imagine and implement. Here, the present input value is added to a finite number of past values, then scaled by the number of total samples.

$$g_{FIR_n} = 0.1 \cdot \left(f_n + f_{n-1} + f_{n-2} + f_{n-3} + f_{n-4} + f_{n-5} + f_{n-6} + f_{n-7} + f_{n-8} + f_{n-9} \right)$$

An arguable similar IIR version would be a low pass filter, where the incoming data has the same weight as the FIR example and the remaining weight is allotted to its last state.

$$g_{IIR_n} = 0.1 f_n + 0.9 g_{IIR_{n-1}}$$

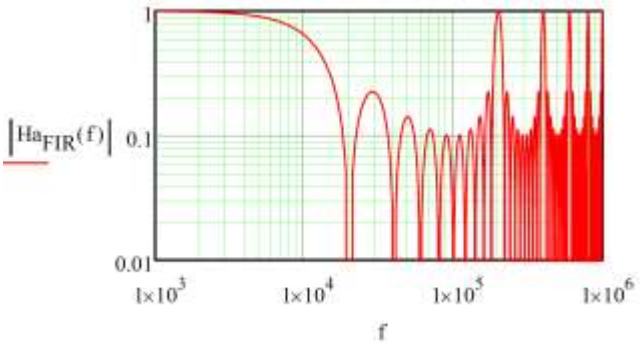


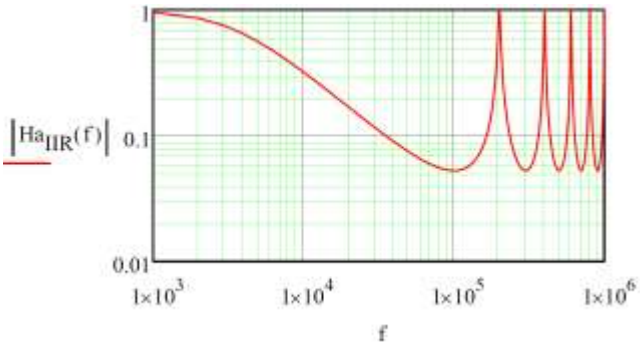
Both filters are excited by the same input pulse. At the start of the pulse the filters have a similar slew rate but they quickly begin to diverge. By 10 samples, the FIR is saturated and at its final value. The IIR with only one memory value and less complex math exponentially approaches its final value.

A sampling rate of 200 kHz is applied to these filters to produce frequency domain graphs.

$$Ha_{FIR}(f) = H_{FIR}\left(e^{j\cdot 2\pi\, f\cdot Hz\cdot T_m}\right)$$

$$Ha_{IIR}(f) = H_{IIR}\left(e^{j\cdot 2\pi\, f\cdot Hz\cdot T_m}\right)$$





6 Short-Cut to LPF (Working in Reverse)

Using the previous example...

$$g_{IIR_n} = 0.1f_n + 0.9g_{IIR_{n-1}}$$

Simply use the equation from section 3.2Single Pole (LPF) Construct

$a_1 = 0.9 = e^{-\alpha T_M} \quad where\ \alpha = 2\pi f_c$

Solve for frequency. $f_c = 3.35kHz$

Done.

6.1 Short-Cut to Create LPF

Example to create a 10 kHz single pole LPF in a system processing math at 200 kHz.

$\alpha = 2\pi f_c = 62{,}832$

The decay factor $a_1 = e^{-\alpha T_M} = 0.73$.
$b_0 = 1 - a_1$

Done.

7 Convergence

IIRs have a feedback term, therefore there is a potential for them to be unstable. In order to launch results out into the world, the stability must be verified..
Luckily, only 2 of the four constructs have a feedback term. The constructs without feedback will always be stable and converge to zero.
The two that do have feedback, have a similar form. The only difference is the value for a_1

$$H(z) = \frac{b_0}{1 - a_1 z^{-1}}$$

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

Looking at the discrete time domain equation above, stable results will be provided when $0 \leq a_1 \leq 1$.

This is seen in how the equation responds to an impulse and how it responds to a sustained value.

After an impulse, g_n will take on a value and f_n will be zero.

If $a_1 = 0$	g_n will become zero instantly.	✔
If $a_1 = 1$	g_n will stay at g_{n-1}indefinitely.	✔
If $0 < a_1 < 1$	g_n will decay exponentially towards zero.	✔
If $1 < a_1$	g_n will increase indefinitely.	✖
If $-1 < a_1 < 0$	g_n will decay, but oscillate between positive and negative values.	✖
If $a_1 = -1$	g_n will oscillate between $-g_{n-1}$ and g_{n-1}.	✖
If $a_1 < -1$	$\lvert g_n \rvert$ will increase indefinitely.	✖

For a converging LPF Construct, an impulse input will converge to zero and be stable. ✔
For any sustained input, the LPF will converge to that input value times b_0. ✔
For a converging Integrator Construct, an impulse settles to a fixed value and holds it. ✔
A sustained value will cause the integrator to increase indefinitely. ✖

Therefore, any combination of constructs will always converge as long as only ONE integrator is used.

8 Conclusion

This paper has introduced a straightforward, construct-based methodology for generating digital signal processing (DSP) coefficients. By leveraging a small set of reusable constructs, the approach simplifies the transition from control design concepts to practical implementation. Through worked examples, the method has been shown to be both accessible and effective, enabling rapid development of stable control structures without the need for extensive symbolic manipulation.

Beyond demonstration, a convergence argument has been presented, ensuring that the method consistently yields correct and stable results under typical design conditions. The constructs not only reduce computational and conceptual overhead but also provide a framework that can be extended to more complex control systems with confidence.