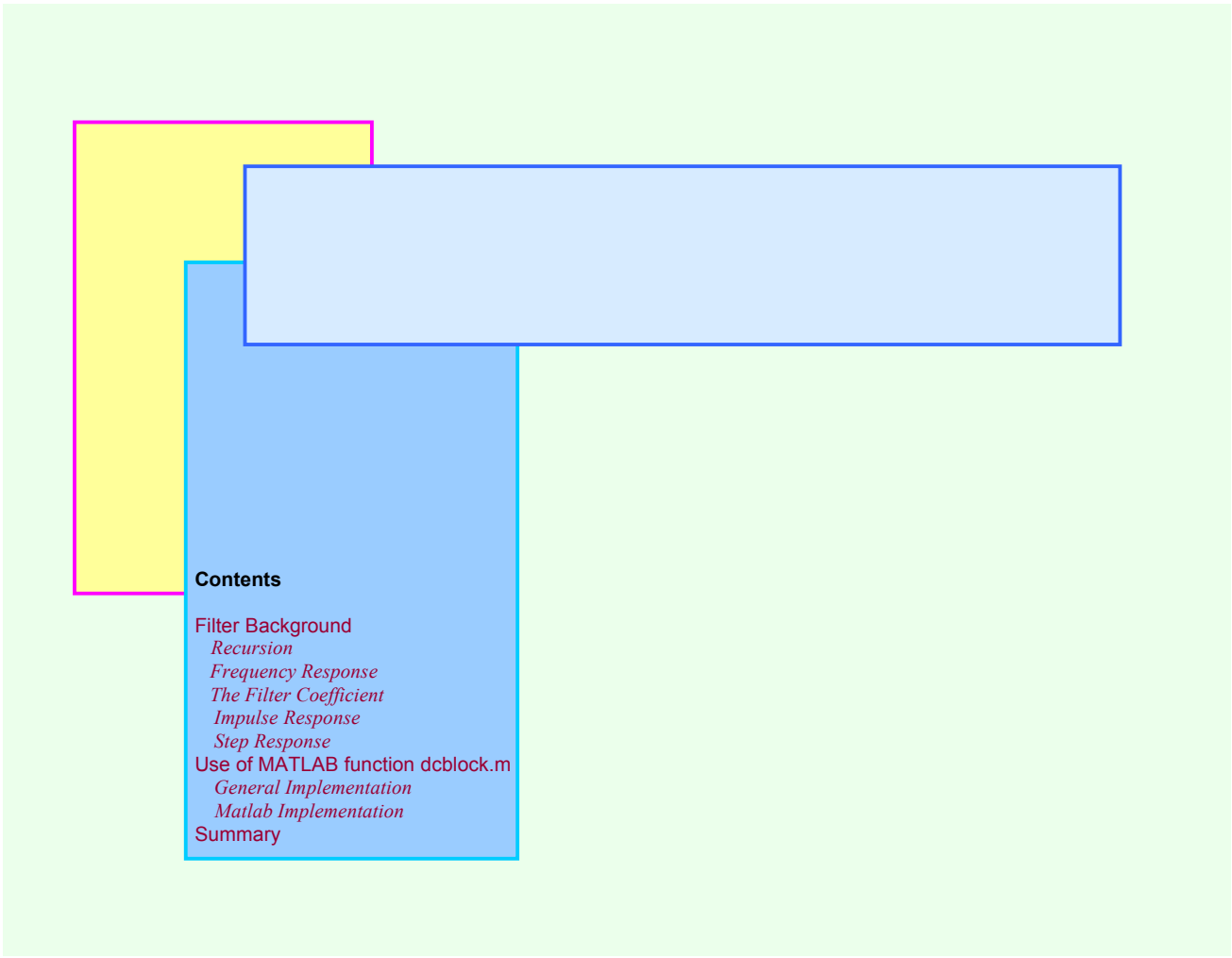

The DC Blocking Filter

J M de Freitas

QinetiQ Ltd, Winfrith Technology Centre, Winfrith, Dorchester DT2 8XJ, England.

Jan 29, 2007

This paper describes the ins and outs of the dc blocking filter, giving the relationship between the lone coefficient a, and filter cut-on frequency f_c, and *vice versa*. General implementation aspects are discussed as well as how to interpret the influence of the filter impulse and step response on the data. A MATLAB function is given for the calculation of the coefficient a. In addition, a table of coefficients is also provided as a quick guide to determine either F_c or a.

Filter Background

The design of high pass filters with cut-on frequencies less than 1% of the Nyquist frequency is a common requirement in many signal processing applications. While this may seem straightforward, if the sampling rate is relatively high in comparison to the required cut-on frequency, standard finite impulse response (FIR) designs lead to very long filter lengths. This in itself is not a drawback, but most DSP systems have a limited amount of memory and so a small number of coefficients is desirable. A particular class of the infinite impulse response (IIR) filters is often preferred for this application because it requires only a single filter coefficient, it is recursive, and can achieve extremely low cut-on frequencies. This is affectionately known as the DC blocking filter.

This paper takes a look at the important relationships governing the dc blocking filter, its applications and typical artefacts seen during implementation.

Recursion

The DC blocking filter is formed by the combination of a digital differentiator and an integrator. The filter is best represented by its difference equation

$$y_t = x_t - x_{t-1} + a y_{t-1} \tag{1}$$

where y_t = current output of the filter
y_{t-1} = previous output of the filter
x_t = current input to the filter
x_{t-1} = previous input to the filter
a = pole of the filter

The first two terms on the right hand side, $(x_t - x_{t-1})$ are the first order backward difference operator, and turns out to be an approximation to the digital differentiator. The last term in (1) is a modification of the digital integrator where the lone filter coefficient a, determines the cut-on frequency. The filter in (1) is recursive because it uses the last output as an input along with two new samples. In this way the filter output contains a measure of every input and is therefore of infinite extent.

Frequency Response

To gain some insight into the frequency response we will use the z-operator approach, whereby $z^{-1} x_t = x_{t-1}$. Rearranging (1) in terms of the z-operator, results in

$$\frac{y_t}{x_t} = \frac{1 - z^{-1}}{1 - a z^{-1}} \tag{2}$$

In (2), the numerator is the operator representation of the first order differentiator; the integrator $[1/(1\text{-}az^{-1})]$ exhibits a pole at a. For stability, the filter coefficient a must be inside the unit circle, i.e. a must be between 0 and 1.

If we substitute $z = e^{j\pi F}$ into (2) and take the modulus, we find that the normalized amplitude and phase transfer functions are

$$H(F) = \frac{(1+a)\sin(\pi F/2)}{\sqrt{1 + a^2 - 2a\cos(\pi F)}} \tag{3a}$$

$$\varphi(F) = \tan^{-1}\left[\frac{(1-a)}{(1+a)}\left(\frac{\sin \pi F}{1 - \cos \pi F}\right)\right] \tag{3b}$$

where F is the normalized frequency and varies between 0 and 1. The normalized frequency is the frequency f (Hz) divided by the Nyquist frequency F_N (Hz) i.e. $F = f/F_N$. Figure 1 shows a plot of the differentiator, integrator and final dc blocking filter formed from the product of these two responses when $a = 0.99$.

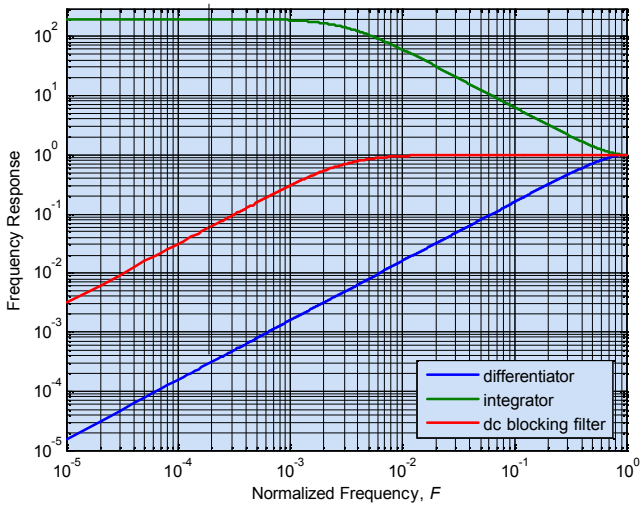

Fig 1. dc blocking/high pass filter represented in terms of its constituent parts. $a = 0.99$.

The filter coefficient

The coefficient a does not need to be chosen at random, or by trial and error. We can determine the value of a by equating $H(F)$ in (3) to ½. The pole turns out to be

$$a = \frac{\sqrt{3} - 2\sin(\pi F_c)}{\sin(\pi F_c) + \sqrt{3}\cos(\pi F_c)} \tag{4}$$

where F_c is the normalized cut-on frequency. Alternatively, if a is known and we want to find the cut-on frequency, then

$$F_c = \frac{1}{\pi}\tan^{-1}\left[\frac{\sqrt{3}(1-a^2)}{1 + 4a + a^2}\right] \tag{5}.$$

A few home truths:

- the normalised cut-on frequency is defined at the –6dB point;
- the coefficient a varies between 0 and 1;
- the cut-on frequency F_c varies between 0 and $F_{c,max} = \tan^{-1}(\sqrt{3})/\pi = 1/3$; this falls out from (5) when $a = 0$, i.e. we have a pure differentiator, and this limits how large we can make F_c; See Figure 2;
- the pass-band of the filter is absolutely flat at the Nyquist frequency;
- the roll-on or slope of the filter is 20dB/decade;
- the phase response φ at F_c is $\pi/3$.
- the phase response roll-off is 20dB/decade beyond F_c. i.e., the phase decreases by a factor of 10 for a 10-fold increase in frequency. Note though, that the phase response at the Nyquist frequency is zero, and this means that for fairly large F_c, the filter behaves more like the differentiator showing an overall linear response.

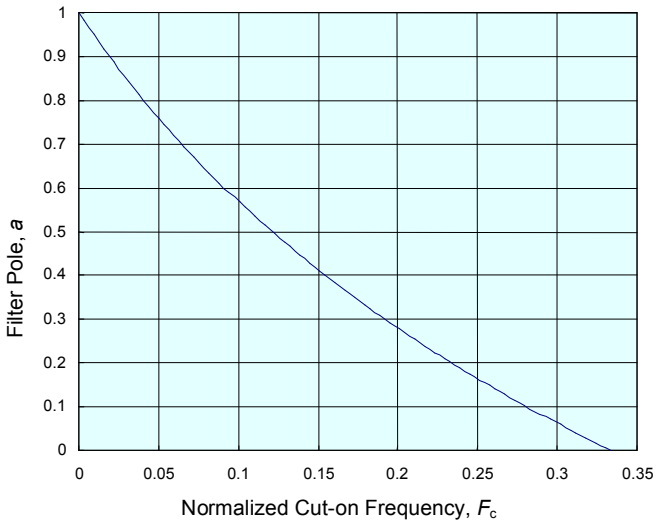

Fig 2. Filter pole a as a function of normalised cut-on frequency.

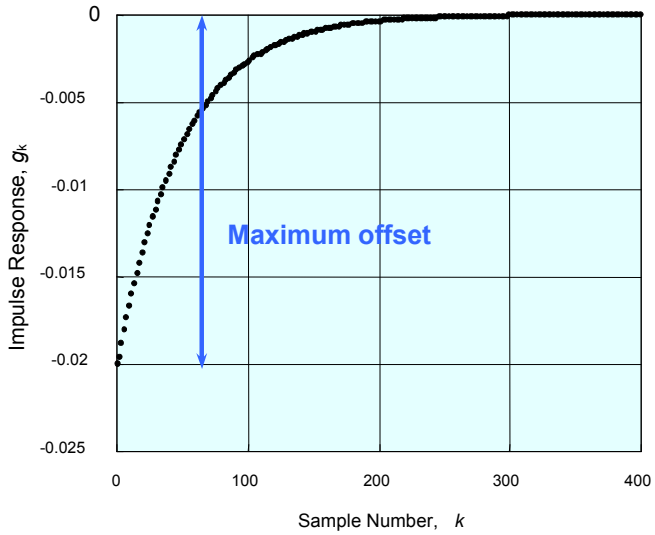

Fig 3. Impulse response for a = 0.98. Note that g_0 has been removed to show decay path more clearly.

Impulse Response

The impulse response is the response of the filter to a spike and gives an idea of how long the filter takes to settle down after an event. The impulse response g_k emerges from a series expansion of (2) so that

$$\frac{1-z^{-1}}{1-az^{-1}} = \sum_{k=0}^{\infty} g_k z^{-k} \qquad (6)$$

where g_k is

$$g_k = \begin{cases} 1 & k = 0 \\ (a-1)a^{k-1} & k \geq 1 \end{cases} \quad (0 < a < 1) \qquad (7)$$

This is all academic stuff, but the practical point of interest here is that the impulse response decays to zero according to $(a-1)a^{k-1}$: the closer a is to 1, i.e. very low cut-on frequencies, the longer the filter takes to stabilize. In fact, the impulse response of the recursive filter given in (1) is dominated by the integrator rather than the differentiator.

The number of samples N, for the response to reach 10^{-m} in absolute value (i.e. $-20m$ dB) is

$$N \approx -\frac{\log(1-a)+m}{\log a} \qquad (a + 10^{-m} \leq 1) \quad (8)$$

Figure 3 shows the impulse response decay for a = 0.98; N turns out to be 148 for m = 3, i.e. an absolute value of 0.001. You would also notice that the maximum initial offset is –2% obtained for k = 1; in fact, for any coefficient a, the offset is $(a-1)100\%$. This occurs every time there is a spike in the data. For larger values of a, the offset is increasingly smaller but then it takes longer for the filter to settle down. For example, if a = 0.999, then $N \approx 2300$ before the filter output is less than an absolute value of 10^{-4}, bearing in mind that from the start, the maximum offset is $|(a-1)|$ = 0.001. This latter observation is important because *the offset level may be perfectly acceptable for the intended application and so there may be no need to wait for the data to settle*. The maximum offset level is certainly one of the things to consider when choosing the value of a.

The other point to note *en passant*, is that if g is truncated to some large number N, say using (8), then the impulse response g_k (k = 0,1,…,N) form the coefficients of an FIR filter. In this case, the FIR filter is not linear phase. Truncated infinite impulses seem more appropriate to the simulation of random processes where error estimates are taken into consideration.

Step Response

The dynamic response of the dc blocking filter to a unit step input[1] is not really different to that of the impulse response. The step decays by the same rate as the impulse response. If the unit step input is denoted by u_k, and the output by U_k, then

$$U_k = e^{-k \ln a} u_k \qquad (9)$$

In practice, if the data suddenly shows a decay pattern, you can be sure there has been a step change in the data. This could be a nuisance sometimes if there is data dropout or loss of data fidelity since it could take a very long time to return to the zero offset level. The number of samples N_s before the data returns to a level below 10^{-m} of the step amplitude is

[1] The unit step input is defined as

$$u_n = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}.$$

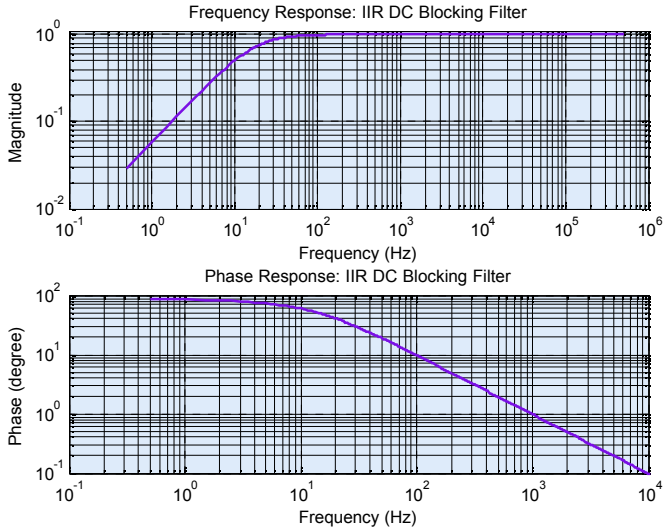

Fig 4. Amplitude and phase response for f_c = 10Hz with sampling frequency f_s = 1MHz, using the plot facility in function dcblock.m.

$$N_s = -\frac{m}{\log a} \qquad (10)$$

To put this into perspective, *the step response decay is far more dramatic than the impulse response because of the amplitude of the step* (usually thousands of times greater than the impulse offset level mentioned earlier). As such the decay of the step response will be visible for a longer time and should be taken into account in any risk assessment of data fidelity.

Use of MATLAB function dcblock.m

The function **dcblock.m** that accompanies this article calculates the filter coefficient a, given the cut-on frequency F_c, and *vice versa*. The main syntax is

a = dcblock(F_c)
a = dcblock(f_c, f_s)

where F_c, f_c and f_s are the normalised, actual cut-on frequency in Hz, and sampling frequency also in Hz. In both function calls we use (4) to calculate a. For example, if the normalised cut-on is 1×10^{-4} then the call **a = dcblock(1E-4)** returns a value of 0.99945600819006 for a when the format is long.

The same function is also used to calculate F_c and f_c when a and f_s are passed to it. In this case,

[F_c , f_c] = dcblock(a, f_s)

and relation (5) is used to do this. If the function call does not include an output parameter, e.g. **dcblock(F_c)**, the function plots the amplitude and phase response of the filter. However, note that if two parameters are passed to the function without an output argument, the plot routine

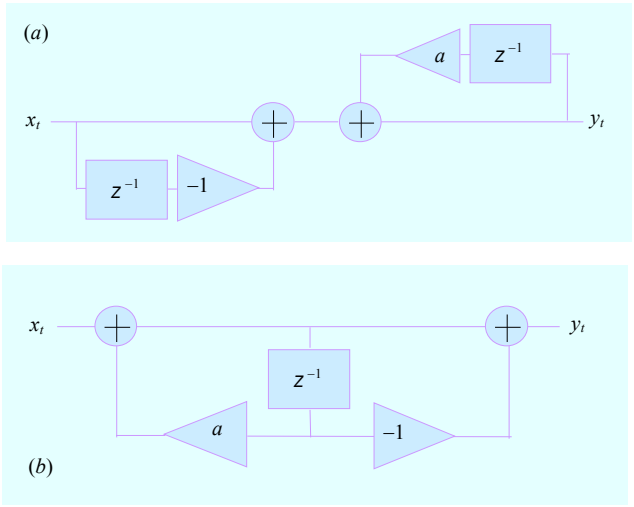

Fig 5. Flow graph of first order IIR DC blocking/high pass filter. Graphs (*a*) and (*b*) give identical output.

always interprets the first as f_c and the second as f_s. Thus **dcblock(a,f_s)** assumes that the cut-on value is a. Further examples and explanation are provided in the help section via the **help dcblock** command. Figure 4 shows the plot output for **dcblock(10,1E6)** corresponding to a cut-on of 10Hz and a sampling frequency of 1MHz.

Implementation

General implementation

There are several ways of implementing the first order IIR DC blocking filter. Some applications may involve the direct coding of the flow graphs, since quite a few DSP environments contain delay, adder and multiplier blocks. Figure 5 shows two configurations of the dc blocking filter; both give the same output.

For the standard coding route (e.g. C, Pascal, Fortran etc), the implementation of (1) in floating point arithmetic is straightforward and follows directly from the recursion. The algorithm below can be implemented in most high and low level programming languages:

$y_{old} = 0$; / initialise previous filter output
$y_{new} = 0$; / initialise current filter output
$x_{old} = 0$; / initialise previous input into filter
begin function
Input: {a / dc blocking filter coefficient
x_j (j = 1,2 …,N) / data to be filtered
y_{old} / initialisation value
x_{old} } / initialisation value
for j = 1 **to** N
$y_{new} = x_j - x_{old} + a*y_{old}$;
$x_{old} = x_j$; / reset current filter input
$x_j = y_{new}$; / current output of filter
$y_{old} = y_{new}$; / reset previous output of filter
end
Output: {x_j ; y_{old}; x_{old}}
end

In this implementation the original array is replaced by the filtered output. If consecutive data blocks are passed to the filter, then y_{old} and x_{old} must be retained as the initialisation values for the next block. Other approaches are also possible.

Matlab implementation

Let us take a quick look at a few implementations in Matlab®[2]; the latter uses a number of approaches, including both time-domain and FFT-based techniques. The Matlab **filter** function is probably the simplest and in this case we need to define the numerator vector **b** and the denominator vector, **a** corresponding to the numerator and denominator in (2). For the dc blocking filter, **b** = **[1 –1]**, and **a** = **[1 –*p*]**, where ***p*** is the dc blocking filter coefficient. If **x** is the input to the filter, the output **y = filter(b,a,x)**. The **[y,zf] = filter(b,a,x,zi)** form of the function should be called when data blocks are used and initialisation values must be passed from one block to the other.

The Matlab script below shows how to implement the dc blocking filter for a noisy Gaussian pulse signal with offset and low frequency background. The sampling frequency is 10kHz and the filter cut-on is 70Hz.

```
t = -0.1:1/10000:0.1-1/10000;    % set up time base
s = sin(2*pi*2.5*t+pi/6)+0.01*randn(size(t))+1.25; % offset & noise
x = gauspuls(t,250,0.6) + s;     % set up noisy Gaussian test pulse
p = dcblock(70,10000);           % get filter coefficient
b = [1 -1];                      % set up differentiator (numerator)
a = [1 -p];                      % set up integrator (denominator)
y = filter(b,a,x);               % filter the test data
plot(t,x,t,y)                    % plot results to screen
legend('Original','DC Filtered','location','northwest');
```

Figure 6 shows the output from this script. Note that after filtering, the Gaussian pulse is no longer symmetric. This happens because of the distortion introduced by the non-linear phase response of the filter. The Gaussian envelope, which is at a lower frequency, sees a different phase to the faster signal inside the envelope. Taken together, this differential phase response explains why the overall shape of the signal is distorted. The differential phase response, or group delay[3], as a function of phase and normalized frequency is simply

$$\tau_g = \frac{\sin 2\varphi}{2\sin \pi F} \quad \text{[number of samples]} \quad (11)$$

The larger the group delay the greater the distortion. The differential phase response for the dc blocking filter is largest for frequencies up to F_c, after which it falls off rapidly. Its effect could still be noticeable up to $10F_c$. On the other hand, the signal inside the envelope at 250Hz, corresponding to 7% of the Nyquist frequency, loses about

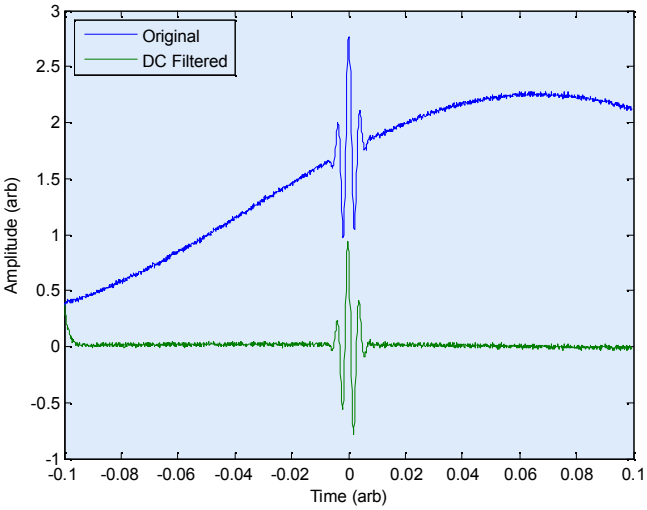

Fig 6. Plot example from the help information in **dcblock.m**. The cut-on was chosen to remove the DC offset and to a large extent the low frequency component.

5% in amplitude through the filter; this is mainly due to the slow roll-on at the transition edge. Note that the cut-on at 70Hz is relatively large at 1.4% of the Nyquist; cut-on frequencies at 0.01% Nyquist are not unusual when settling issues are acceptable. The amount of distortion that can be tolerated is certainly another consideration when applying the dc blocking filter.

It is worthwhile noting that the Matlab interface **fvtool** can also be used to plot the amplitude, phase and impulse response of the filter. This is obtained by returning the command **fvtool(b,a)** at the Matlab prompt, where **b** and **a** are the vectors described above.

Having said so much about distortion, this article would not be complete without focusing on its removal. If the filter is applied both in the forward and backward direction then the distortion is completely removed. The backward run of the filter removes the distortion introduced by the forward run. In Matlab this is implemented by the **filtfilt** function. If you now try the following script,

```
y2 = filtfilt(b,a,x);            % filter the test data with filtfilt
plot(t,x,t,y,t,y2)               % plot new results to screen
legend('Original','DC Filtered', 'Undistorted','location','northwest');
```

you will immediately notice that the Gaussian pulse symmetry has been restored. In spite of this achievement, there will always be a trade-off between data settling time and cut-on frequency; a relatively large cut-on frequency implies a small settling time and *vice versa*. The **filtfilt** implementation completely removes signal distortion and moreover, it attempts to remove the residual dc it sees in the backward direction, but it does not completely remove signal decay that arises from the impulse or step response in the forward direction. In fact, it adds a second decay in the reverse direction.

[2] The Mathworks, Inc, 3 Apple Hill Drive, Natick, MA, USA.

[3] In terms of normalized frequency, the group delay in units of time is defined as $T_g = -\frac{d\varphi}{dF}\frac{1}{\pi f_s} = \frac{\tau_g}{f_s}$.

Summary

We have looked at a relatively simple IIR filter that could be used for DC blocking or other high pass filtering tasks. While the filter implementation is relatively straightforward, we should remain alert to the undesirable effects that the forward nonlinear phase response can have on the filter output, data settling time and initial offset values, bearing in mind that a forward and backward pass of the filter removes signal distortion. A Matlab function **dcblock.m** was presented with a few examples of its use. The said function does not carry out any filtering but merely returns the filter coefficient or plots the filter responses.

Perhaps the best way to summarise our discussion is to give the coefficients *a* for the DC filter in table format for a limited range of frequencies. This could really be useful in re-jigging the filter after an initial test and getting a feel for the number of 9's behind the decimal point, or purely as a reference guide. In fact, most of the 'home truths' are summarised in the amplitude and phase pictographs. The use of the coefficient table below is rather straightforward. The coefficient values *a* are given in terms of multiples of the normalized cut-on base frequency, for example, for a base frequency of 10^{-5} we have *a* being 0.99994559, 0.99989118, and 0.99983677 corresponding to normalized frequencies 1×10^{-5}, 2×10^{-5}, and 3×10^{-5} etc. Further details on usage are provided with the table.

Bibliography

[1] Oppenheim, A V and Schafer, R W. *Discrete-time Signal Processing*. Prentice-Hall. New Jersey. 1999.

[2] De Freitas, J M and Player, M A. Synthesis of the linear stationary stochastic process. *IEE Proc.-F.* Vol. **140**. No.3. 187-190 (1993).

[3] Bristow-Johnson, R. Fixed-Point DC Blocking Filter with Noise Shaping. 2000. www.dspguru.com/comp.dsp/tricks/alg/dc_block.htm

Table of coefficient values *a* for DC Blocking Filter*

F_c	×1	×2	×3	×4	×5	×6	×7	×8	×9
10^{-7}	0.99999946	0.99999891	0.99999837	0.99999782	0.99999728	0.99999674	0.99999619	0.99999565	0.99999510
10^{-6}	0.99999456	0.99998912	0.99998368	0.99997823	0.99997279	0.99996735	0.99996191	0.99995647	0.99995103
10^{-5}	0.99994559	0.99989118	0.99983677	0.99978237	0.99972797	0.99967357	0.99961917	0.99956478	0.99951039
10^{-4}	0.99945601	0.99891231	0.99836891	0.99782581	0.99728300	0.99674048	0.99619826	0.99565633	0.99511470
10^{-3}	0.99457336	0.98917607	0.98380785	0.97846845	0.97315761	0.96787507	0.96262059	0.95739392	0.95219480
10^{-2}	0.94702301	0.89675599	0.84897320	0.80347302	0.76007478	0.71861613	0.67895072	0.64094633	0.60448314
10^{-1}	0.56945231	0.27977278	0.06240347						

**Usage*: To find the value of the coefficient *a* corresponding to normalized cut-on frequency F_c of 0.00004, first locate the row containing 0.00001 i.e. the third row from top in the column labelled F_c, then look under the fourth column (labelled as '×4', i.e. to give 0.00001×4) and read off 0.99978237. The normalized cut-on frequency $F_c = 2f_c/f_s$, where f_c and f_s are the actual cut-on and sampling frequencies in Hz, respectively.

IIR High-Pass/DC Blocking filter:

$$y_t = x_t - x_{t-1} + ay_{t-1}$$

y_t = current output of the filter; y_{t-1} = previous output of the filter; x_t = current input to the filter; x_{t-1} = previous input to the filter; a = pole of the filter