Non-causal Zero Phase FIR Filter With Examples

Cheng-Yang Tan

Accelerator Division/Tevatron

ABSTRACT: This is a quick (but not short) note to see how a non-causal zero phase FIR filter can be implemented with an incoming continuous data stream. Obviously, for non-causal filters to to work, the sampling rate of the incoming stream must be higher than the outgoing stream because non-causal filters must know not only the past, it must also know the future. I will describe three ways to construct the non-causal zero phase FIR filter with examples. I will also show an implementation in *C++* which will use existing FIR filter functions to make the non-causal zero-phase filters.

INTRODUCTION

It is well-known that causal filters cannot be constructed such that the signal at the output of the filter is not phase shifted w.r.t. the input signal. There exists a class of filters called zero-phase filters which phase shifts by 0 or π the input signal. This is the common definition of “zero phase filters” which is not what I thought “zero-phase” means. Most of the arguments and constructions in this note are found on the web and some text books but since they are scattered around I am just collecting the information here. I will not go through many detailed proofs although I will show some proofs to keep things straight. I will assume throughout this paper that the filter is odd in length and symmetric. I will also assume that there is a continuous input stream $\overrightarrow{\boldsymbol{x}}$ that is sampled at a higher frequency than the outgoing stream because non-causal filters must know not only the past but also the future. Thus the output of the zero-phase filter must be connected to another filter, like an averaging filter to down sample the output of the non-causal filter.

There are at least three ways to construct a non-causal filter $H_{NC}(z)$ given a causal FIR filter $H_C(z)$. These three methods are discussed below, and I will show that only methods 0 and 1 are suitable for feedback systems.

Method 0

This is the simplest and most obvious way to construct the the new non-causal filter $H_{NC}(z)$. First, I can write the causal filter $H_C(z)$ as

$$H_C(z) = \sum_{k=0}^{2N} a_k z^{-k} \tag{1}$$

Then $H_{NC}(z)$ is constructed by symmetrizing the coefficients a_k about $k = 0$ rather than at $k = N$, i.e.

$$\left.\begin{aligned} H_{NC}(z) &= \sum_{k=-N}^{N} a_{N-k} z^{-k} \\ &\equiv \sum_{k=-N}^{N} b_k z^{-k} \end{aligned}\right\} \tag{2}$$

where $b_k = a_{N-k}$. H_{NC} is clearly non-causal and it is symmetric about $k = 0$ because $b_{-k} = b_k$. The symmetry is important because zero-phase comes from here. The proof is trivial. In Fourier space, $z \rightarrow e^{i\omega}$

$$\left.\begin{aligned} H_{NC}(e^{i\omega}) &= \sum_{k=-N}^{N} b_k e^{-ik\omega} \\ &= b_0 + \sum_{k=1}^{N} b_k \left[e^{ik\omega} + e^{-ik\omega} \right] \\ &= b_0 + 2 \sum_{k=1}^{N} b_k \cos k\omega \end{aligned}\right\} \tag{3}$$

Clearly, the phase of $H_{NC}(e^{i\omega})$ can be at 0 or π. As long as there are no abrupt phase shifts in the passband of the filter, it can be used in a feedback loop.

Method 1

The second way, which is used by the MATLAB function *filtfilt()*, is to construct the new non-causal filter $H_{NC}(z)$ from $H_C(z)$ by

$$H_{NC}(z) = H_C(1/z) \cdot H_C(z) \tag{4}$$

Clearly $H_C(1/z)$ is non-causal because if

$$\left.\begin{aligned} & H_C(z) = \sum_{k=0}^{2N} a_k z^{-k} \qquad \text{is causal} \\ \text{then} \qquad & H_C(1/z) = \sum_{k=0}^{2N} a_k z^{k} \qquad \text{is non-causal.} \end{aligned}\right\} \tag{5}$$

In Fourier space, $z \rightarrow e^{i\omega}$ and thus

$$\left.\begin{aligned} H_{NC}(e^{i\omega}) &= H_C(e^{-i\omega}) \cdot H_C(e^{i\omega}) \\ &= H_C^*(e^{i\omega}) \cdot H_C(e^{i\omega}) \\ &= |H_C(e^{i\omega})|^2 \end{aligned}\right\} \tag{6}$$

where "*" is the complex-conjugate operator. Clearly H_{NC} is completely real and greater than zero in Fourier space and so any signal going through H_{NC} will have identically zero phase shift. This filter is most useful in feedback loops because it does not introduce any phase shift at all.

Method 2

The third way is which was described in ?? (textbook at 1002)

$$H_{NC}(z) = H_C(1/z) + H_C(z) \tag{7}$$

Now in Fourier space $z \rightarrow e^{i\omega}$ and so

$$H_{NC}(e^{i\omega}) = 2 \times \mathrm{Re}\Big[H_C(e^{i\omega})\Big] \tag{8}$$

Thus in Fourier space H_{NC} is completely real and therefore, the phase shift is either zero or π between the input and the output. Like method 0, this type of filter is certainly *not* zero phase when I really want "zero phase" and is probably not useful in feedback loops because of the sudden phase changes between 0 and π in its phase response.

Proof that $H_C(1/z)$ is equivalent to the operations time reversal, H_C and time reversal

I want to show that the filter $H_C(1/z)$ is equivalent to filtering with $H_C(z)$ of a time reversing the input data and then time reversing one more time the filtered result. (In all the webs sources, this is the algorithm which is described. However, it is actually not clear to me at this time why doing it this way is more efficient than just doing $H_C(1/z)$ straight up. I will, however, indulge the web authors and assume that this is the most efficient way to do things.)

If the input sequence is $\overrightarrow{\boldsymbol{x}} = \{\ldots, x(-1), \underset{\uparrow}{x(0)}, x(1), \ldots, x(2N), \ldots\}$ and $\underset{\uparrow}{x(0)}$ is the zeroth term of the sequence, then the result y_{NC} after going through $H_C(1/z)$ is simply

$$y_{NC}(n) = \sum_{k=0}^{2N} a_k x(n+k) \tag{9}$$

Now, if I time-reverse $\boldsymbol{x}$, i.e. $\overrightarrow{\boldsymbol{x}} \rightarrow \overleftarrow{\boldsymbol{x}} = \{\ldots, x(2N+1), \underset{\uparrow}{x(2N)}, x(2N-1), \ldots, x(1), x(0), x(-1), \ldots\} \equiv \{\ldots, x'(-1), \underset{\uparrow}{x'(0)}, x'(1), \ldots\}$ then the zeroth term of this sequence is $\underset{\uparrow}{x(2N)} = \underset{\uparrow}{x'(0)}$. Passing this time reversed signal into $H_C(z)$ gives

$$\left.\begin{aligned} y_C(m) &= \sum_{k=0}^{2N} a_k x'(m-k) \\ &= y_{NC}(2N-m) \end{aligned}\right\} \tag{10}$$

which is a time-reversed copy of y_{NC}. Thus time reversing $y_C(m)$ will give me back $y_{NC}(n)$.

EXAMPLES

In the following examples, I will suppose that the input data is over sampled and I will process the data in blocks. I will assume that H_C is odd in length and is symmetric. It will become clear that these examples show the way of filtering image data and is not efficient for filtering feedback loop input data that is infinitely long. A realistic implementation for doing non-causal zero phase filtering will be discussed in the section *Realistic Implementation*.

For all these examples, the filter coefficients which I will use is

$$H_C(z) = 1 + 2z^{-1} + z^{-2} \tag{11}$$

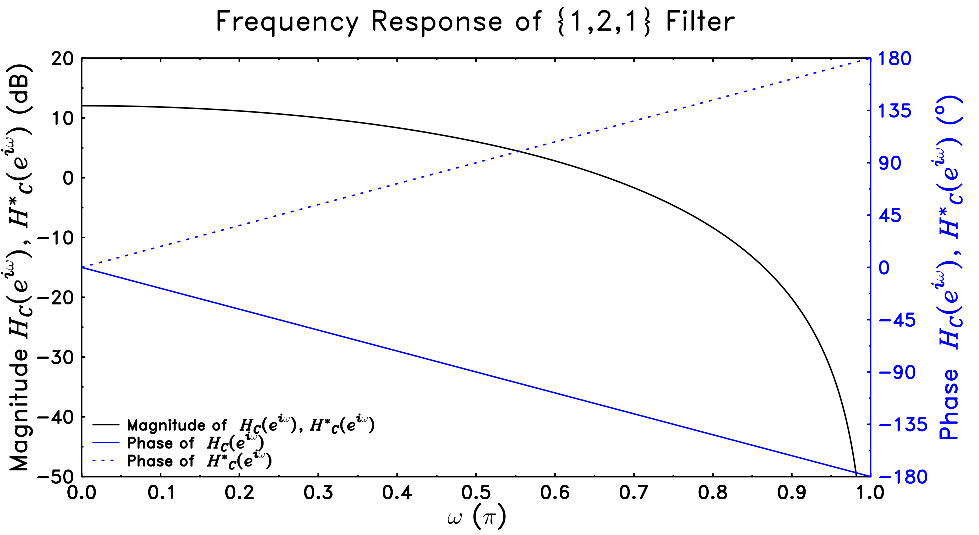

Figure 1 This is the frequency response of H_C and H_C^*. It is clear from the phase response that H_C^* is non-causal.

Method 0

Using method 0, the filter H_{NC} is

$$H_{NC}(z) = z^{-1} + 2 + z \tag{12}$$

The input stream of random numbers is $\overrightarrow{\boldsymbol{x}}$

$$\overrightarrow{\boldsymbol{x}} = \{\ldots, -5, 3, 8, -7, -1, \overbrace{-10, \underbrace{-8, 3, \underset{\uparrow}{2}, -10, -6}_{\text{sub-block } 0}, -9}^{\text{block } 0}, , -9, -7, -3, -9, 3, -6, 0, -10, \ldots\} \tag{13}$$

Block 0 is the block to data that needs to be stored. Sub-block 0 is the data that will be processed and then sent to the next stage for down sampling. Note that the length of block 0 is exactly the length of sub-block 0 plus (length of filter − 1) (the sub-block length must be **odd** for an impulse response calculation which can be used to calculate the filter coefficients. See section *Realistic Implementation*. For this example, I have chosen the length of the sub-block to be 5) . Since the sub-block is odd in length, this means that the block length must also be odd. Block 0 going through $H_C(z)$, gives

$$\left.\begin{aligned} \overrightarrow{y_c}(-2) &= (1 \times -10) + (2 \times -8) + (1 \times 3) = -23 \\ \overrightarrow{y_c}(-1) &= (1 \times -8) + (2 \times 3) + (1 \times 2) = 0 \\ \overrightarrow{y_c}(0) &= (1 \times 3) + (2 \times 2) + (1 \times -10) = -3 \\ \overrightarrow{y_c}(1) &= (1 \times 2) + (2 \times -10) + (1 \times -6) = -24 \\ \overrightarrow{y_c}(2) &= (1 \times -10) + (2 \times -6) + (1 \times -9) = -31 \end{aligned}\right\} \tag{14}$$

The result is $\overrightarrow{\boldsymbol{y_c}} = \{-23, 0, \underset{\uparrow}{-3}, -24, -31\}$.

The next block to process is block 1

$$\overrightarrow{\boldsymbol{x}} = \{\ldots, -5, 3, 8, -7, -1, -10, -8, 3, \underset{\uparrow}{2}, -10, \overbrace{-6, \underbrace{-9, -9, -7, -3, -9}_{\text{sub-block } 1}, 3}^{\text{block } 1}, -6, 0, -10, \ldots\} \tag{15}$$

and the result of going through H_{NC} is $\overrightarrow{\boldsymbol{y_c}} = \{-33, -34, -26, -22, -18\}$.

Method 1

For this method I will use the filter $H_{NC} = H_C(1/z) \cdot H_C(z)$. H_{NC} can be interpreted as first taking the input signal $\overrightarrow{\boldsymbol{x}}$ and sending into H_C to get the result $\overrightarrow{\boldsymbol{y_c}}$. Second , I use the result from the previous section that $H_C(1/z)$ is equivalent to time reversing $\overrightarrow{\boldsymbol{y_c}}$ to give $\overleftarrow{\boldsymbol{y_c}}$ and sending it through H_C again to get $\overleftarrow{\boldsymbol{y_c'}}$ and finally, time reversing one more time to get the result $\overrightarrow{\boldsymbol{y}}$. (Whew!)

Again, I will use the same stream of random numbers $\overrightarrow{\boldsymbol{x}}$

$$\overrightarrow{\boldsymbol{x}} = \{\ldots, -5, 3, 8, -7, \overbrace{-1, -10, \underbrace{-8, 3, \underset{\uparrow}{2}, -10, -6}_{\text{sub-block 0}}, -9, -9}^{\text{block 0}}, -7, -3, -9, 3, -6, 0, -10, \ldots\} \tag{16}$$

Block 0 going through $H_C(z)$, gives

$$\left.\begin{aligned}
\overrightarrow{y_c}(-2) &= (1 \times -1) + (2 \times -10) + (1 \times -8) = -29 \\
\overrightarrow{y_c}(-1) &= (1 \times -10) + (2 \times -8) + (1 \times 3) = -23 \\
\overrightarrow{y_c}(0) &= (1 \times -8) + (2 \times 3) + (1 \times 2) = 0 \\
&\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
\overrightarrow{y_c}(4) &= (1 \times -9) + (2 \times -9) + (1 \times -6) = -33
\end{aligned}\right\} \tag{17}$$

Note that I did not bother calculating $\overrightarrow{y_c}(-4)$ and $\overrightarrow{y_c}(-3)$ because they will not be used. The result $\overrightarrow{\boldsymbol{y_c}} = \{-29, -23, \underset{\uparrow}{0}, -3, -24, -31, -33\}$ is time reversed to give

$$\overleftarrow{\boldsymbol{y_c}} = \{-33, -31, -24, -3, \underset{\uparrow}{0}, -23, -29\} \tag{18}$$

Taking $\overleftarrow{y_c}$ and sending it through $H_C(z)$ one more time and not calculating $\overleftarrow{y_c}(-4)$ and $\overleftarrow{y_c}(-3)$ gives

$$\overleftarrow{\boldsymbol{y_c'}} = \{-119, -82, \underset{\uparrow}{-30}, -26, -75\} \tag{19}$$

which has exactly the same length as sub-block 0. Time reversing one more time, gives the result of sending sub-block 0 through the zero phase filter $H_{NC}(z)$

$$\overrightarrow{\boldsymbol{y}_{\boldsymbol{c}}'} = \{-75, -26, \underset{\uparrow}{-30}, -82, -119\} \tag{20}$$

The entire procedure for processing block 0 is summarized in Table 1. I have also worked out block 1

$$\overrightarrow{\boldsymbol{x}} = \{\ldots, -5, 3, 8, -7, -1, -10, -8, 3, \underset{\uparrow}{2}, -10, -6, \overbrace{-9, -9, \underbrace{-7, -3, -9, 3, -6}_{\text{sub-block 1}}, 0, -10}^{\text{block 1}}, \ldots\} \tag{21}$$

in Table 2.

Table 1. Processing Block 0 with Method 1

$\boldsymbol{n}$	-4	-3	-2	-1	0	1	2	3	4
$\overrightarrow{\boldsymbol{x}}$	-1	-10	-8	3	2	-10	-6	-9	-9
$\overrightarrow{\boldsymbol{y_c}}$	$*$	$*$	-29	-23	0	-3	-24	-31	-33
$\overleftarrow{\boldsymbol{y_c}}$	-33	-31	-24	-3	0	-23	-29	$*$	$*$
$\overleftarrow{\boldsymbol{y}_{\boldsymbol{c}}'}$	$*$	$*$	-119	-82	-30	-26	-75	$*$	$*$
$\overrightarrow{\boldsymbol{y}_{\boldsymbol{c}}'}$	$*$	$*$	-75	-26	-30	-82	-119	$*$	$*$

Table 2. Processing Block 1 with Method 1

n	3	4	5	6	7	8	9	10	11
$\overrightarrow{\boldsymbol{x}}$	−9	−9	−7	−3	−9	3	−6	0	−10
$\overrightarrow{\boldsymbol{y_c}}$	*	*	−34	−26	−22	−18	−9	−9	−16
$\overleftarrow{\boldsymbol{y_c}}$	−16	−9	−9	−18	−22	−26	−34	*	*
$\overleftarrow{\boldsymbol{y'_c}}$	*	*	−43	−45	−67	−88	−108	*	*
$\overrightarrow{\boldsymbol{y'_c}}$	*	*	−108	−88	−67	−45	−43	*	*

Method 2

In method 2, I process $\overrightarrow{\boldsymbol{x}}$ once with $H_C(z)$ to get $\overrightarrow{\boldsymbol{y_c}}$. Then I process $\overrightarrow{\boldsymbol{x}}$ with $H_C(1/z)$ using the time reversal, $H_C(z)$, time reversal method as before to get $\overrightarrow{\boldsymbol{y'_c}}$. The output of $H_{NC}(z)$ is the sum $\overrightarrow{\boldsymbol{y}} = \overrightarrow{\boldsymbol{y_c}} + \overrightarrow{\boldsymbol{y'_c}}$.

I start with the same data stream $\overrightarrow{\boldsymbol{x}}$ and partition the stream into blocks like I did before

$$\overrightarrow{\boldsymbol{x}} = \{\ldots, -5, 3, 8, -7, \overbrace{-1, -10, \underbrace{-8, 3, \underset{\uparrow}{2}, -10, -6}_{\text{sub-block 0}}, -9, -9}^{\text{block 0}}, -7, -3, -9, 3, -6, 0, -10, \ldots\} \tag{22}$$

Block 0 going through $H_C(z)$ gives $\overrightarrow{\boldsymbol{y_c}}$

$$\overrightarrow{\boldsymbol{y_c}} = \{-29, -23, \underset{\uparrow}{0}, -3, -24\} \tag{23}$$

Next, to calculate $\overrightarrow{\boldsymbol{y'_c}}$, I have to time reverse $\overrightarrow{\boldsymbol{x}}$

$$\overleftarrow{\boldsymbol{x}} = \{\ldots, -9, -9, -6, -10, \underset{\uparrow}{2}, 3, -8, -10, -1, \ldots\} \tag{24}$$

Go through $H_C(z)$

$$\overleftarrow{\boldsymbol{y_c'}} = \{-33, -31, \underset{\uparrow}{-24}, -3, 0\} \tag{25}$$

Time reverse $\overleftarrow{\boldsymbol{y_c'}}$

$$\overrightarrow{\boldsymbol{y_c'}} = \{0, -3, \underset{\uparrow}{-24}, -31, -33\} \tag{26}$$

Finally, the result of going through $H_{NC}(z)$ is the sum

$$\begin{aligned}\overrightarrow{\boldsymbol{y}} &= \overrightarrow{\boldsymbol{y_c}} + \overrightarrow{\boldsymbol{y_c'}} \\ &= \{-29, -26, \underset{\uparrow}{-24}, -34, -57\}\end{aligned} \tag{27}$$

The entire process shown above is summarized in Table 3. Processing of block 1 is shown in Table 4.

Table 3. Processing Block 0 with Method 2

$\boldsymbol{n}$	−4	−3	−2	−1	0	1	2	3	4
$\overrightarrow{\boldsymbol{x}}$	−1	−10	−8	3	2	−10	−6	−9	−9
$\overrightarrow{\boldsymbol{y_c}}$	*	*	−29	−23	0	−3	−24	*	*
$\overleftarrow{\boldsymbol{x}}$	−9	−9	−6	−10	2	3	−8	−10	−1
$\overleftarrow{\boldsymbol{y_c'}}$	*	*	−33	−31	−24	−3	0	*	*
$\overrightarrow{\boldsymbol{y_c'}}$	*	*	0	−3	−24	−31	−33	*	*
$\overrightarrow{\boldsymbol{y}}$	*	*	−29	−26	−24	−34	−57	*	*

Table 4. Processing Block 1 with Method 2

$\boldsymbol{n}$	3	4	5	6	7	8	9	10	11
$\overrightarrow{\boldsymbol{x}}$	-9	-9	-7	-3	-9	3	-6	0	-10
$\overrightarrow{\boldsymbol{y_c}}$	*	*	-34	-26	-22	-18	-9	*	*
$\overleftarrow{\boldsymbol{x}}$	-10	0	-6	3	-9	-3	-7	-9	-9
$\overleftarrow{\boldsymbol{y_c'}}$	*	*	-16	-9	-9	-18	-22	*	*
$\overrightarrow{\boldsymbol{y_c'}}$	*	*	-22	-18	-9	-9	-16	*	*
$\overrightarrow{\boldsymbol{y}}$	*	*	-56	-44	-31	-27	-25	*	*

REALISTIC IMPLEMENTATION

The previous examples in section *Examples* illustrate the way how not to implement zero-phase filtering in any realistic feedback loop signal processing setup. The recipe for a realistic implementation of non-causal filtering is as follows:

(*i*) Calculate the causal filter coefficients of H_C using any of the standard packages available on the web.

(*ii*) If I cloose to calculate non-causal filter coefficients using the methods shown in *Examples* then the sub-block size must be odd in length. The block length is (sub-block length + filter length − 1) for method 0 and sub-block length + 2 × (filter length − 1) for methods 1 and 2. Otherwise I skip to (*iiia*).

(*iii*) The impulse response of H_{NC} is calculated using the examples shown previously. For example $\overrightarrow{\boldsymbol{x}} = \{0, 0, 0, 0, \underset{\uparrow}{1}, 0, 0, 0, 0\}$ is the input data block for methods 1 and 2. Method 0, of course, has the same coefficients as H_C.

(*iiia*) Or more directly, $H_{NC}(z)$ is calculated using the formulas (4) for method 1 and (7) for method 2. The coefficients a_k become the filter coefficients of H_{NC}.

(*iv*) Use the impulse response as the filter coefficients of an FIR filter which is used to process the data blocks.

(*v*) Once I have the filter coefficients, the block size is no longer constrained to be odd. An addition of a simple memory manipulation is all that is needed to existing FIR filter functions to implement non-causal filtering.

So following the recipe, in step (*i*), I have calculated an 11 tap FIR low pass filter using a Hamming window with a cut-off frequency at $\pi/8$. Its coefficients are shown in Table 5 and its frequency response and impulse response are shown in Figures 2 and 3.

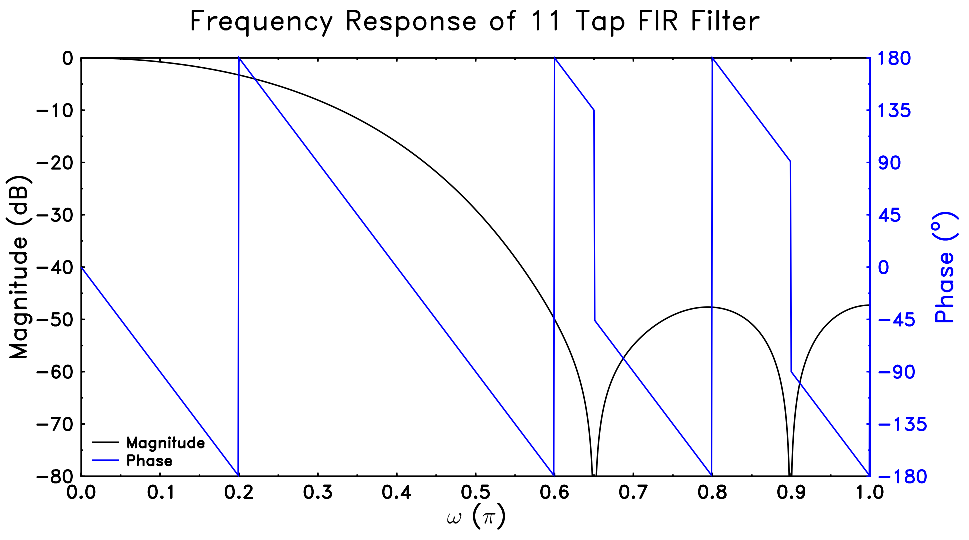

Figure 2 This is the frequency response of the 11 point FIR low pass filter.

Table 5. 11 tap FIR low pass filter

k	a_k	k	a_k
0	−0.0038713	10	−.0038713
1	0.0000000	9	0.0000000
2	0.0320878	8	0.320878
3	0.1167086	7	0.1167086
4	0.2207012	6	0.2207012
5	0.2687474		

In step (*ii*), I have chosen the length of the sub-block to be 13, and so with this 11 tap FIR filter, the length of each block is $13+(11-1) = 23$ for method 0 and $13+2\times(11-1) = 33$ for methods 1 and 2. Step (*iii*) in the recipe is continued in the following subsections.

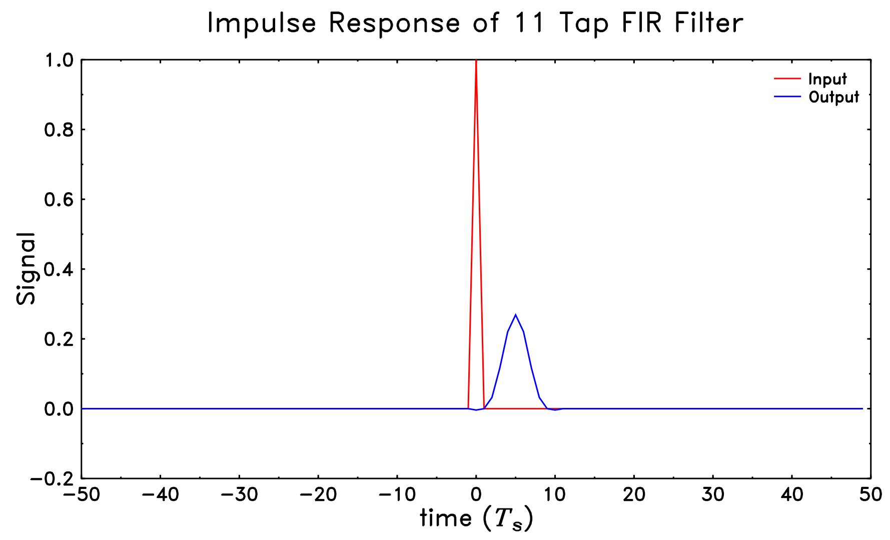

Figure 3 The impulse response of the 11 point FIR filter. Notice that the delay between the input and the output is $(11 - 2)/2 = 5$ samples as expected. The sampling time is T_s.

Method 0 in Step (*iii*) or (*iiia*)

If I use step (*iii*) for method 0, then the impulse response of H_{NC} is calculated with the procedure shown in the examples with $\overrightarrow{\boldsymbol{x}} = \{\ldots, 0, \underset{\uparrow}{1}, 0 \ldots\}$ where the length of $\overrightarrow{\boldsymbol{x}}$ is 23 and the "1" is the 12'th element (the first element is numbered 1) of $\overrightarrow{\boldsymbol{x}}$. Or if I use step (*iiia*), then the coefficients are just renumberd like in (2).

The frequency response is shown in Figure 4. The impulse response is shown in Figure 5 and the filter coefficients of H_{NC} are shown in Table 6. Note that the coefficients are the same as those in Table 5 but with k different.

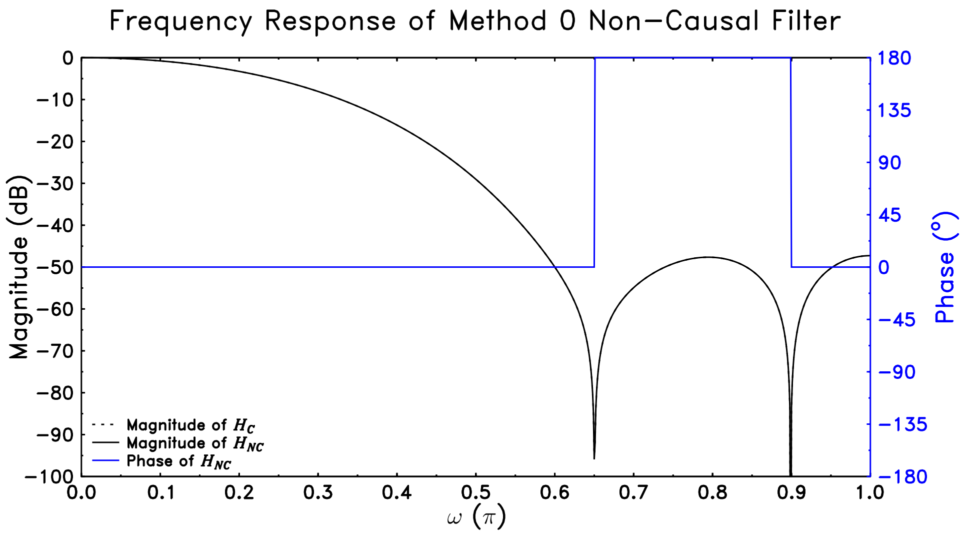

Figure 4 This is the frequency response H_{NC} of the filter constructed using method 0. The phase has discontinuities outside the passband. The magnitude response of the 11 point FIR filter H_C is identical to the magnitude response H_{NC}.

Table 6. Method 0 filter coefficients

k	a_k	k	a_k
−5	−0.0038713	5	−.0038713
−4	0.0000000	4	0.0000000
−3	0.0320878	3	0.320878
−2	0.1167086	2	0.1167086
−1	0.2207012	1	0.2207012
0	0.2687474		

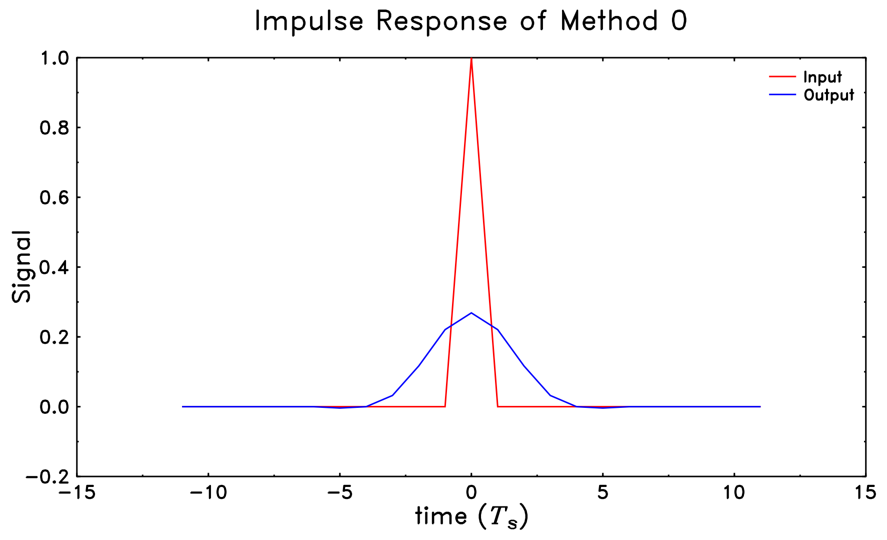

Figure 5 The impulse response of the non-causal filter using method 0. Notice that the response is symmetric about zero and there is zero delay between the input and the output.

Method 1 in Step (*iii*) and (*iiia*)

If I chosse step (*iii*) for method 1, then the impulse response of H_{NC} is calculated with the procedure shown in the examples with $\overrightarrow{\boldsymbol{x}} = \{\ldots, 0, \underset{\uparrow}{1}, 0 \ldots\}$ where the length of $\overrightarrow{\boldsymbol{x}}$ is 33 and the "1" is the 17'th element (the first element is numbered 1) of $\overrightarrow{\boldsymbol{x}}$. Or If I choose step (*iiia*), I just multiply out $H_C(1/z) \cdot H_C(z)$ in Mathematica and get the same solution as step (*iii*).

The frequency response of this filter is shown in Figure 6. The impulse response is shown in Figure 7 and the filter coefficients of H_{NC} are shown in Table 7. Notice that the number of coefficients has increased from 11 (which is the length of H_C) to 21.

Table 7. Method 1 filter coefficients

k	a_k	k	a_k
-10	1.4987×10^{-5}	10	1.4987×10^{-5}
-9	0.0000000	9	0.0000000
-8	-0.000248443	8	-0.000248443
-7	-0.00090363	7	-0.00090363
-6	-0.000679173	6	-0.000679173
-5	0.00540906	5	0.00540906
-4	0.0260758	4	0.0260758
-3	0.0678589	3	0.0678589
-2	0.125354	2	0.125354
-1	0.177631	1	0.177631
0	0.198974		

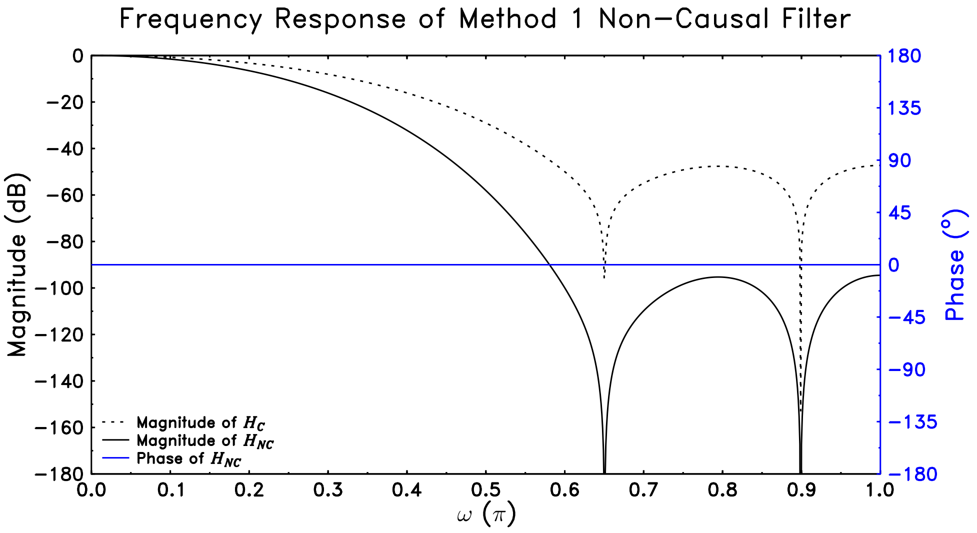

Figure 6 This is the frequency response H_{NC} of the filter constructed using method 1. Clearly the phase is zero in the entire bandwidth of the filter. The original magnitude response of the 11 point FIR filter H_C is also plotted here for comparison.

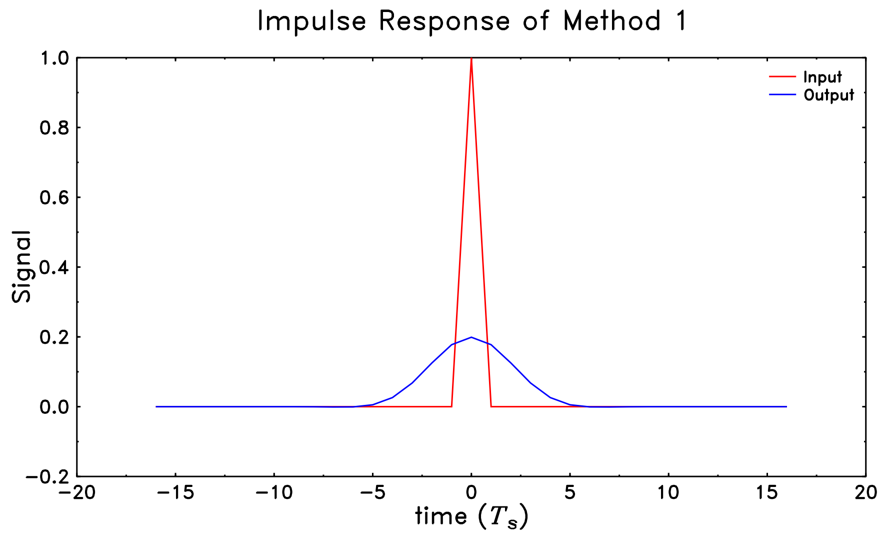

Figure 7 The impulse response of the non-causal filter using method 1. Notice that the response is symmetric about zero and there is zero delay between the input and the output.

Method 2 in Step (*iii*) and (*iiia*)

If I choose step (*iii*) for method 2, then the impulse response of H_{NC} is calculated with the procedure shown in the examples with $\overrightarrow{\boldsymbol{x}} = \{\ldots, 0, \underset{\uparrow}{1}, 0\ldots\}$ where the length of $\overrightarrow{\boldsymbol{x}}$ is 33 and the "1" is the 17'th element (the first element is numbered 1) of $\overrightarrow{\boldsymbol{x}}$. Or If I choose step (*iiia*), I just add $H_C(1/z) + H_C(z)$ in Mathematica and get the same solution as step (*iii*).

The frequency response of this filter is shown in Figure 8. Looking at the frequency response, it becomes clear that H_{NC} is drastically different from H_C and so this method is probably not the best way to implement zero-phase filters. For completeness, I have calculated the impulse response which is shown in Figure 9 and the its filter coefficients in Table 8.

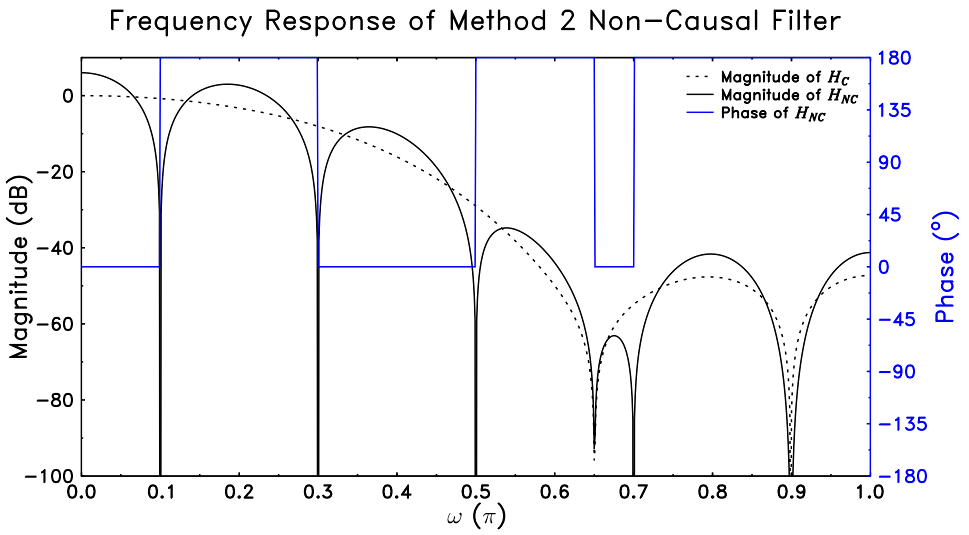

Figure 8 This is the frequency response H_{NC} of the filter constructed using method 2. Notice the increased number of notches in the magnitude response as well as the sudden changes in the phase response between 0 and 180° of H_{NC}.

Table 8. Method 2 filter coefficients

k	a_k	k	a_k
−10	−0.0038713	10	−0.0038713
−9	0.0000000	9	0.0000000
−8	0.0320878	8	0.0320878
−7	0.116709	7	0.116709
−6	0.220701	6	0.220701
−5	0.268747	5	0.268747
−4	0.220701	4	0.220701
−3	0.116709	3	0.116709
−2	0.0320878	2	0.0320878
−1	0.0000000	1	0.0000000
0	−0.0077426		

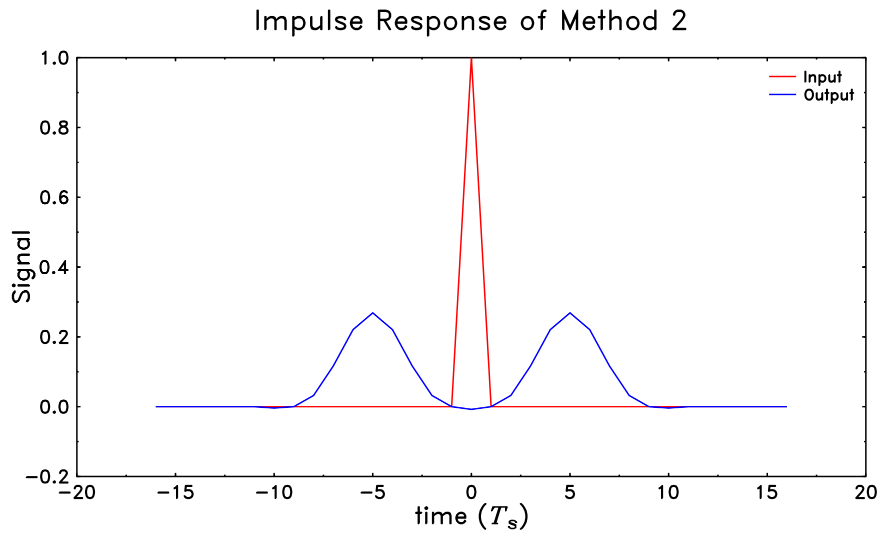

Figure 9 The impulse response of the non-causal filter using method 2. Although the respose is symmetric about zero, its response at zero is essentially zero.

Step (*iv*)

For step (*iv*), I will just take one set of filter coefficients from Table 6, 7 or 8 and use them in a *C++* programme. See Listing 1.

Listing 1 C++ Partial Source Listing

```
#define SUBBLOCK_SIZE 22 // for method 0, and 12 for methods 1 and 2
void fir(double * const x, const int Nx,
   const double * const h, const int Nh,
   double* r)
{
/*
   x:    input data
   Nx:  length of input data
   h:    filter coefficients
   Nh:  length of the filter
   r:  the result after filtering
*/
...
}
int main()
{
    const double h[] = {...}; // the filter coefficients
    const int FILTER_LEN = sizeof(h)/sizeof(double);
    const int HALF_FILTER_LEN = (FILTER_LEN-1)/2;
    const int BLOCK_SIZE = SUBBLOCK_SIZE + (FILTER_LEN-1);
    double x[BLOCK_SIZE]; // input data
    double r[BLOCK_SIZE]; // result
    int i = HALF_FILTER_LEN;

    memset(x, 0, sizeof(double)*BLOCK_SIZE); // init x
    while(cin >> x[i]){
       if(++i >= BLOCK_SIZE){
          // perform FIR filtering
          fir(x, BLOCK_SIZE, h, FILTER_LEN, r);

          // make result non-causal by shifting result to the left
          // so that only array values from 0 to SUBBLOCK_SIZE-1
          // are valid
          memmove(r, r+FILTER_LEN-1, SUBBLOCK_SIZE*sizeof(double));

          // show the filtered result
          for(int j=0; j < SUBBLOCK_SIZE; j++){
                cout << r[j] << "\n";
          }
          // copy the old x values at the end to the beginning of x
          memmove(x, x+(BLOCK_SIZE-FILTER_LEN+1),
                            (FILTER_LEN-1)*sizeof(double));
          // reset counter
          i=FILTER_LEN-1;
       }
     }
```

The filter coefficents go into `h[]`. The data is read from stdin into `x[]` starting from array position `FILTER_LEN-1`. The number of data points read in is `(BLOCK_SIZE - (FILTER_LEN -1))` (after the first read). Previous data is left in `x[0,...,FILTER_LEN-2]` so that the final length of `x[]` is `BLOCK_SIZE`. A call to an existing FIR filter function `fir()` is used to process `x[]`. The result `r[]` is causal and to make it non-causal, the data points in `r[]` must be shifted to the left by `(FILTER_LEN-1)`. Once this is done, only elements `r[0]` to `r[SUBBLOCK_SIZE-1]` contain valid filtered data. The old data from `x[BLOCK_SIZE-FILTER_LEN-1]` to `x[BLOCK_SIZE-1]` are copied to the beginning of `x[]`. The new set of input data is stuffed into `x[]` starting from array position `FILTER_LEN-1`.

Step (v)

In Step (v), I will demonstrate non-causal filtering with the simple *C++* programme shown in Listing 1. I will filter the following incoming stream of data which I choose to be

$$\overrightarrow{x}(n) = \cos\left(\frac{\pi}{16}n\right) + \frac{1}{10}\sin\left(\frac{9\pi}{10}n\right) \qquad \text{where } n \in \mathbb{N} \cup \{0\} \tag{28}$$

The sub-block size I have chosen for method 0 is 22 and so the block size is $22+(11-1) = 32$. For methods 1 and 2 the sub-block size is 12, so that the block size is $12 + (21 - 1) = 32$.

First the result of filtering with H_C using the coefficients from Table 5 is shown in Figure 10. It is clear that the filtered signal is delayed w.r.t. noisy input signal.

The result of filtering with the non-causal filter H_{NC} which comes from method 0 using the coefficients from Table 6 is shown in Figure 12. There is no delay between the input and the output. Note that the output is simply the blue signal of Figure 10 shifted to the left by 5 samples.

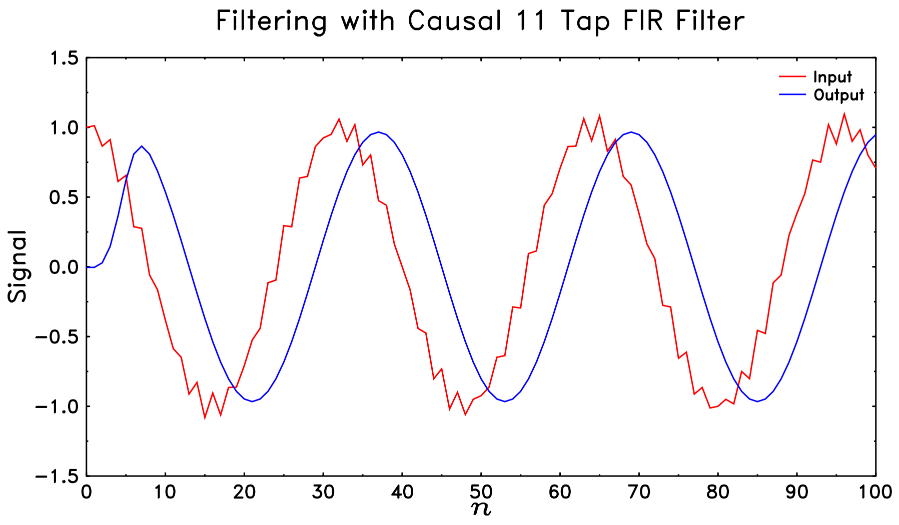

Figure 10 The noisy signal (red) is input into the FIR filter with coefficients from Table 5. The output (blue) is clearly delayed w.r.t. noisy input signal.

Figures 12 and 13 show the result of filtering the same noisy input with the filter coefficients from Tables 7 and 8 respectively. Clearly there is no delay between the input and the output. Note that there is a transient right at the start in all the methods, but since the input signal is continuously coming in so this should not be a problem.

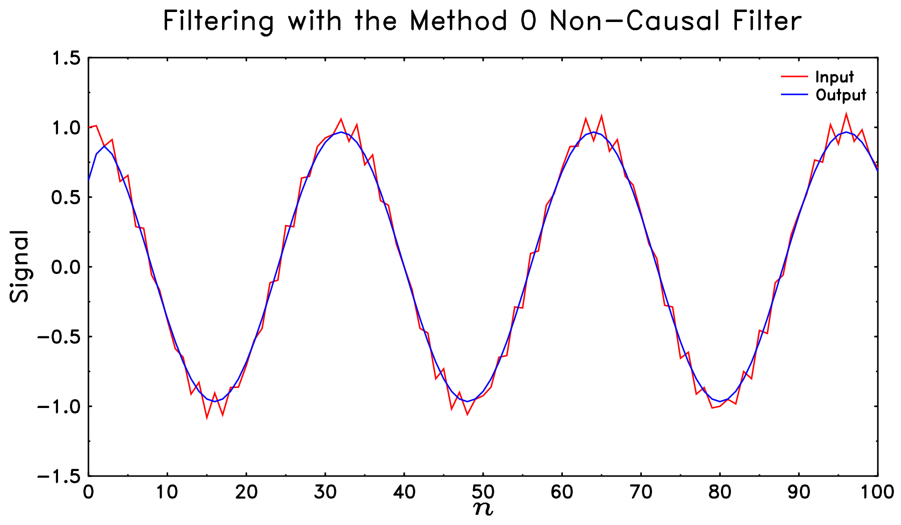

Figure 11 The noisy signal (red) is filtered with the non-causal filter using method 0 with coefficients from Table 6. There is no delay between the input and the output.

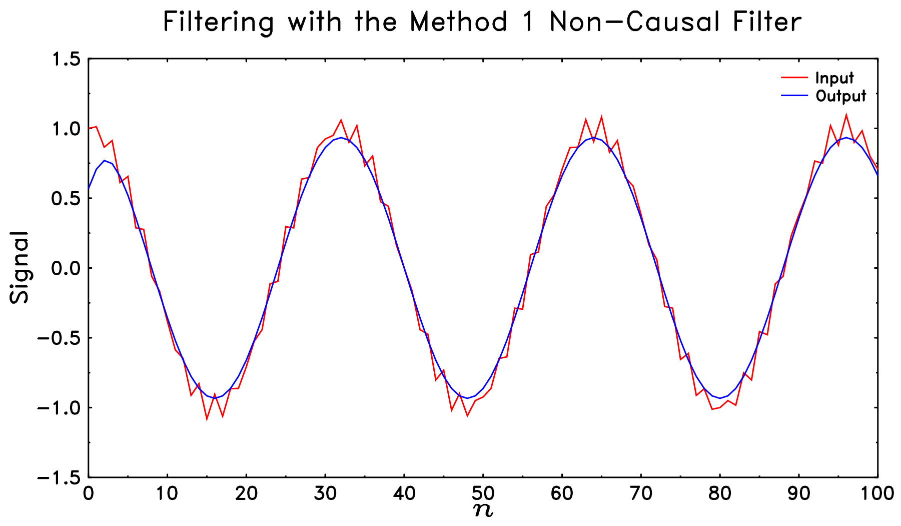

Figure 12 The noisy signal (red) is filtered with the non-causal filter using method 1 with coefficients from Table 7. Again, there is no delay between the input and the output.

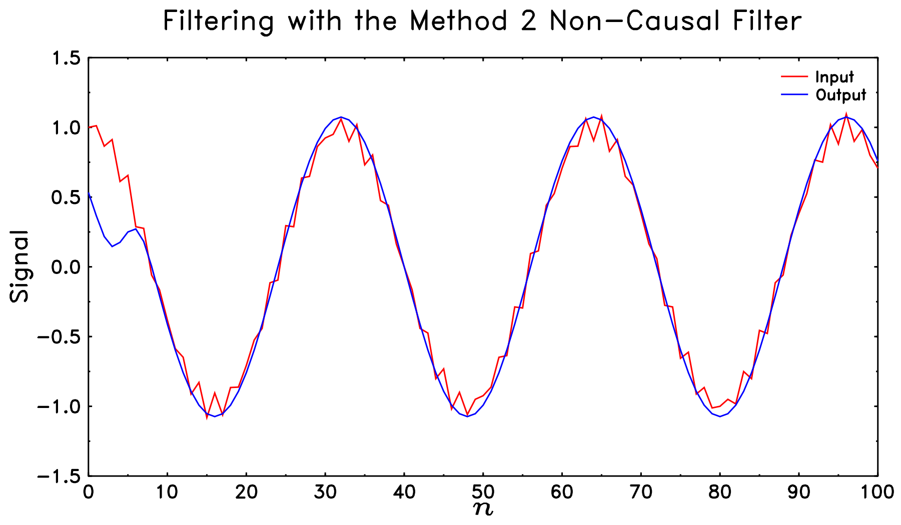

Figure 13 The noisy signal (red) is filtered with the non-causal filter using method 2 with coefficients from Table 8. Again, there is no delay between the input and the output.

CONCLUSION

I have shown three methods for calculating the non-causal zero phase filter. The *C++* code which I use is one pssible way of implementing this filter using existing FIR filter functions. There is a way to make the code go a little faster by applying the filter coefficients forwards in time rather than backwards which I have used. Whether going forwards or backwards in time, the data points in the data block `x[]` must still be shifted to keep the block coherent for the next set of data points.