Solutions

The basic mathematical expression of a simple linear filter is

\[ y_t = x_t * f_t \]

where

- \( y_t \) = output of filter
- \( x_t \) = input to filter
- \( f_t \) = filter impulse response function

and \( * \) is the mathematical operation of convolution.

A discrete formulation of the convolution operation is:

\[ y_t = \sum_s x_s \cdot f_{t-s} \]

1. Compute (and plot) the output \( y_t \) that you would expect from the filter:

\[ f_t = 6, 5, -1, 3, 2 \]

for the following inputs:

a) \( x_t = 1, 0, 0, 0, 0 \)

This can be done explicitly using the formula above, by using the convolver in MATLAB or by using z transforms. Here I will use Z transforms (because it’s an easy one).

\[ F(z) = 6z^0 + 5z^1 - 1z^2 + 3z^3 + 2z^4 \]
\[ X(z) = 1z^0 \]

\[ y_t = x_t * f_t \] is equivalent by the convolution theorem to
\[ Y(z) = F(z) \cdot X(z) = (1) \cdot (6 + 5z^{1-1}z^2 + 3z^3 + 2z^4) = 6 + 5z^{1-1}z^2 + 3z^3 + 2z^4 \] which corresponds of course to
\[ y_t = 6, 5, -1, 3, 2 \]
in other words, the output of a filter for a impulse input is just filter itself. That is why \( f_t \) is called the filter’s impulse response function.

And if you wish to find out what an unknown filter will do, just put in an input and measure the corresponding output to determine its \( f_t \).

b) \( x_t = 0, 0, 1, 0, 0 \)

This is another good one to do with Z transforms.

\[ Y(z) = F(z) \cdot X(z) = (1z^2) \cdot (6 + 5z^{1-1}z^2 + 3z^3 + 2z^4) \]
\[ = 6z^2 + 5z^{3-1}z^4 + 3z^5 + 2z^6, \text{ or} \]
\[ y_t = 0, 0, 6, 5, -1, 3, 2 \]
or the output is simply a delayed version of the filter response.

c) \( x_t = -8, 2, 3, 6, 7 \)

OK, now I have to do some real math, so let’s go to MATLAB to get

\[ y_t = \text{conv}(x_t, f_t) = -48 \quad -28 \quad 36 \quad 25 \quad 59 \quad 42 \quad 17 \quad 33 \quad 14 \]

Now I’ll use the bar function in matlab to plot all these guys:

\[ f_t \]
2. Compute (and plot) the Fourier Transforms of $y_t$, $f_t$, and $x_t$ from 1c above.

OK, by the numbers:

$$f_t = 6, 5, -1, 3, 2$$
$$x_t = -8, 2, 3, 6, 7$$
$$y_t = -48, -28, 36, 25, 59, 42, 17, 33, 14$$

now I’ll use `fft` in matlab to obtain $F_j$, $X_j$, $Y_j$

Here is what I got:

$F = 15.0000, 6.5451 - 0.5020i, 0.9549 - 5.5676i, 0.9549 + 5.5676i, 6.5451 + 0.5020i$

$X = 10.0000, -12.5000 + 6.5186i, -12.5000 + 0.0858i, -12.5000 - 0.0858i, -12.5000 - 6.5186i$

$Y = 1.5000, -1.6265 + 0.1130i, -0.5890 + 0.5819i, -0.8400 + 0.2425i, 0.1455 + 0.2611i, 0.1455 - 0.2611i, -0.8400 - 0.2425i$
-0.5890 - 0.5819I, -1.6265 - 0.1130I (all times 100)

Now the product of X and F is:

\[ X \cdot F = 1.5000, -0.7854 + 0.4894i, -0.1146 + 0.6968i, -0.1146 - 0.6968i, -0.7854 - 0.4894i \]

How does the product of the Fourier Transforms of \( x_t \) and \( f_t \) compare with the Fourier Transform of \( y_t \)?

Hmmmm.... this doesn’t look correct. After all, according to the convolution theorem, \( X \cdot F \) should equal \( Y \). But it obviously doesn’t.

The explanation of this difference has to do with the nature of the deconvolution we used. Remember that the Fourier Transform assumes that both \( f_t \) and \( x_t \) are periodic. However, the convolution (as defined above) treats the functions as they are, that is, non-periodic (or linear).

In order to make the convolution theorem work in this context, we have to use something called circular convolution, one that treats the functions \( f_t \) and \( x_t \) as periodic.

Manually this looks like:

\[
\begin{array}{cccccc}
6 & 5 & -1 & 3 & 2 & 6 \\
7 & 6 & 3 & 2 & -8 & 7 \\
 & & & 6 & 3 & 2 \\
\end{array}
\]

\[ -48 + 35 - 6 + 9 + 4 = -6 \]

\[
\begin{array}{cccccc}
6 & 5 & -1 & 3 & 2 & 6 \\
-8 & 7 & 6 & 3 & 2 & -8 & 7 \\
 & & & 6 & 3 & 2 \\
\end{array}
\]

\[ 12 - 40 - 7 + 18 + 6 = -11 \]

\[
\begin{array}{cccccc}
6 & 5 & -1 & 3 & 2 & 6 \\
-8 & 7 & 6 & 3 & 2 & -8 & 7 \\
 & & & 6 & 3 & 2 \\
\end{array}
\]

\[ 18 + 10 + 8 + 21 + 12 = 69 \]

\[
\begin{array}{cccccc}
6 & 5 & -1 & 3 & 2 & 6 \\
3 & 2 & -8 & 7 & 6 & 3 \\
 & & & 2 & -8 & 7 \\
\end{array}
\]

\[ 6 + 2 - 8 + 7 + 6 + 3 + 2 \]
\[36+15-2-24+14=39\]

\[6\ 5\ -1\ 3\ 2\ 6\ 5\ -1\ 3\ 2\ 6\ 5\]
\[6\ 3\ 2\ -8\ 7\ 6\ 3\ 2\ -8\ 7\ 6\ 3\ 2\]
\[42+30-3+6-16=59\]

Now we take the fft of this and get:

\[Y_j = 1.5000, -0.7854 + 0.4894i, -0.1146 + 0.6968i, -0.1146 - 0.6968i, -0.7854 - 0.4894i\]

which is identical to \(X \cdot F\), so the convolution theorem lives!

As some of you noted, you can “fool” the linear convolution into doing a circular convolution by padding with enough zeroes so that the number terms in \(X\) and \(F\) is equal to the number of terms in \(Y\) computed from the original series \(x\) and \(f\).

Here are plots of the above transforms.
3. The convolution operation is reversible. Deconvolution can be thought of as applying a filter \( f_t^{-1} \) that negates the effect of some original filter \( f_t \), i.e.,

\[
f_t * f_t^{-1} = 1, \text{ therefore}
\]

\[
y_t * f_t^{-1} = x_t * f_t * f_t^{-1} = x_t * 1 = x_t
\]
The computation of this inverse can be aided by the use of Fourier (transform, divide, inverse transform) or Z (polynomial division) transforms. MATLAB has a command that performs deconvolution by polynomial division.

Compute (and plot) the first 4 coefficients of the spiking deconvolution filter corresponding to \( f_t = 36, -2, 18, -2 \).

I’ll be lazy and use MATLAB here. First I’ll define my spike to be:

\( y_t = 1,0,0,0,0,0,0 \)

This gives the first 4 terms as:

\( f_t^{-1} = 0.0278, 0.0015, -0.0138, 0.0000 \)

Note that I used a bunch of zeroes to be sure that deconv returns enough terms (i.e. the number of terms it returns is equal to the difference between the spike output and the filter)

What is the result of convolving this deconvolution filter with \( f_t \)?

\( y_t = f_t * f_t^{-1} = 1.0000 0 0.0000 0.0000 -0.2516 0.0277 -0.00000 \)

which looks pretty much like a spike:

Plot the result.
Compute (and plot) the first 10 coefficients of the spiking deconvolution filter corresponding to $f_t$. How does this result compare with the output from the 4 term decon filter?

$$f_t^{-1} = 0.0278 \quad 0.0015 \quad -0.0138 \quad 0.0000 \quad 0.0070 \quad -0.0004 \quad -0.0035 \quad 0.0004 \quad 0.0018 \quad -0.0003$$

The resulting spike looks like:

$$\text{spike} = 1.0000 \quad 0 \quad 0.0000 \quad 0.0000 \quad 0.0000 \quad 0.0000 \quad -0.0000 \quad 0.0000 \quad 0.0000 \quad 0.0000 \quad 0.0314 \quad -0.0087 \quad 0.0006$$

Again not bad.
Compute the least squares error of the two resulting deconvolution outputs above assuming that the desired output is the perfect “spike”, i.e. 1,0,0,0,.....

This we can get from the remainder of the deconvolution. If we just sum the squares of the respective remainders we get:

4 terms \( \text{mse} = 0.2531 \)

10 terms \( \text{mse} = 0.0332 \)

How does this measure of goodness of fit compare between the two filters?

Obviously the longer filter is better.

What does this suggest as an approach to picking a spiking deconvolution filter?

More terms are better.