Finite Impulse Response (FIR) Filter Design and Analysis

by Orang J Assadi, B. Electronics and Communication Engineering, University of Canberra, Australia, assadij@hotmail.com

Abstract: Finite Impulse Response Filter Design and Analysis.
Application Areas/Subjects: Engineering, Electrical and Electronic Engineering, Applied Examples, Communications and Signal Processing
Keywords: Filter, frequency, response, spectrum, spectra, signal processing, communications

Figure 1: N+1 Order FIR filter. Each 1/z block corresponds to one sample-time delay.

$$y_n = \sum_{k=0}^{N} h_k x_{n-k}$$

> **restart:**

Finite Impulse Response (FIR) Filter

> **readlib(FFT):**

> **with(plots):**

DSProcessor specification, filter and analysis

> **readlib(FFT):**

requirements

> **N := 100; # Number of delays(must be even)=order of filter - 1**

$$N := 100$$

> **with(plots):**

DSProcessor specification, filter and analysis requirements

> **N := 100; # Number of delays(must be even)=order of filter - 1**

$$N := 100$$

> **fs := 10000; # Sampling frequency**

$$fs := 10000$$

> **fl := 3000; # Low pass cut-off frequency**

$$fl := 3000$$

> **fh := 2000; # High pass cut-off frequency**

$$fh := 2000$$

> **m := 10; # 2^m = munber of points for analysis (must be > N)**

$$m := 10$$

Analysis functions and constants

> **T := 2^m-1;**

$$T := 1023$$

> **F1 := evalf(fl/fs);**

$$F1 := .3000000000$$

> **F2 := evalf(fh/fs);**

$$F2 := .2000000000$$

> **Dirac(0) := 1;**

$$\mathrm{Dirac}(0) := 1$$

Selecting the filter type: 1-Lowpass 2-Highpass 3-Bandpass 4-Bandstop (fh > fl)

```
> typ := 1; # Filter type
```

$$typ := 1$$

```
> if typ = 1 then
> g := sin(t*2*Pi*F1)/(t*Pi);
> elif typ = 2 then
> g := Dirac(t)-sin(t*2*Pi*F2)/(t*Pi);
> elif typ = 3 then
> g := (sin(t*2*Pi*F2)-sin(t*2*Pi*F1))/(t*Pi);
> else
> g := sin(t*2*Pi*F1)/(t*Pi)+Dirac(t)-sin(t*2*Pi*F2)/(t*Pi);
> fi;
```

$$g := \frac{\sin(.6000000000\, t\, \pi)}{t\, \pi}$$

```
> C := (n) -> limit(g,t=n);
```

$$C := n \to \lim_{t \to n} g$$

Compute the FIR coefficients (Rectangular Window)

```
> h := aray(0..N):
```

```
> for n from 0 to N/2 do
> h[N/2-n] := evalf(C(n));
> h[N/2+n] := h[N/2-n];
> od:
```

Declare input x(n) and output y(n) signals

```
> x := array(-N..T):
> y := array(0..T):
```

Set input = 0 for time < 0

```
> for n from -N to -1 do
> x[n] := 0;
> od:
```

Set input signal as an impulse

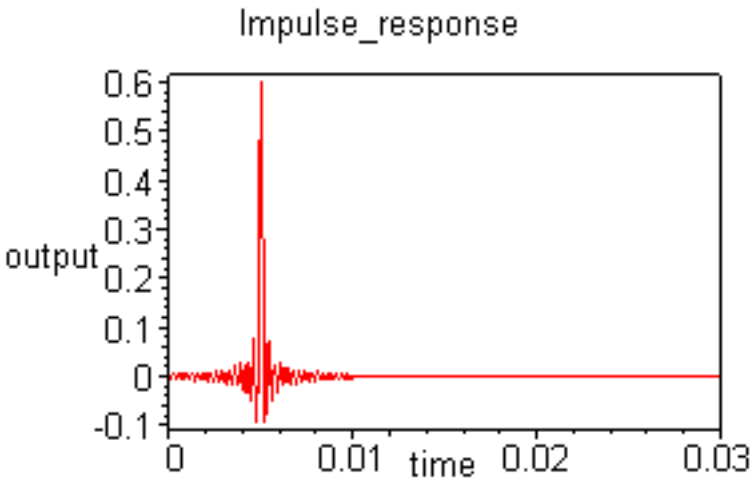
```
> for n from 0 to T do
> x[n] := Dirac(n);
> od:
```

Compute the corresponding output signal

```
> for n from 0 to T do
> y[n] := sum(h[k]*x[n-k],k=0..N);
> od:
```

Plot the impulse response

```
> p := [seq([j/fs,y[j]],j=0..T)]:

> plot(p, time=0..3*N/fs, labels=[time,output], axes=boxed, xtickmarks=4,
title='Impulse_response');
```

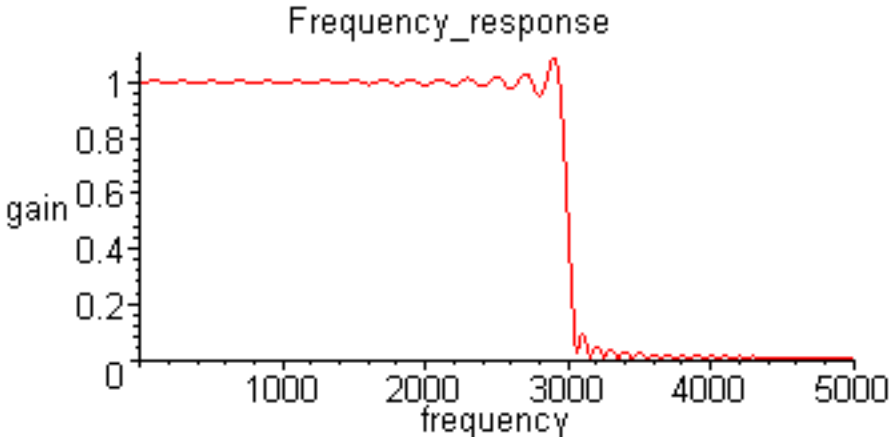


Compute the frequency response from the impulse response

```
> ro := array(1..T+1):

> io := array(1..T+1):

> for n from 0 to T do

> ro[n+1] := y[n];

> io[n+1] := 0;

> od:

> FFT(m,ro,io):
```

Plot the frequency response; gain vs frequency

```
> p := [seq([j*fs/(T+1),abs(ro[j+1]+io[j+1]*I)],j=0..T/2)]:

> plot(p, frequency=0..fs/2, labels=[frequency,gain], title='Frequency_response');
```



Now we test the filter in time and frequency domain by applying a test signal

Set input test-signal as a 2Vpp, 500Hz square-wave + Noise

```
> l := round(fs/2/500):
> for n from 0 by 2*l to T do
> for n2 from 0 to l-1 do
> if n+n2 <= T then
> x[n+n2] := evalf(-1+rand()/10^12-0.5);
> fi;
> if n+n2+l <= T then
> x[n+n2+l] := evalf(1+rand()/10^12-0.5);
> fi;
> od:
> od:
```

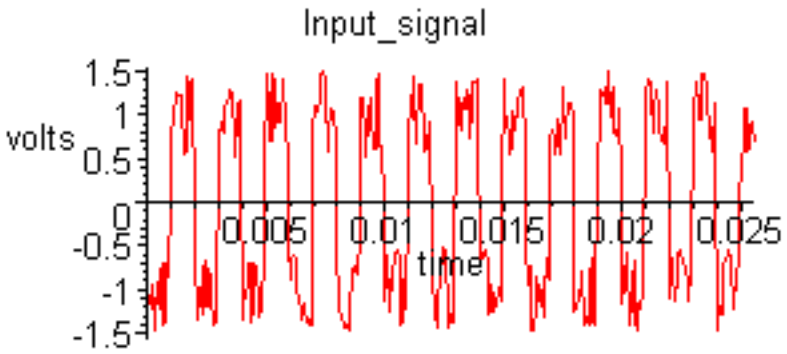
Compute the corresponding output signal

```
> for n from 0 to T do
```

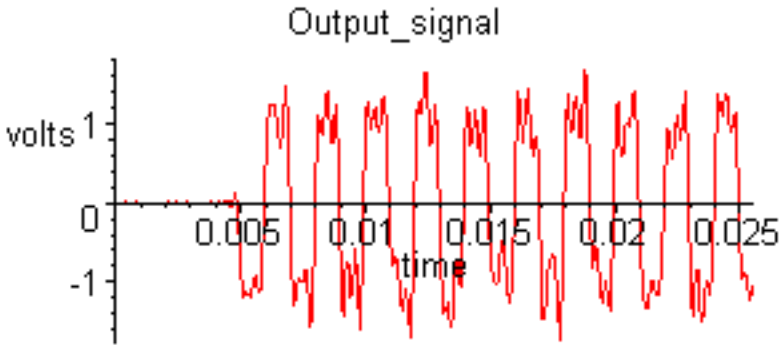
```
> y[n] := sum(h[k]*x[n-k],k=0..N);

> od:
```

Plot the input and output signal

```
> p := [seq([j/fs,x[j]],j=0..T)]:

> q := [seq([j/fs,y[j]],j=0..T)]:

> plot(p,time=0..T/fs/4,labels=[time,volts],title='Input_signal');
```




```
> plot(q,time=0..T/fs/4,labels=[time,volts],title='Output_signal');
```



Compute the input & output spectrum

```
> ri := array(1..T+1):

> ii := array(1..T+1):

> for n from 0 to T do

> ri[n+1] := x[n]*2/T;
```

> **ii[n+1] := 0;**

> **ro[n+1] := y[n]*2/T;**

> **io[n+1] := 0;**

> **od:**

> **FFT(m,ri,ii):**

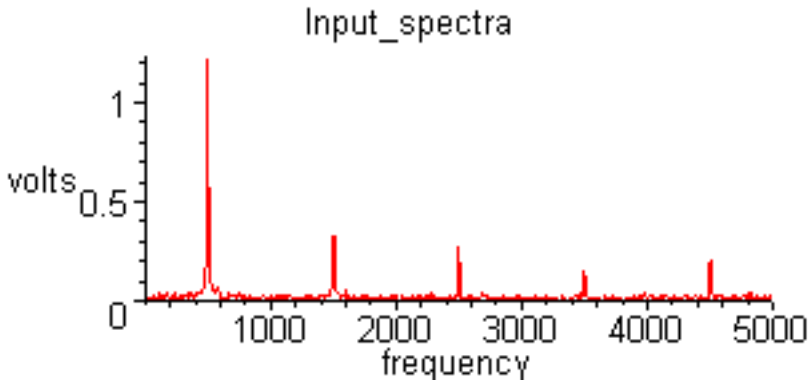
> **FFT(m,ro,io):**

Plot the input and output spectrum for the test-signal

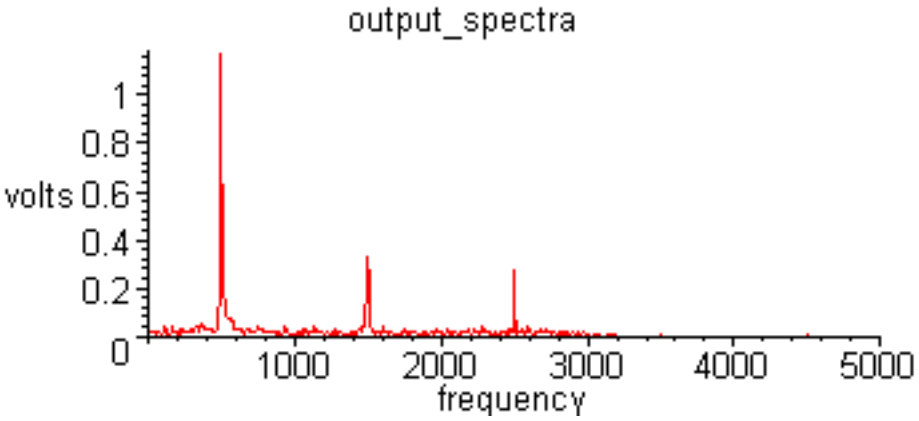
> **p := [seq([j*fs/(T+1),abs(ri[j+1]+ii[j+1]*I)],j=0..T/2)]:**

> **q := [seq([j*fs/(T+1),abs(ro[j+1]+io[j+1]*I)],j=0..T/2)]:**

> **plot(p, frequency=0..fs/2,labels=[frequency,volts], title='Input_spectra'); # frequency range cannot exceed Nyquist frequency (fs/2)**



> **plot(q, frequency=0..fs/2,labels=[frequency,volts], title='output_spectra'); # frequency range cannot exceed Nyquist frequency (fs/2)**



Reference

[1] Digital Signal Processing with TMS320C25

>