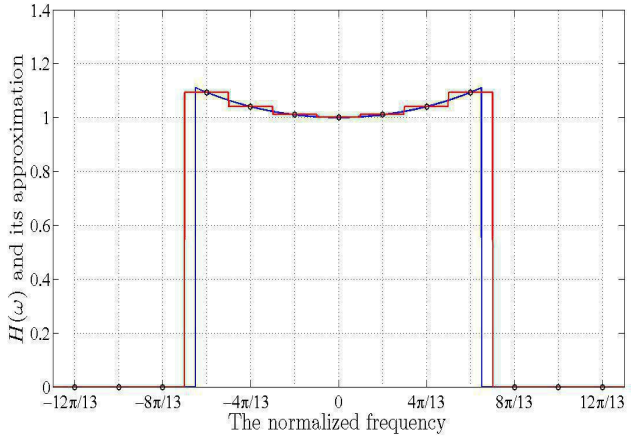
# FIR filter design by frequency sampling

The frequency sampling method is based on defining the desired frequency response of the filter and after that by sampling the desired frequency response at a given number of points. For example, if it's desired to design a low pass filter we will start with Fig. 1. The red curve shows the approximation of the desired frequency response.




The desired frequency response is sampled at 32 points at regular frequency intervals. Based on the samples of the desired frequency response we can calculate the impulse response of the filter using the following equation.

$h_d(n)=1/(2*pi)sum(H(omega_k)e^{jomegakn})$

We start with a low pass filter and a cut off frequency equal to 6/16 which is normalized to half of the sampling rate. In the table below we give the desired frequency points. The desired filter length is 10.

| F | 0 | 1/16 | 2/16 | 3/16 | 4/16 | 5/16 | 6/16 | 7/16 | 8/16 | 9/16 | 10/16 | 11/16 | 12/16 | 13/16 | 14/16 | 15/16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H(o) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

In a sequel we show the filter design for a given test example.

$$Hd := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix} \qquad n := 31$$

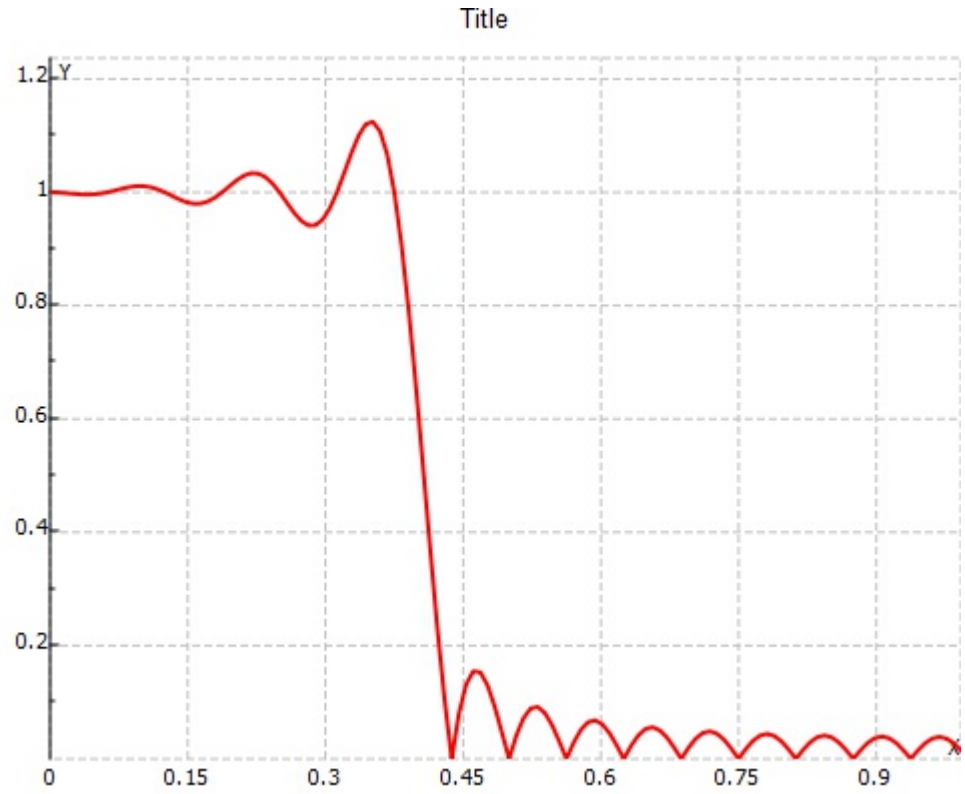
$h := firfreqsampl(n, Hd, 1, 0)$

Next, we calculate the frequency response of the obtained filter using the function, firfreqres

$$Ho := \text{firfreqres}\left(h / \text{vecsum}(h)\ ,\ 128\ ,\ 1\right)$$

$$freq := \text{curve2d}(z\ ,\ 0\ ,\ 1 - 1/128\ ,\ 128)$$

$$frgraf := \text{join mat cols}\left(\text{col2vec}(freq\ ,\ 0)\ ,\ \text{fabs}(Ho)\right)$$



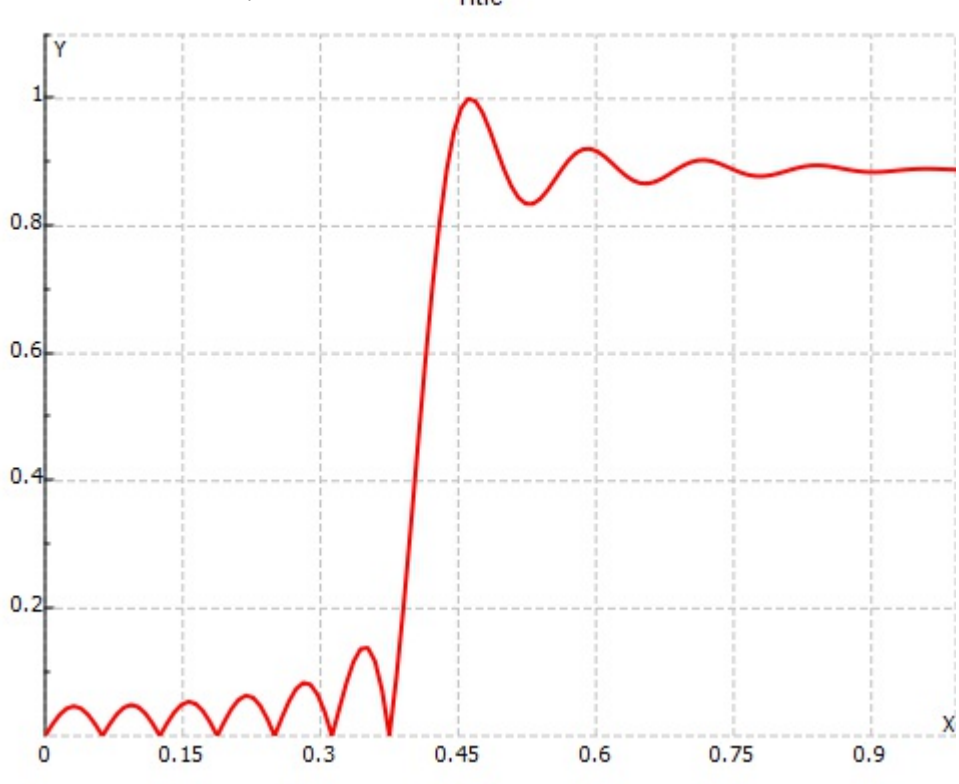

Next, we design a highpass filter with the same parameters.

$$H1d := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

$h1 := \text{firfreqsampl}(n , H1d , 1 , 1)$

$Ho1 := \text{firfreqres}(h1 , 128 , 1)$

$frgraf1 := \text{join mat cols}\left(\text{col2vec}(freq , 0) , \text{fabs}(Ho1) / \text{mat max}\left(\text{fabs}(Ho1)\right)\right)$



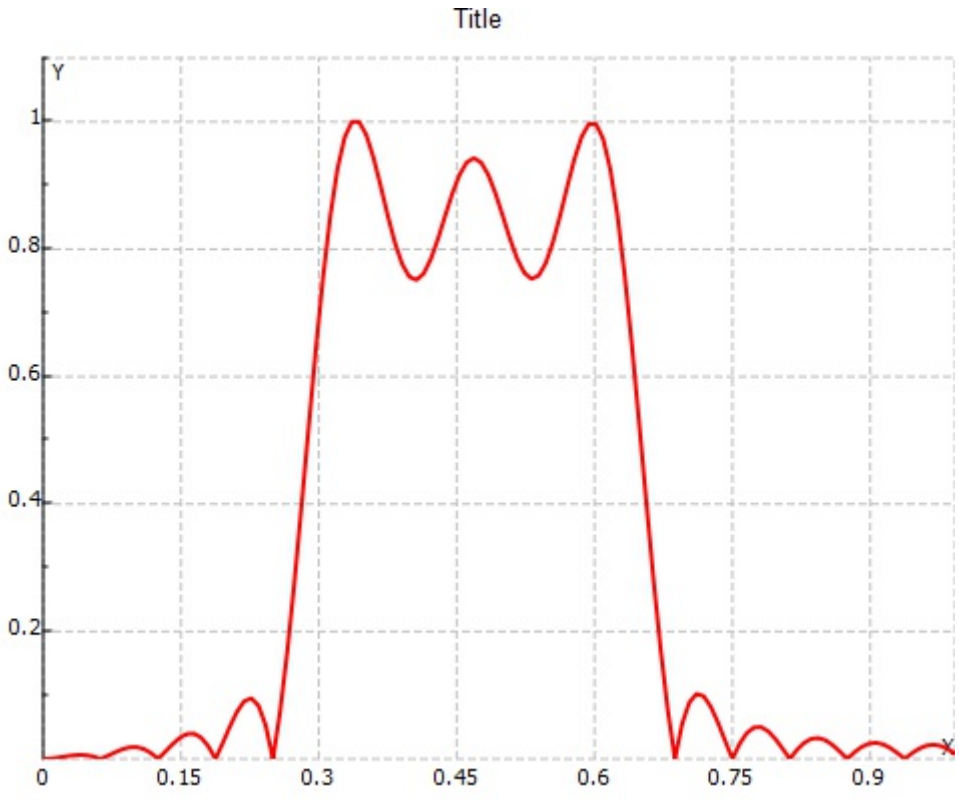

Bandpass and bandstop filters are both designed in a sequel.

$$Hband := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ \vdots \end{bmatrix}$$

$hbp := \text{firfreqsampl}(n, Hband, 0.5, 2)$

$Hobp := \text{firfreqres}(hbp, 128, 1)$

$frgraf2 := \text{join mat cols}\left(\text{col2vec}(freq, 0), \text{fabs}(Hobp) / \text{mat max}\left(\text{fabs}(Hobp)\right)\right)$

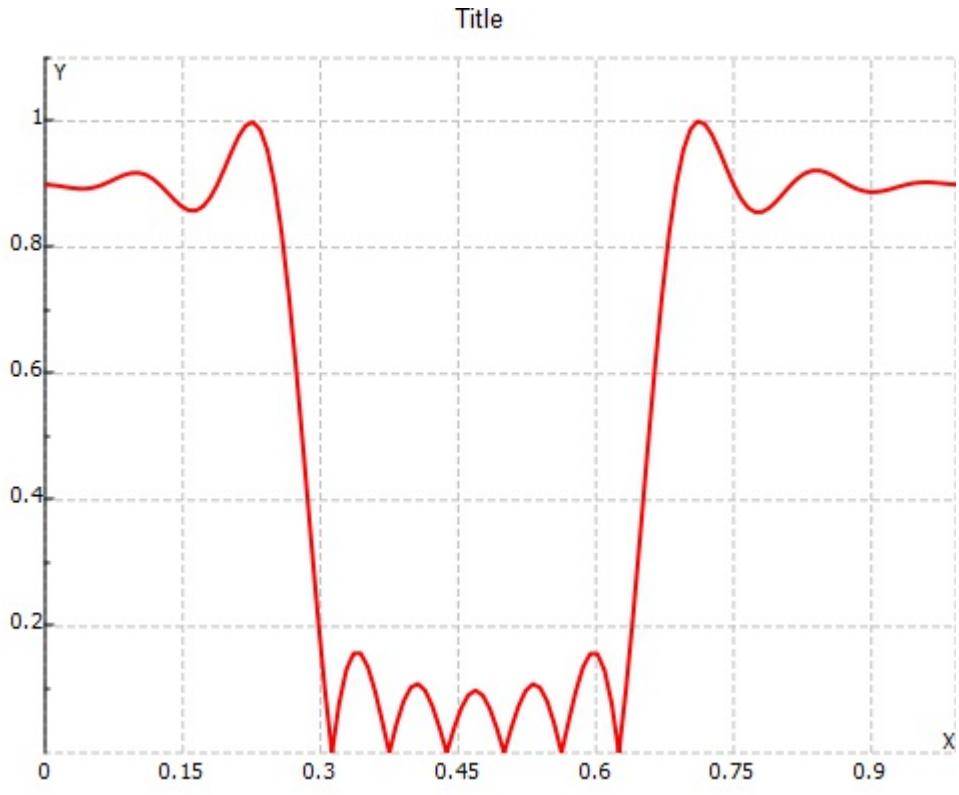


$$Hbands := \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

$$hbs := \text{firfreqsampl}(n - 1 \ , Hbands \ , 0.5 \ , 3)$$

$$Hobs := \text{firfreqres}(hbs \ , 128 \ , 1)$$

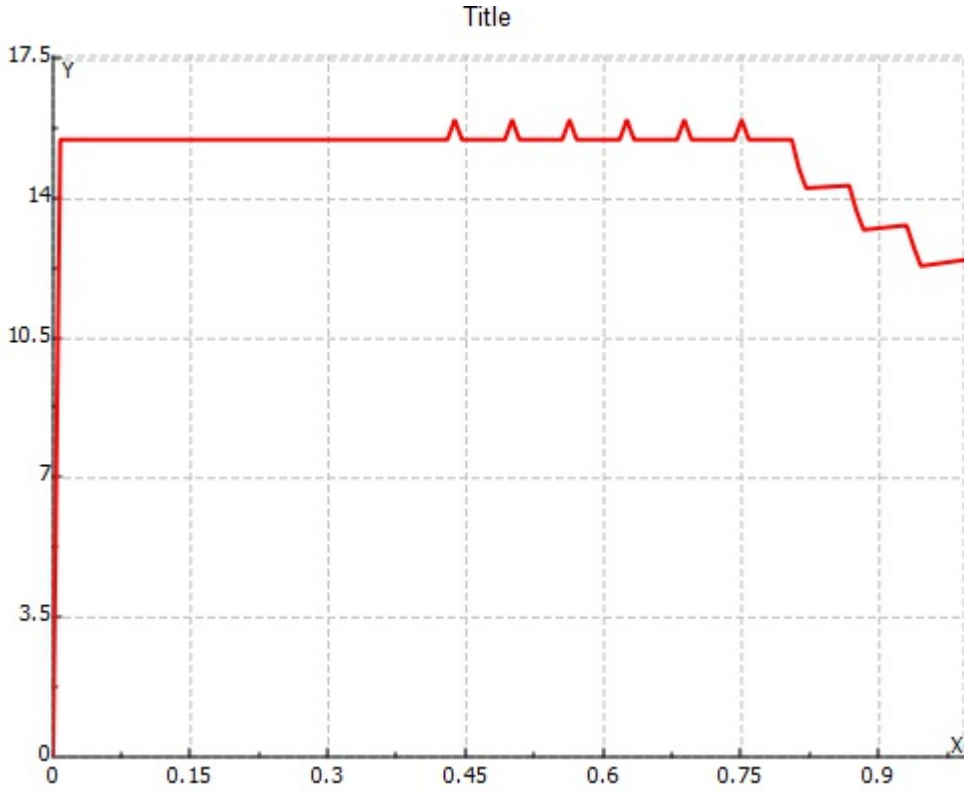
$$frgraf3 := \text{join mat cols}\left(\text{col2vec}(freq \ , 0) \ , \text{fabs}(Hobs) / \text{mat max}\left(\text{fabs}(Hobs)\right)\right)$$



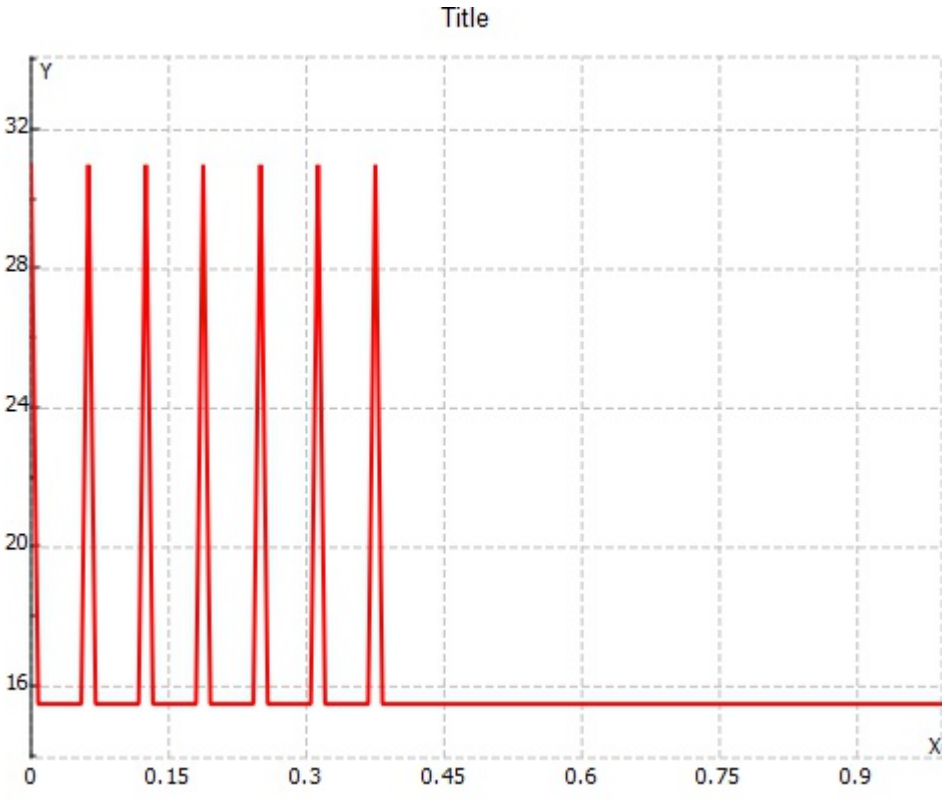
```
vecsum(vec1)
{
    ssum := 0
    for(i := 0 , i < size(vec1) , i += 1 )
    {
        ssum = ssum + vec1[i]
    }

    return(ssum)
}


firphasedelay(vec1 , numpoints)
{
    frres := firfreqres(vec1 , numpoints , 1)
    phres := contphase(frres)
    fr := vector create(numpoints , 0 , 0)
    mat := vector create(numpoints , 0 , 0)
    for(i := 1 , i < numpoints , i += 1 )
    {
        fr[i] = (π) · i / numpoints
        phres[i] = 0 - phres[i] / fr[i]
    }

    return(phres)
}
```

We check the phase delay and group delay of an FIR filter

firfd := join mat cols(col2vec(freq , 0) , firphasedelay(h , 128))




firgd := join mat cols(col2vec(freq , 0) , firgroupdelay(h1 , 128))



```
firgroupdelay(vec1 , numpoints)
{

    ve1 := vec1
    oa := size(ve1) - 1
    if(oa < 0 )
    {
        ve1 = 1
        oa = 0
    }

    oc := oa
    c := flip(conj(ve1) , 1)
    cr := vector create(oc + 1 , 0 , 0)
    for(i := 0 , i <= oc , i += 1 )
    {
        cr[i] = c[i] · i
    }

    num := fft1n(cr , 2 numpoints)
    den := fft1n(c , 2 numpoints)
    for(k := 0 , k < numpoints , k += 1 )
    {
        if( fabs(den[k]) < 10^-16 )
        {
            num[k] = 0
            den[k] = 1
        }
    }
    Matr := vector create(numpoints , 0 , 0)
    for(j := 0 , j < numpoints , j += 1 )
    {
        Matr[j] = complexre(num[j] / den[j]) - oa
        Matr[j] = -Matr[j]
    }

    return(Matr)
}
```