Signal & Spectral Processing

CPE 381 Foundations of Signals & Systems
for Computer Engineers

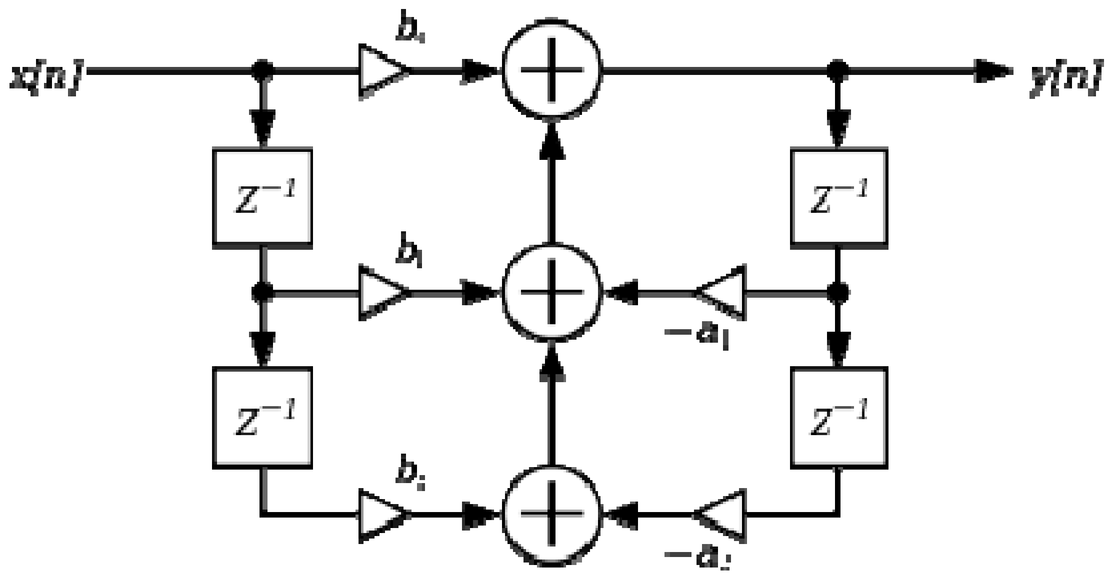
Dr. Emil Jovanov

Filter Implementation: Direct Form 1

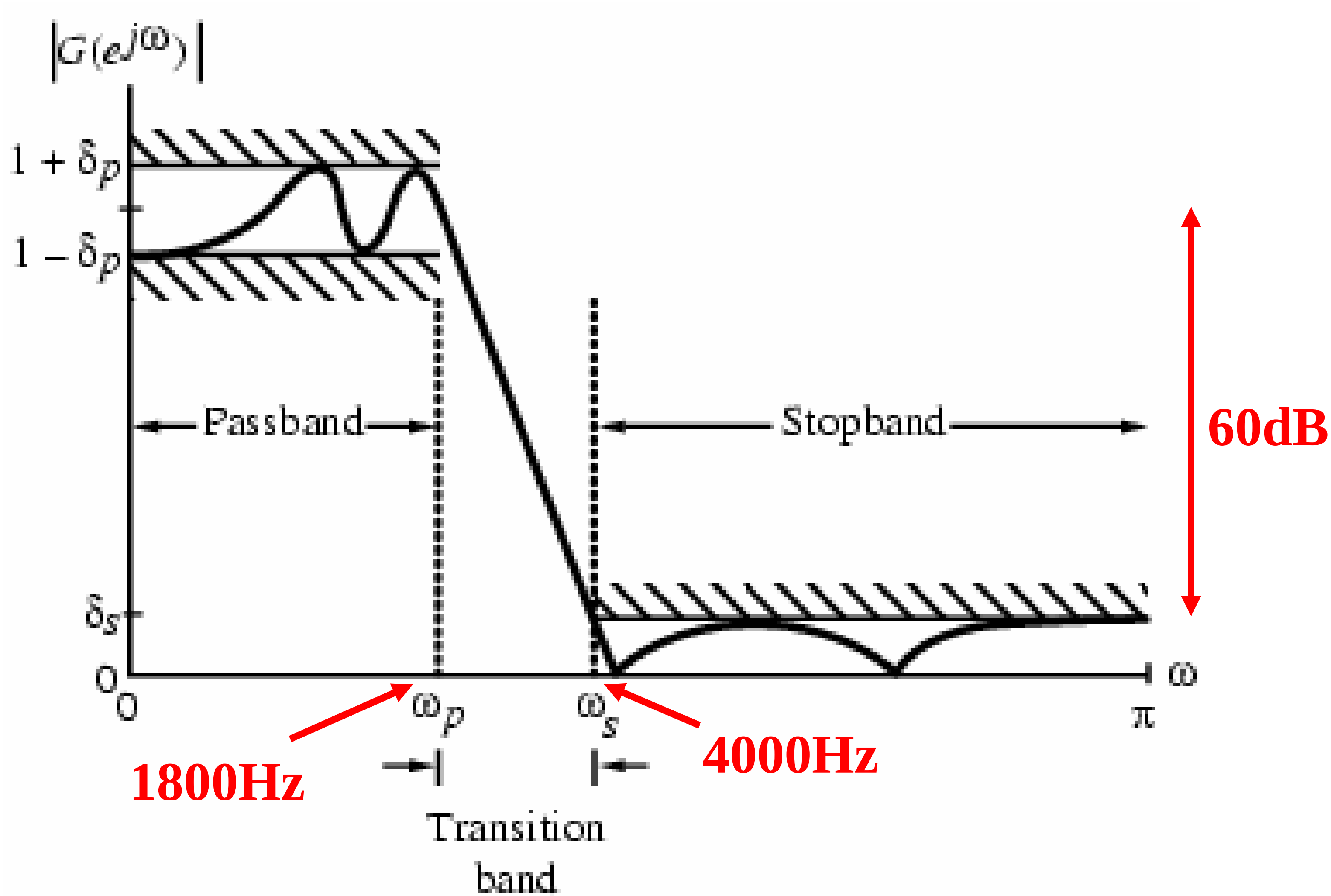
- The most straightforward implementation is the Direct Form 1, which has the following different equation:

$$y(n) = b_0 x(n) - b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2)$$

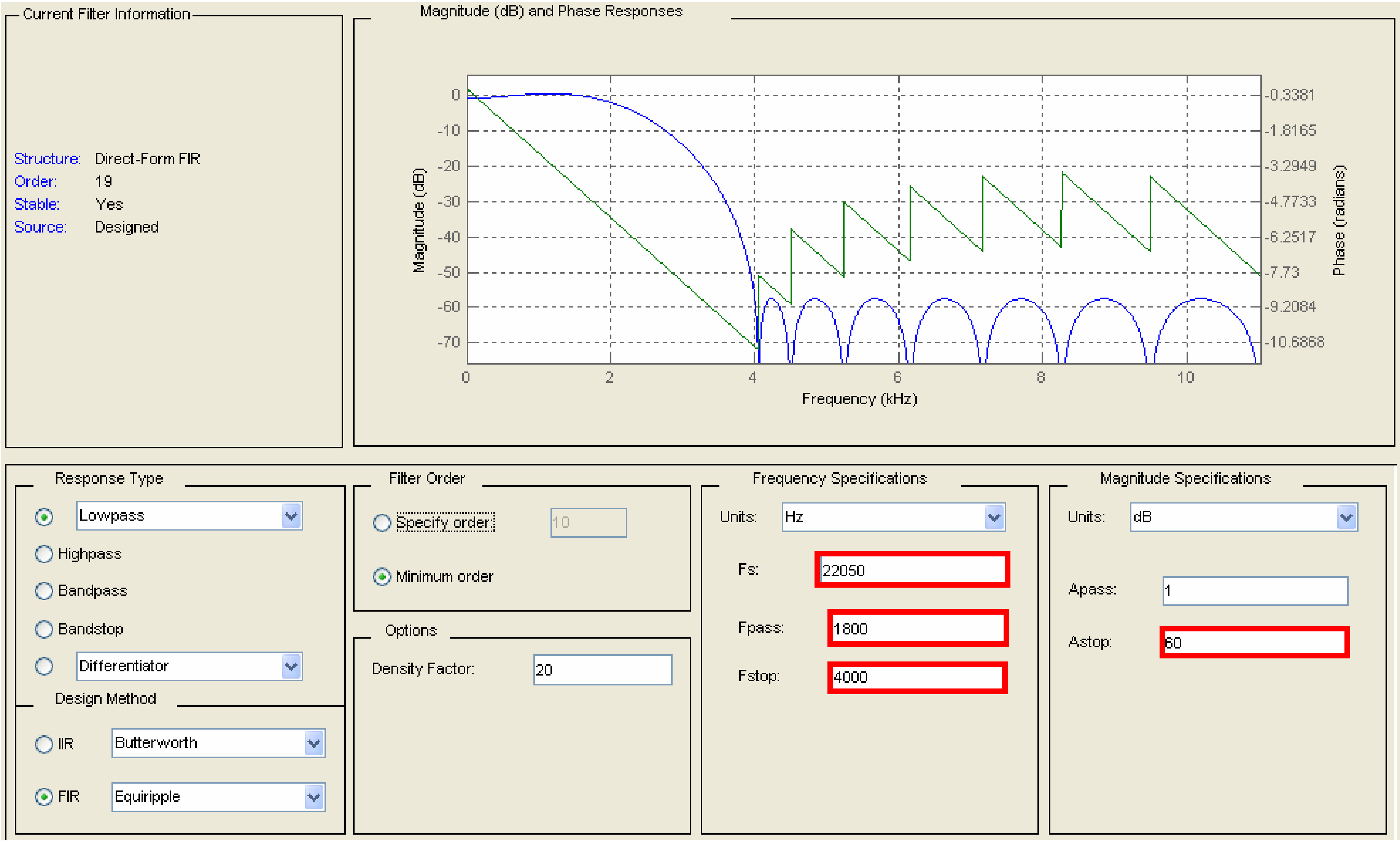
- Here the b0, b1 and b2 coefficients determine zeros, and a1, a2 determine the position of the poles. Flow graph of biquad filter:



Filter Specification



Filter Design – Matlab Filter Design and Analysis Tool



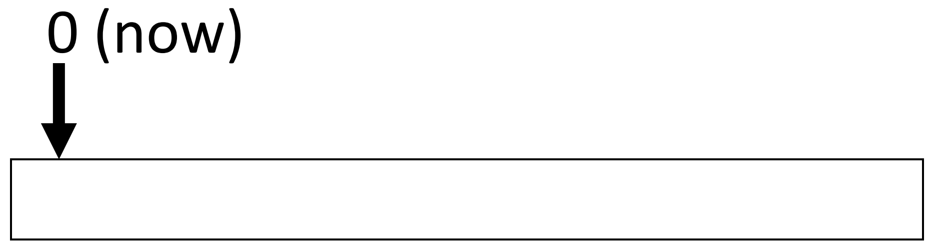
Filter Coefficients

- FIR filter coefficients (floating point)

```
/* Filter Coefficients (C Source) generated by  the Filter Design and Analysis Tool
 *
 * Generated by MATLAB(R)
 */
const int BL_22050 = 20;
const double B_22050[20] = {
   -0.00273488123218, -0.01117280270486, -0.02299447730651,  -0.0337783690733,
   -0.03171608567814,-0.006618745064523,  0.04555196284397,   0.1157521469277,
     0.1837840794056,   0.2257629836841,   0.2257629836841,   0.1837840794056,
     0.1157521469277,  0.04555196284397,-0.006618745064523, -0.03171608567814,
    -0.0337783690733, -0.02299447730651, -0.01117280270486, -0.00273488123218
};

const int BL_44100 = 40;
const double B_44100[40] = {
  -0.0005622283432447,-0.002553596820629,-0.004059851824371,-0.006854899768296,
   -0.00991062107269, -0.01312755662113, -0.01585699258984, -0.01742469011063,
    -0.0170517927008, -0.01401690254818,-0.007763346089489, 0.001973852556403,
    0.01507411052057,  0.03098082946882,   0.0487176168622,  0.06696024808886,
    0.08417689221254,  0.09880533118292,   0.1094463136109,   0.1150490972761,
     0.1150490972761,   0.1094463136109,  0.09880533118292,  0.08417689221254,
    0.06696024808886,   0.0487176168622,  0.03098082946882,  0.01507411052057,
   0.001973852556403,-0.007763346089489, -0.01401690254818,  -0.0170517927008,
   -0.01742469011063, -0.01585699258984, -0.01312755662113, -0.00991062107269,
  -0.006854899768296,-0.004059851824371,-0.002553596820629,-0.0005622283432447
};
```

Filtering

- Init




```c
// input & output samples
#define FILT_LEN 12

int NB=FILT_LEN;                    // filter length


/*** Filter initialization ***/
void filt_init_var(int *x, int *y) {
  register int ii;

  for (ii=0; ii<FILT_LEN; ii++)
    x[ii] = y[ii] = 0;
}
```

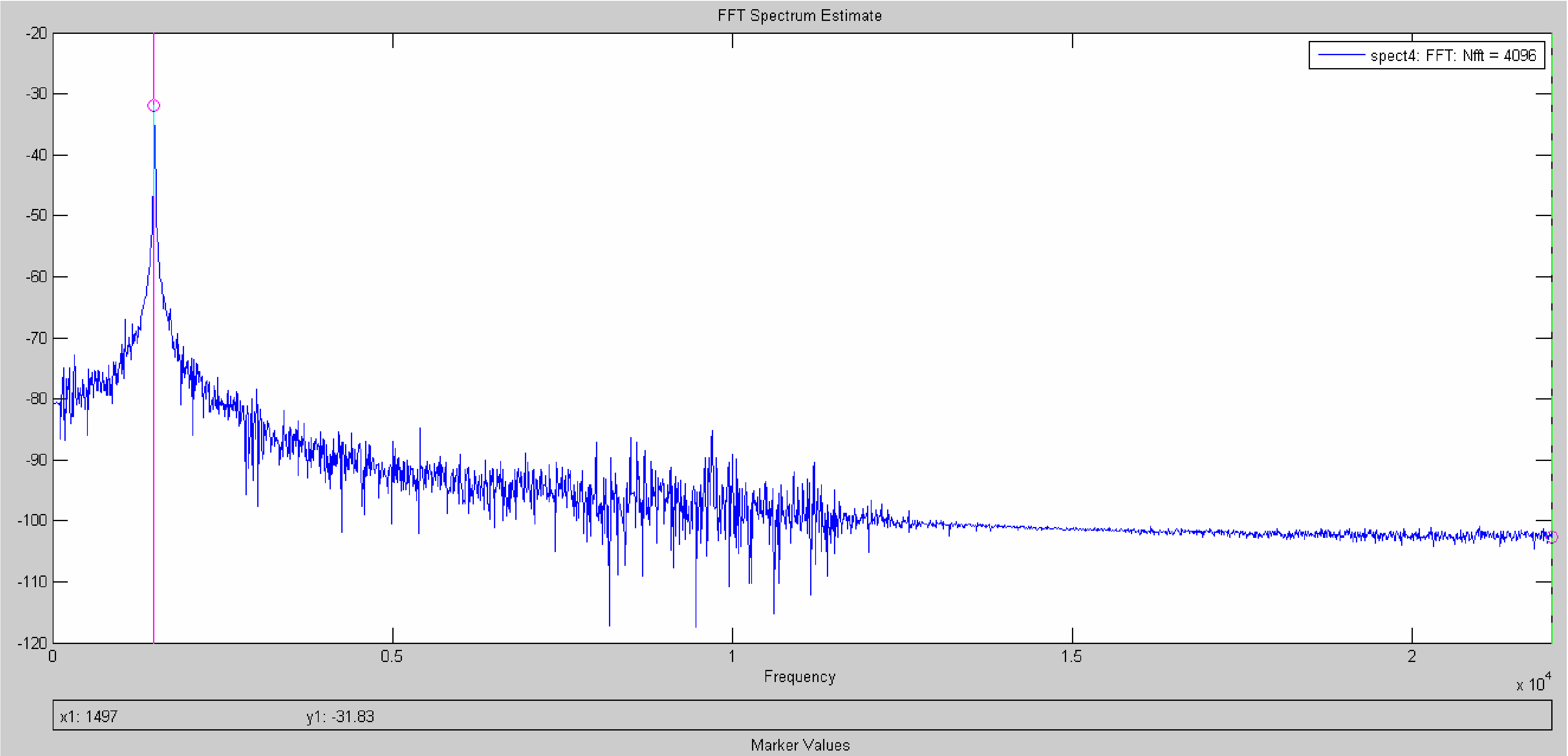
Filtering

- FIR filter (floating point)

```
void xiir_filter(int * x, int * y, int sample)
{
        /* fixed point filter procedure
                xin - input signal
                yout - filtered input signal
        */
        long templ;
        register int i;

        /* the latest sample is at index 0, all other are shifted */
        for (i=NB-1;i>0;i--) {
                x[i]=x[i-1];
                y[i]=y[i-1];
        }
        x[0]=sample;

// FIR filter
        templ=0;
        for (i=0;i<NB;i++) {
                templ += x[i]*B[i];
                }

        y[0]=(int)templ;
}
```

BONUS - Determining the frequency of the added sine wave

- FFT Spectrum of the wave file with added sine wave



FFT Calculation in C

- Many available libraries (e.g.: FFTW.org)
- Simple function by Jon Harrop *

```
void fft(int sign, vector<complex<double>> &zs) {
    unsigned int j=0;

    for(unsigned int i=0; i<zs.size()-1; ++i) {
        if (i < j) {
            auto t = zs.at(i);
            zs.at(i) = zs.at(j);
            zs.at(j) = t;
        }
        int m=zs.size()/2;
        j^=m;
        while ((j & m) == 0) { m/=2; j^=m; }
    }
    for(unsigned int j=1; j<zs.size(); j*=2)
        for(unsigned int m=0; m<j; ++m) {
            auto t = pi * sign * m / j;
            auto w = complex<double>(cos(t), sin(t));
            for(unsigned int i = m; i<zs.size(); i+=2*j) {
                complex<double> zi = zs.at(i), t = w * zs.at(i + j);
                zs.at(i) = zi + t;
                zs.at(i + j) = zi - t;
            }
        }
}
```

* http://stackoverflow.com/questions/10121574/safe-and-fast-fft

Determining frequency of sine wave noise

```
while(!feof(rFile))
    {
    fread(&inBufferCur, sizeof(short), 1, rFile); //Get next sample

    if(fftCount<FFT_LEN)  // Placing first FFT_LEN samples in buffer for FFT analysis
        {
        fftBuff.at(fftCount)= inBufferCur;
        fftCount++;
        }
        ....
    }
```

Preparing samples for FFT analysis
Performing FFT analysis

```
    //Performing FFT on then first FFT_LEN samples
    fft(1, fftBuff);

    double maxSpec = (fftBuff.at(0).real())*(fftBuff.at(0).real()) +  (fftBuff.at(0).imag())*(fftBuff.at(0).imag());
    double tmp = 0;
    int maxIndex = 0;

    //Determining frequency of sine wave noise by calculating position of the MAXIMUM in the spectrum
    for (int j=1;j<FFT_LEN;j++)
    {
    tmp = (fftBuff.at(j).real())*(fftBuff.at(j).real()) +  (fftBuff.at(j).imag())*(fftBuff.at(j).imag());
    if(tmp>maxSpec)
        {
        maxSpec = tmp;
        maxIndex = j;
        }
    }
```

Searching position of the MAX in the FFT Spectrum
Printing Frequency of the sine wave noise

```
    printf("Frequency of sine wave noise: %u Hz\n", maxIndex * fileHeader.SampleRate / FFT_LEN );
```

FM modulated Data Transfer

- Noise always present
  - Consider signal to noise ratio
- Example: FM modulated data transmission; Data represented by sine waves with different frequencies
  - Digital “0” – sine wave at 1600Hz
  - Digital “1” – sine wave at 2000Hz
- Challenge – fast and reliable detection each sine wave

Detection of a sine wave in real-time

```
% spectral analysis example, detection of a sine wave in real-time
fs=16000;    % sampling frequency
N=100000;    % number of samples
NFFT=1024;   % length of FFT window
df=fs/1024;  % delta frequency in FFT spectrum
n=1:N;
dt=1/fs;     % delta time (Ts)
t=n*dt;      % time [s]
```

Creating always present noise

```
noise=rand(1,N)-0.5;     % some random signal in the range -0.5 : 0.5
```

```
fsig1=1600;
fsig2=2000;
s1=2*sin(2*pi*fsig1.*t);  % embedded signal #1
s2=2*sin(2*pi*fsig2.*t);  % embedded signal #2
sn=noise;
```

Embedding signals for "Digital 0" and "Digital 1"

Creating signals for "Digital 0" and "Digital 1"

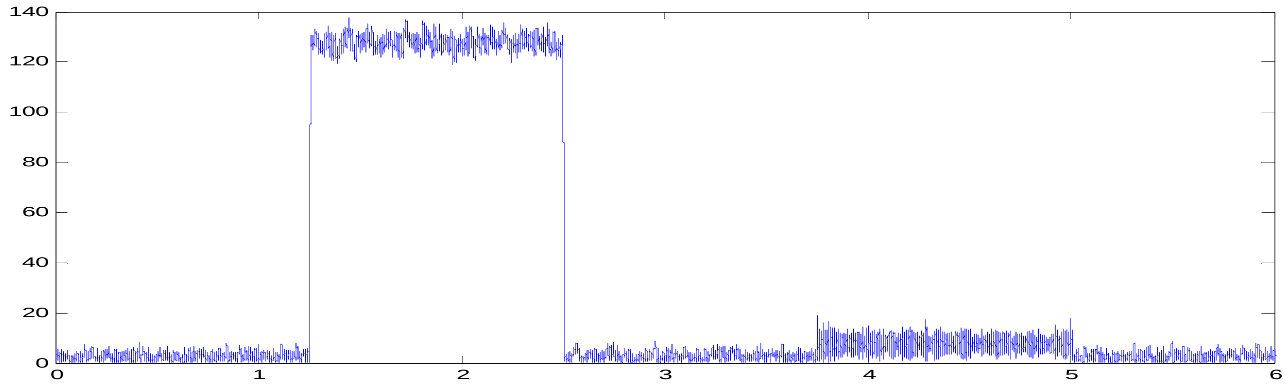
```
% embed signals #1 and #2
ind=20000:40000;
sn(ind)=s1(ind)+noise(ind);
sn(ind+40000)=s2(ind+40000)+noise(ind+40000);
```

Creating and applying matching filters

```
% FFT-like sine detection, matched filter of size 128
cs=sin(2*pi*fsig2*(1:128).*dt);
cc=cos(2*pi*fsig2*(1:128).*dt);
ys=filter(cs,1,sn);
yc=filter(cc,1,sn);
y=sqrt(ys.^2+yc.^2);    % spectrum magnitude at target frequency
plot(t,y)
```

Detection of a sine wave in real-time #2

- Signal after applying matching filters at 1600Hz



- Signal after applying matching filters at 2000Hz