EXPERIMENT-5

Theme: Digital FIR Filter Design

Name: Anupama Kulshreshtha

Roll no.: EE22BTECH11009

Design of digital filters such as Low Pass Filter (LPF), Band Pass Filter (BPF)

Aim of the experiment: To Design Digital FIR Low Pass and Band Pass filters using functions in MATLAB and C, and to plot their impulse and frequency response

Theory of the digital FIR filter:

FIR filters are a class of digital filters whose impulse response (the response to a unit impulse input) settles to zero in a finite period of time. This means that the output of the filter is solely determined by a finite number of input samples.

Filter Design Parameters:

- **Filter Length (N):** The number of coefficients in the filter determines its length. Longer filters typically provide sharper frequency response characteristics but may also introduce more computational complexity.
- **Cutoff Frequency (fc):** This is the frequency at which the magnitude response of the filter is attenuated by a certain amount. For low-pass and band-pass filters, this is the frequency below which or between which the majority of the signal is allowed to pass, respectively.
- **Sampling Frequency (fs):** The rate at which the signal is sampled. It determines the frequency range that can be represented in the digital domain without aliasing.

Types of Filters:

- **Low Pass Filter (LPF):** A low-pass filter allows frequencies below a certain cutoff frequency to pass while attenuating higher frequencies. The design involves determining filter coefficients that create the desired frequency response. In this experiment, we code a LPF function which designs a low-pass filter.
- **High Pass Filter (HPF):** A high-pass filter allows frequencies above a certain cutoff frequency to pass while attenuating lower frequencies. Similar to LPF, the design of an HPF involves determining filter coefficients that achieve the desired frequency response.
- **Band Pass Filter (BPF):** A band-pass filter allows a specific range of frequencies to pass while attenuating others. It typically has two cutoff frequencies defining the passband.

Windowing: It is a technique used to improve the performance of FIR filters. In the code, a Hamming window is applied to the filter coefficients to reduce the effect of spectral leakage, which occurs due to the abrupt truncation of the ideal impulse response.

As part of this experiment, we develop a MATLAB and C code which demonstrates the design and implementation of FIR filters, specifically low-pass and band-pass filters, using windowing method with Hamming window used for improved performance. These filters can be useful in various signal processing applications for tasks such as noise reduction, signal separation, and frequency band selection.

MATLAB Code:

```
% Define filter parameters
fc = 400; % Cutoff frequency in Hz for LPF
wc = pi/2; % Cutoff frequency in radians
N = 39; % Filter length
fs_lpf = 2*pi*fc/wc; % Low pass filter sampling frequency
n = -(N-1)/2:(N-1)/2; % Time index

fc1 = 500;
fc2 = 1200;
fs_bpf = 6000;

[h_lpf] = LPF(fc,fs_lpf,N);
[h_bpf] = BPF(fc1, fc2, fs_bpf, N);

disp('h[n] for LPF:');
disp(h_lpf);
disp('h[n] for BPF:');
disp(h_bpf);

fvtool(h_lpf);
fvtool(h_bpf);

function h = LPF(fc, fs, N)
    wc = 2 * pi * fc / fs;
    hd = zeros(1,N);

    % Calculating impulse response
    for k = 1:N
        n = k - (N+1)/2;
        if n == 0
            hd(k) = wc/pi;
        else
            hd(k) = sin(wc*n)/(pi*n);
        end
    end

    % Define Hamming window
    n1 = 0:N-1;
    WH = zeros(1, N);
    l = (n1 >= 0) & (n1 <= N-1);
    WH(l) = 0.54 - 0.46 * cos(2 * pi * n1(l) / (N-1));
```

```
    % Apply window to filter coefficients to get practical impulse response
    h = hd .* WH;
end

function h = BPF(fc1, fc2, fs, N)
    % Calculate the cutoff frequencies in radians
    wc1 = 2*pi*fc1/fs;
    wc2 = 2*pi*fc2/fs;
    hd = zeros(1,N);

    % Calculating impulse response
    for k = 1:N
        n = k - (N+1)/2;
        if n == 0
            hd(k) = (wc2 - wc1)/pi;
        else
            hd(k) = (sin(wc2*n) - sin(wc1*n))./(pi*n);
        end
    end

    % Define Hamming window
    n1 = 0:N-1;
    WH = zeros(1, N);
    l = (n1 >= 0) & (n1 <= N-1);
    WH(l) = 0.54 - 0.46 * cos(2 * pi * n1(l) / (N-1));

    % Apply window to filter coefficients to get practical impulse response
    h = hd.*WH;
end
```

Output:

h[n] vector output:

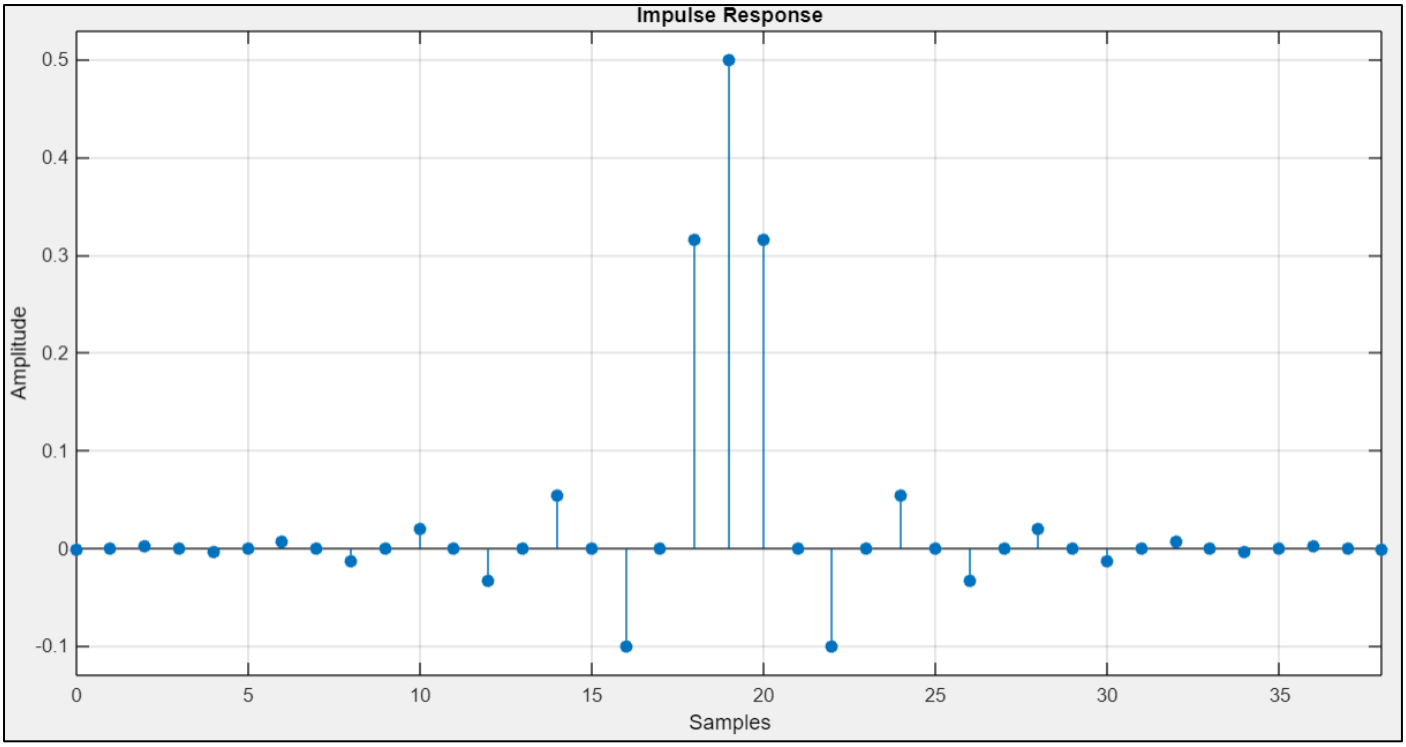
h[n] for LPF:

-0.0013 0.0000 0.0020 -0.0000 -0.0038 0.0000 0.0071 -0.0000 -0.0124 0.0000 0.0204 -0.0000 -0.0330 0.0000 0.0542 -0.0000 -0.1002 0.0000 0.3163 0.5000 0.3163 0.0000 -0.1002 -0.0000 0.0542 0.0000 -0.0330 -0.0000 0.0204 0.0000 -0.0124 -0.0000 0.0071 0.0000 -0.0038 -0.0000 0.0020 0.0000 -0.0013

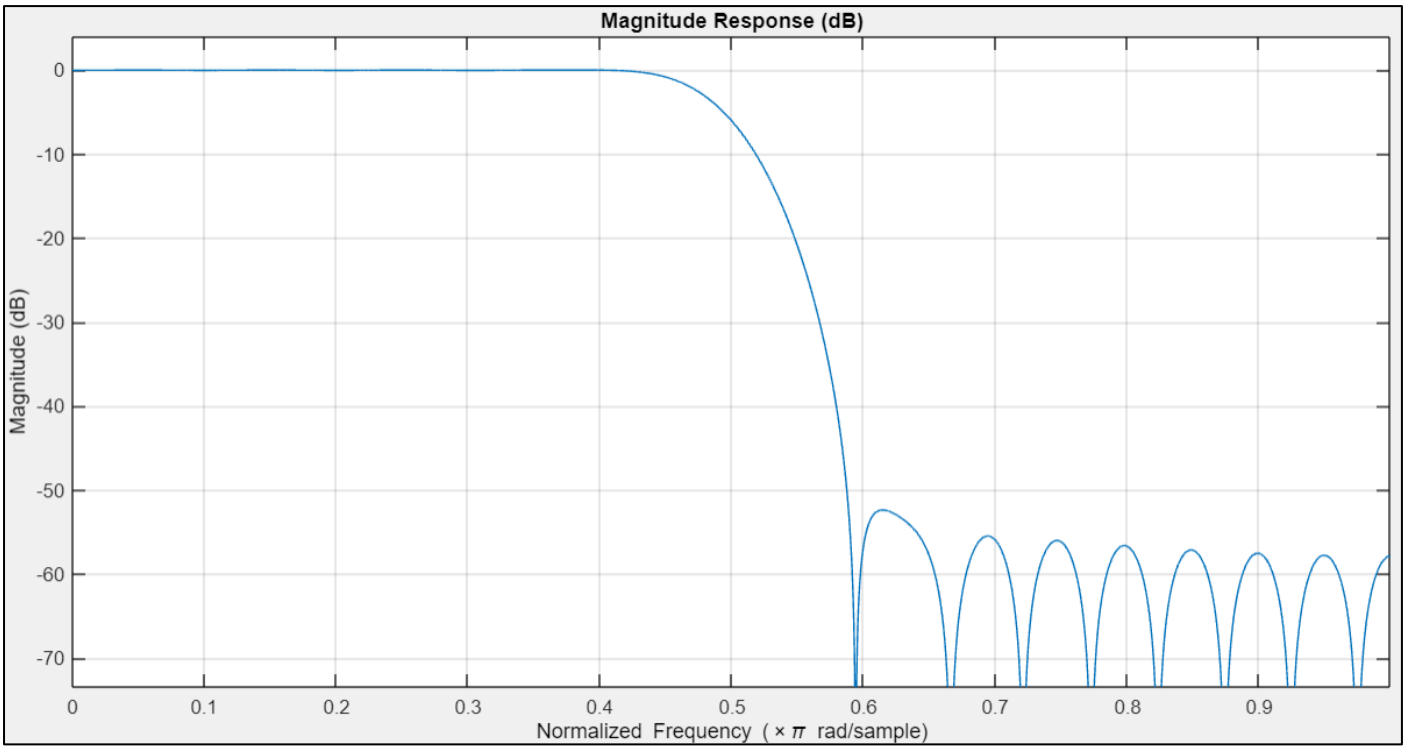
h[n] for BPF:

-0.0006 -0.0009 0.0002 0.0002 -0.0038 -0.0094 -0.0077 0.0055 0.0179 0.0138 0.0010 0.0072 0.0359 0.0399 -0.0271 -0.1306 -0.1591 -0.0432 0.1427 0.2333 0.1427 -0.0432 -0.1591 -0.1306 -0.0271 0.0399 0.0359 0.0072 0.0010 0.0138 0.0179 0.0055 -0.0077 -0.0094 -0.0038 0.0002 0.0002 -0.0009 -0.0006

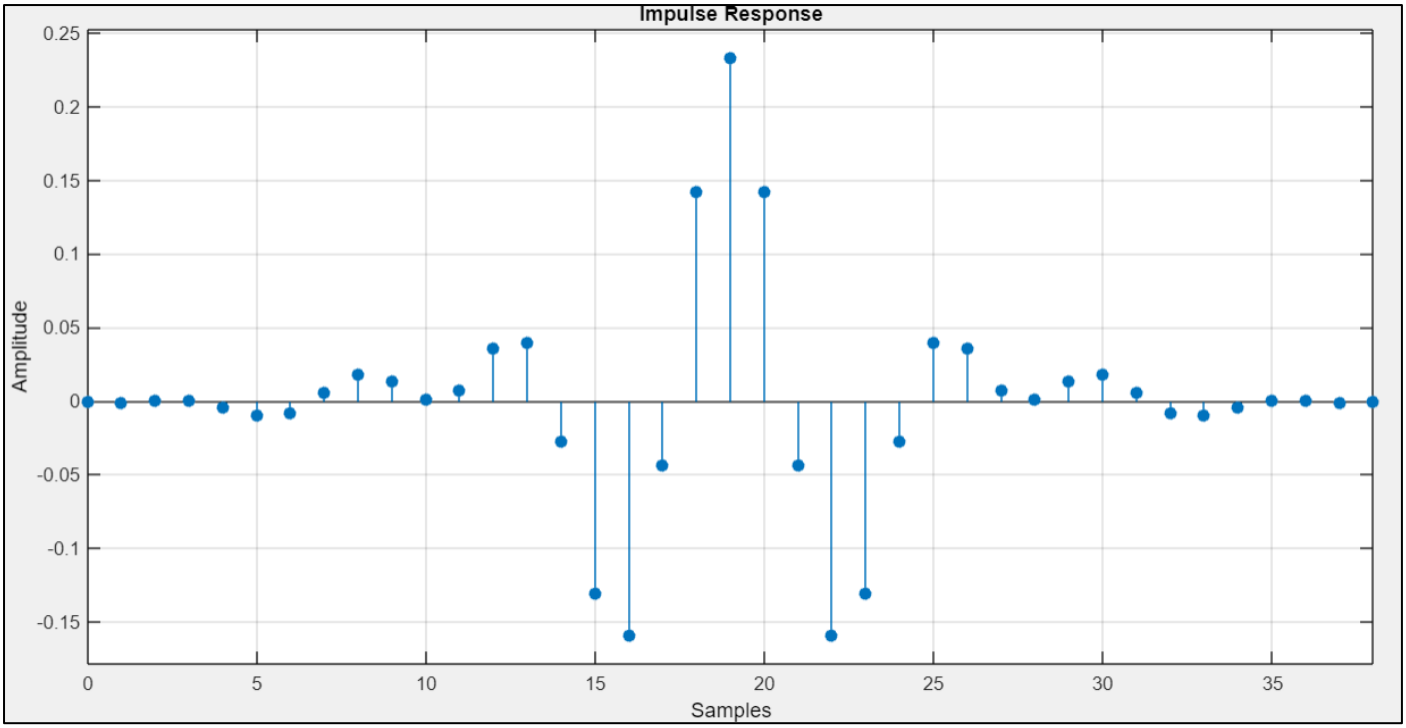
LPF Impulse Response:



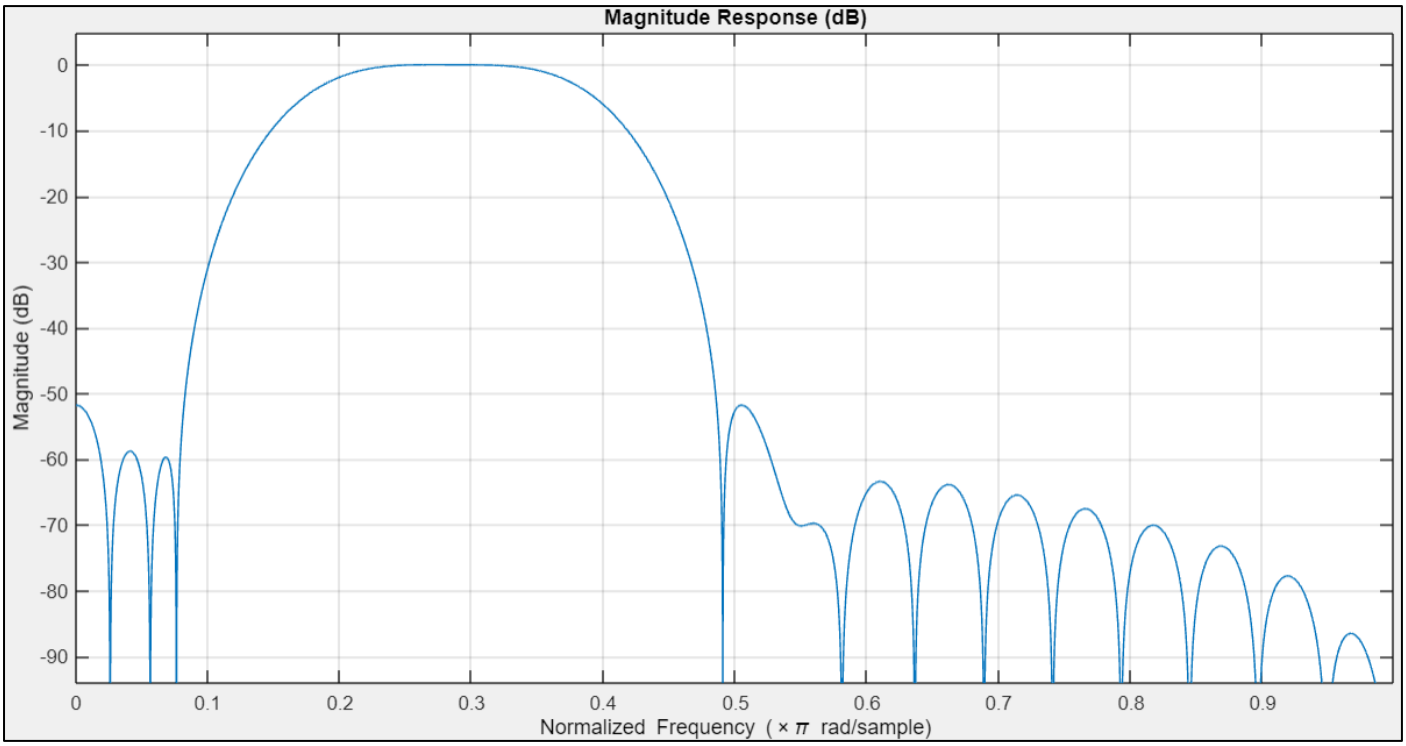
LPF Magnitude Response:



BPF Impulse Response:



BPF Magnitude Response:



C code:

File: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.1416
```

```
int main() {
   double fc = 400;
   double wc = PI / 2;
   double fs_lpf = 2 * PI * fc / wc;
   int N = 39;
   double h_lpf[N];
   double fc1 = 500;
   double fc2 = 1200;
   double fs_bpf = 6000;
   double h_bpf[N];

   LPF(fc, fs_lpf, N, h_lpf);
   BPF(fc1, fc2, fs_bpf, N, h_bpf);

   // Print LPF impulse response
   printf("LPF Impulse Response:\n");
   for (int i = 0; i < N; i++) {
      printf("%f ", h_lpf[i]);
   }
   printf("\n");
   // Print BPF impulse response
   printf("BPF Impulse Response:\n");
   for (int i = 0; i < N; i++) {
      printf("%f ", h_bpf[i]);
   }

   return 0;
}
```

File: functions.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PI 3.1416

void LPF(double fc, double fs, int N, double h_lpf[]) {
   double wc = 2 * PI * fc / fs;

   // Calculating impulse response
   for (int k = 1; k < N + 1; k++) {
      int n = k - (N + 1) / 2;
      if (n == 0) {
         h_lpf[k - 1] = wc / PI;
```

```
        } else {
            h_lpf[k - 1] = sin(wc * n) / (PI * n);
        }
    }

    // Define Hamming window
    double *WH = (double *)malloc(N * sizeof(double));
    for (int i = 0; i < N; i++) {
        WH[i] = 0.54 - 0.46 * cos(2 * PI * i / (N - 1));
    }

    // Apply window to filter coefficients to get practical impulse response
    for (int i = 0; i < N; i++) {
        h_lpf[i] *= WH[i];
    }

    free(WH); // Free dynamically allocated memory
}

void BPF(double fc1, double fc2, double fs, int N, double h_bpf[]) {
    // Calculate the cutoff frequencies in radians
    double wc1 = 2 * PI * fc1 / fs;
    double wc2 = 2 * PI * fc2 / fs;

    // Calculating impulse response
    for (int k = 1; k < N + 1; k++) {
        int n = k - (N + 1) / 2;
        if (n == 0) {
            h_bpf[k - 1] = (wc2 - wc1) / PI;
        } else {
            h_bpf[k - 1] = (sin(wc2 * n) - sin(wc1 * n)) / (PI * n);
        }
    }

    // Define Hamming window
    double *WH = (double *)malloc(N * sizeof(double));
    for (int i = 0; i < N; i++) {
        WH[i] = 0.54 - 0.46 * cos(2 * PI * i / (N - 1));
    }

    // Apply window to filter coefficients to get practical impulse response
    for (int i = 0; i < N; i++) {
        h_bpf[i] *= WH[i];
    }
```

```
    free(WH); // Free dynamically allocated memory
}
```

Output:

```
LPF Impulse Response:
-0.001340 -0.000000 0.001965 0.000000 -0.003756 -0.000000 0.007062 0.000000 -0.012358 -0.000001 0.020442 0.000001 -0.032
958 -0.000001 0.054211 0.000001 -0.100221 -0.000001 0.316312 0.500000 0.316312 -0.000001 -0.100220 0.000001 0.054211 -0.
000001 -0.032958 0.000001 0.020442 -0.000001 -0.012358 0.000000 0.007062 -0.000000 -0.003756 0.000000 0.001965 -0.000000
 -0.001340
BPF Impulse Response:
-0.000604 -0.000897 0.000172 0.000229 -0.003756 -0.009438 -0.007682 0.005538 0.017933 0.013839 0.001001 0.007228 0.03585
1 0.039941 -0.027105 -0.130573 -0.159130 -0.043180 0.142675 0.233333 0.142675 -0.043180 -0.159129 -0.130572 -0.027104 0.
039940 0.035850 0.007228 0.001001 0.013839 0.017933 0.005538 -0.007682 -0.009438 -0.003756 0.000229 0.000172 -0.000897 -
0.000604
Process returned 0 (0x0)   execution time : 1.456 s
Press any key to continue.
```

Observations and Conclusion:

- By plotting the frequency responses of the designed filters, we understand their passband and stopband characteristics. The LPF exhibits significant attenuation beyond its cutoff frequency, allowing only lower frequencies to pass. In contrast, the BPF has a narrower passband defined by two cutoff frequencies, effectively rejecting frequencies outside this range.

- Varying the filter length (N) affects the frequency response of the filters. Longer filters generally exhibit sharper transition bands and better stopband attenuation but come at the expense of increased computational complexity.

- The application of the Hamming window to the filter coefficients results in improved frequency response characteristics, particularly in terms of reducing passband ripple and stopband attenuation.