200452816
Kim, Hoon Seok
Project Report

Question 1.

Codes: Sub-functions

1. void convert()
    * unsigned char in_img[][] => complex<double> C_img[][]

Convert unsigned char in_img[][] array to complex double C_img[][] array.

2. void Rshuffle(complex<double> Coming[][])
    * Comimg[][] => S_img[][]
    * Column decimal index of each row => 8 binary strings
    * 8 binary strings => decimal number with reverse direction
    * Assign new decimal column index values => current index values

Shuffle each row's column index for FFT1D.

3. void Cshuffle(complex<double> Coming[][])
    * Comimg[][] => S_img2[][]
    * Row decimal index of each column => 8 binary strings
    * 8 binary strings => decimal number with reverse derection
    * Assign new decimal row index values => current index values

Shuffle each column's row index for FFT2D.

4. complex<double> W(int M, int u)
    * W_n^u [Euler formula] => $(\mathcal{R},\Im)$:::(Real part, Imaginary part)

Calculate FFT's W_n^u value for n, u and convert it into complex<double> format and return the value.

5. void FFT1D(int z, complex<double> Comimg[][])
    * Comimg[][] => DFT1[][] => Comimg[][]
    * Conduct FFT for one row(The same code with lecture note)

Conduct FFT for one row (256 elements) and save outputs at the same array.

6. void FFT2D(int z, complex<double> Comimg[][])
    * Comimg[][] => DFT2[][] => Comimg[][]
    * Conduct FFT for one column(modified from FFT1D())

Conduct FFT for one column (256 elements) and save outputs at the same array.

7. void DFTresult(complex<double> Comimg[][])

    * Comimg[][] => DFT[][] (center F[0]) => (unsigned char) out_img[][]
    * Send the first FFT result to origin, resort, and save to DFT[][] array
    * Rescale DFT[][], rescale(0-255) and save to unsigned char out_img[][]
    * Comimg[][] => DFT_ori[][]
    * Save original FFT result to DFT_ori[][]

Save original FFT result to DFT_ori[][] array. And center FFT result and save resorted result to DFT[][], Calculate magnitude of DFT[][] and rescale within level 0 to 255, and then save it to out_img[][] as unsigned char type[raw image].

Codes: Main program

1) Read **file** to **in_img[][]** and create output file.
2) convert(): unsigned char **in_img[][]** => complex<double> **C_img[][]**.
3) Rshuffle(C_img): column index shuffle, **C_img[][]** to **S_img[][]**.
4) FFT1D(z, S_img): 1D FFT, **S_img[][]** => **S_img[][]**.
5) Cshuffle(S_img): row index shuffle, **S_img[][]** => **S_img2[][]**.
6) FFT2D(z, S_img2): 2D FFT, **S_img2[][]** => **S_img2[][]**.
7) DFTresult(S_img2):

    Save original FFT result, **S_img2[][]** => **DFT_ori[][]**.

    Center FFT result, **S_img2[][]** => **DFT[][]**.

    Scale DFT[][] within level 0-255, and save it as unsigned char,

        DFT[][] => **out_img[][]**.

8) Save out_img[][] as a raw image file.
9) Save DFT_ori[][] as a file for frequency histogram and IFFT (Q2, Q3).

200452816
Kim, Hoon Seok
Project Report

Question 2a

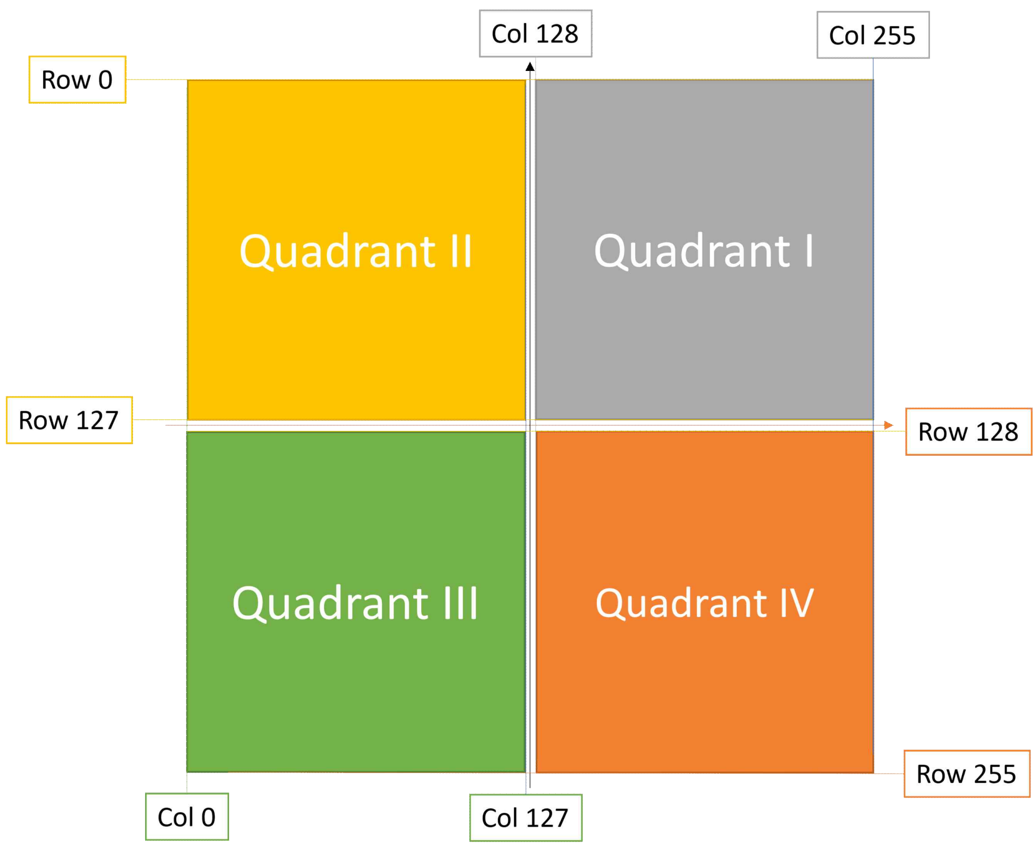
Codes: Main program

1) Read DFT **file** and load data to **DFT_ori[][]**.
2) Center original DFT results, **DFT_ori[][]** => **DFT[][]**.
3) Add all DFT magnitudes and save to **sum**.
4) Increase circle radius from 0 to $182 \approx 128 \times \sqrt{2}$ and sum all **DFT[][]** values inside the circle to array **FHc[]**.
5) Calculate FH[0~127] from FHc[], **FHc[] => FH[]**.

    FH[127] includes all corners' values not included in the largest circle.
6) Calculate FH2[0~181] from FHc[], **FHc[] => FH2[]**.

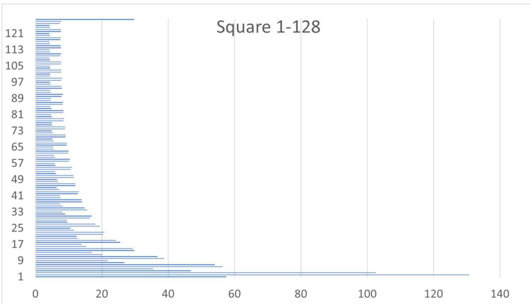
    FH2[] extend circle radius upto corners.

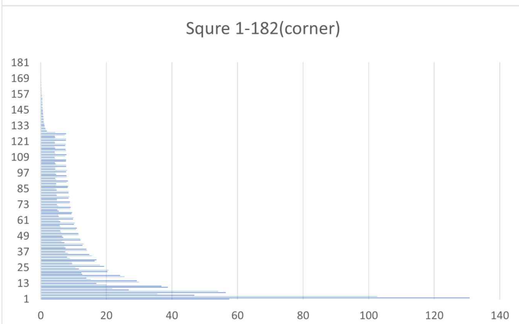
*******************Coordinate figure used for this project...

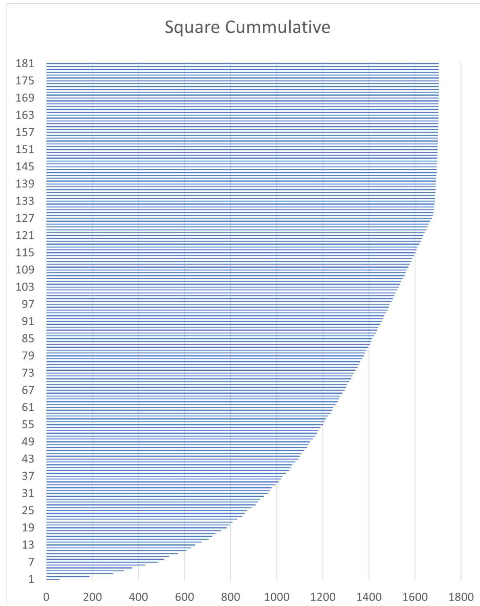


***Square Image Frequency Histogram

Square 1-128 : FH[0-127]   Square 1-182 : FH2[0-181]   Square Cumulative : FHc[0-181]

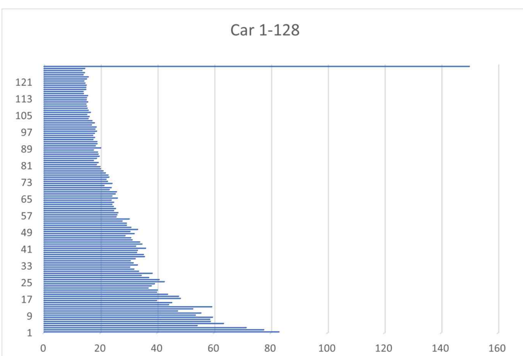


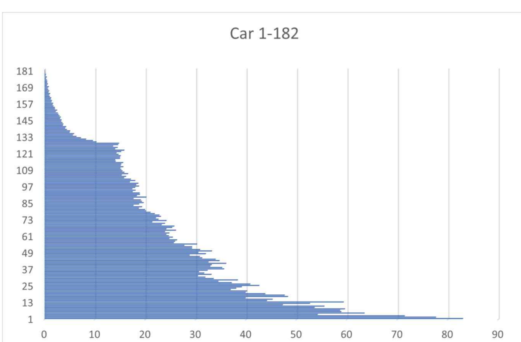


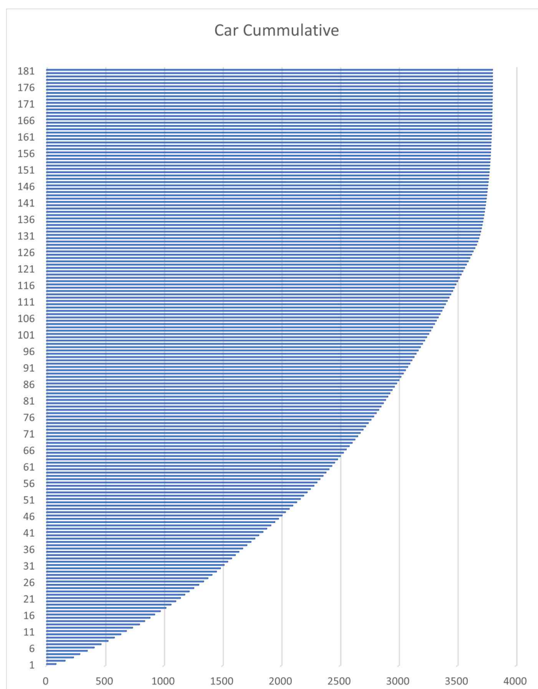


***Car Image Frequency Histogram

Car 1-128 : FH[0-127]   Car 1-182 : FH2[0-181]   Car Cumulative : FHc[0-181]







Question 2b

Codes: Sub-functions

1. void sumQ()

   * Sum of DFT elements in quadrant I, II, III, IV are Q1, Q2, Q3, Q4
   * Overall magnitude of all DFT elements is sum

Calculate sum of magnitudes of DFT array by quadrants and sum of all DFT values.

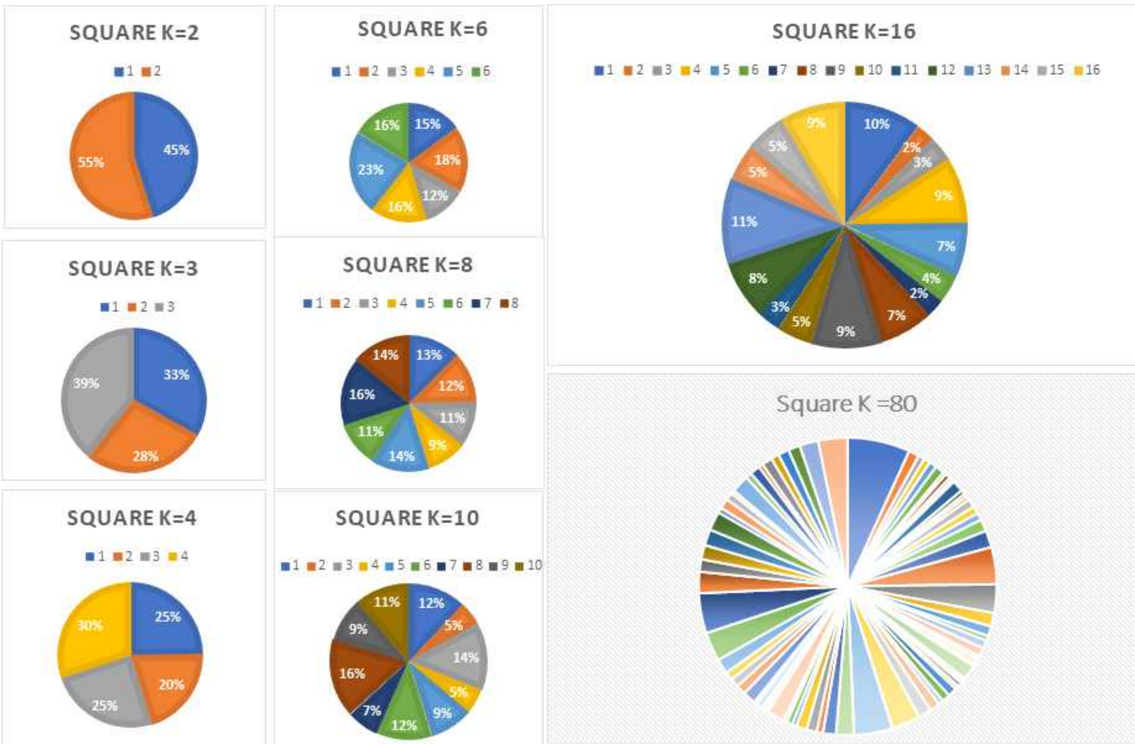
Codes: Main program

1) Read DFT **file** and load data to **DFT_ori[][]**.
2) Center original DFT results, **DFT_ori[][]** => **DFT[][]**.
3) Add all/partial DFT magnitudes, save to **sum, Q1, Q2, Q3, Q4.**

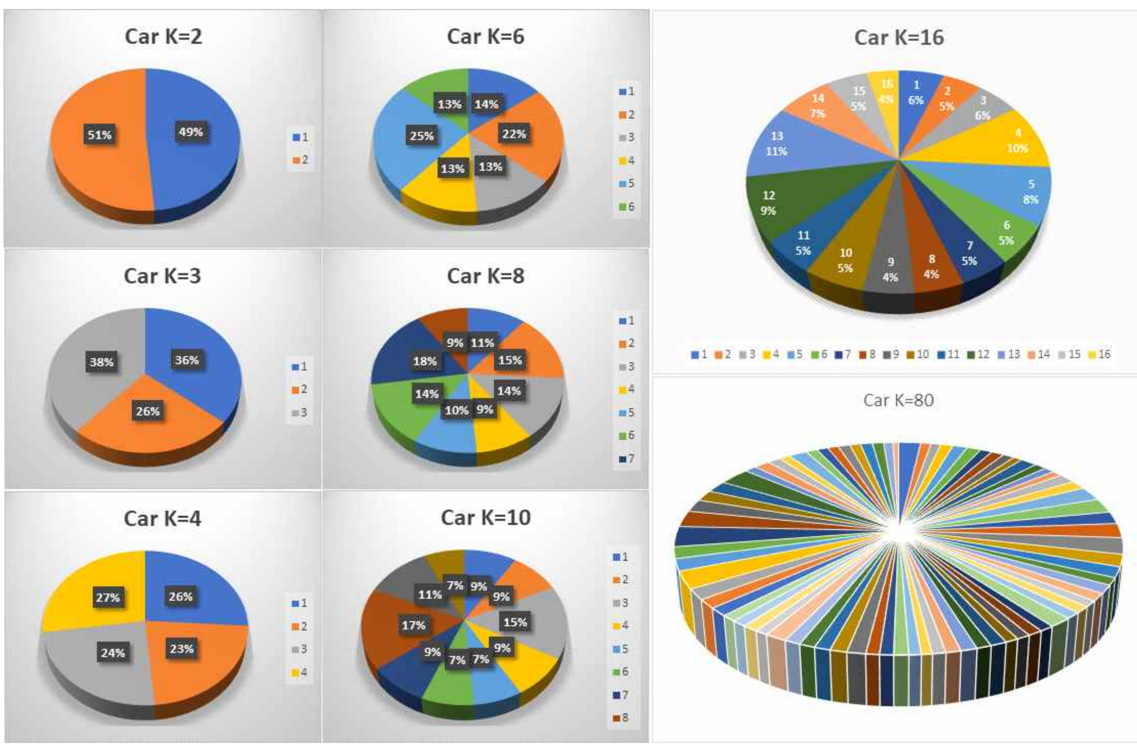
   Sum of abs(DFT[][])s in Quadrant I, II, III, IV: Q1, Q2, Q3. Q4

   Total magnitudes of abs(DFT[][])s is sum
4) Obtain cumulative angular spectrum at **AHc[]**
5) Derive angular spectrum of each sector from AHc[], **AHc[]** => **AH[]**.

***Square Image Angular Histogram::: $FH[K]$ ::: angle$\angle = \frac{2\pi}{K}\times int \sim \frac{2\pi}{K}\times(int-1)$



***Car Image Angular Histogram



Question 3

Codes: Sub-functions

1-1. LPF(int radius)::: radius = cut-off frequency

* ODFT[][] => LFT[][]

Apply ideal low pass filter to ODFT[][], output is LFT[][]

1-2. GLPF(int radius)::: radius = cut-off frequency

* ODFT[][] => LFT[][]

Apply Gaussian low pass filter to ODFT[][], output is LFT[][]

1-3. BLPF(int radius)::: radius = cut-off frequency

* ODFT[][] => LFT[][]

Apply Butterworth low pass filter to ODFT[][], output is LFT[][]

2-1. HPF(int radius)::: radius = cut-off frequency

* ODFT[][] => HFT[][]

Apply ideal high pass filter to ODFT[][], output is HFT[][]

2-2. GHPF(int radius)::: radius = cut-off frequency

* ODFT[][] => HFT[][]

Apply Gaussian high pass filter to ODFT[][], output is HFT[][]

2-3. BHPF(int radius)::: radius = cut-off frequency

* ODFT[][] => HFT[][]

Apply Butterworth high pass filter to ODFT[][], output is HFT[][]

3. void Centering(complex<double> Comimg[][])

* input array => ODFT[][]

Center original DFT values and save it to ODFT[][] array.

4. void Rshuffle(complex<double> Coming[][])
    * Comimg[][] => RDFT[][]
    * Column decimal index of each row => 8 binary strings
    * 8 binary strings => decimal number with reverse direction
    * Assign new decimal column index values => current index values

Shuffle each row's column index for IFFT1D.

5. void Cshuffle(complex<double> Coming[][])
    * Comimg[][] => CDFT[][]
    * Row decimal index of each column => 8 binary strings
    * 8 binary strings => decimal number with reverse direction
    * Assign new decimal row index values => current index values

Shuffle each column's row index for IFFT2D.

6. complex<double> invW(int M, int u)
    * $W_n{}^u$ [Euler formula] => $(\mathcal{R},\Im)$ [(Real part, Imaginary part)]

Calculate IFFT's inverse $W_n{}^u$ value for n, u and convert it into complex<double> format and return the value.

7. void IFFT1D(int z, complex<double> Comimg[][])
    * Comimg[][] => DFT1[][] => Comimg[][]
    * Conduct IFFT for one row(similar code with lecture note)

Conduct IFFT for one row (256 elements) and save outputs at the same array.

8. void IFFT2D(int z, complex<double> Comimg[][])
    * Comimg[][] => DFT2[][] => Comimg[][]
    * Conduct IFFT for one column(modified from IFFT1D())

Conduct IFFT for one column (256 elements) and save outputs at the same array.

9. void Filteredout(complex<double> Comimg[][])
    * Comimg[][] => out_img[][] (unsigned char)
    * If Comimg[][]'s element is greater than 255, set it to 255
    * Save Comimg[][] to out_img[][] as unsigned char without overflow

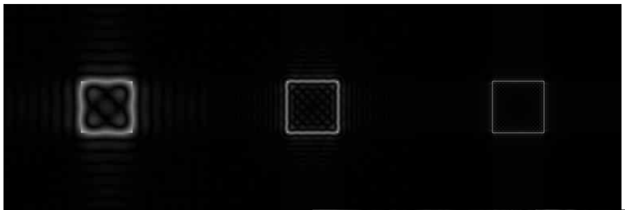
Calculate magnitude of Comimg[][] and if overflow, set level to 255, and then save it to out_img[][] as unsigned char type[raw image].

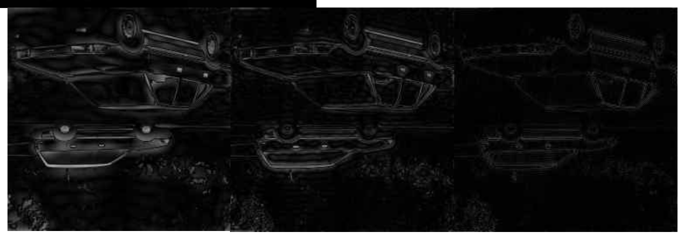
Codes: Main program

1) Read **file** to **DFT[][]** and create output image file.
2) Centering(DFT): center DFT[][], **DFT[][]** => **ODFT[][]**.
3-1) LPF(ideal), GLPF(gaussian), or BLPF(Butterworth):
    apply LPF with cut-off frequency R, **ODFT[][]** => **LFT[][]**.
    if LPF is chosen, go to 4-1)
3-2) HPF(ideal), GHPF(gaussian), or BHPF(Butterworth):
    apply HPF with cut-off frequency R, **ODFT[][]** => H**FT[][]**.
    if HPF is chosen, go to 4-2)
4-1) Centering(LFT): decenter DFT for IFFT, **LFT[][]** => **ODFT[][]**.
4-2) Centering(HFT): decenter DFT for IFFT, **HFT[][]** => **ODFT[][]**.
5) Rshuffle(ODFT): column index shuffle, **ODFT[][]** => **RDFT[][]**.
6) IFFT1D(z, RDFT): 1D IFFT for row z, **RDFT[][]** => **RDFT[][]**.
7) Cshuffle(RDFT): row index shuffle, **RDFT[][]** => **CDFT[][]**.
8) IFFT2D(z, CDFT): 2D FFT for column z, **CDFT[][]** => **CDFT[][]**.
9) Filteredout(CDFT):
    Store IFFT result as unsinged char array, **CDFT** => **out_img[][]**.
10) Save **out_img[][]** as a **raw image file**.

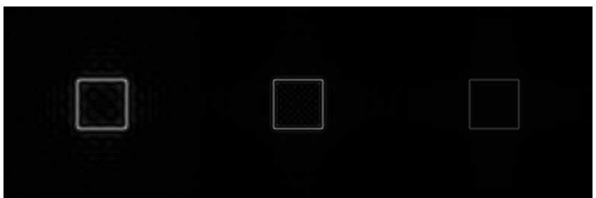
****************** HPF cut-off radius = 10 - 20 - 50

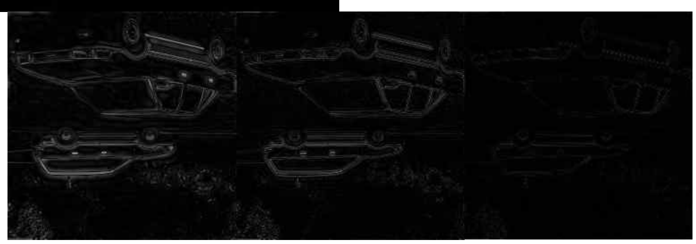
Ideal HPF 10,20,50



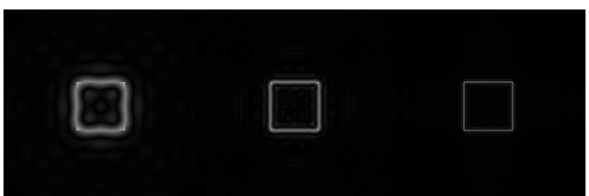


Gaussian HPF 10,20,50





Butterworth HPF 10,20,50::: n=10



****************** LPF cut-off radius = 10 - 20 - 50

Ideal LPF 10,20,50

Gaussian LPF 10,20,50

Butterworth LPF 10,20,50::: n=10