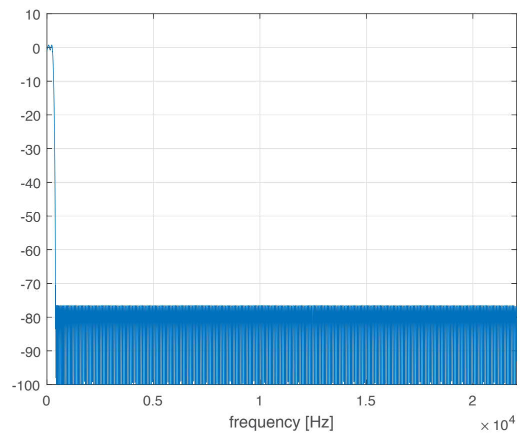
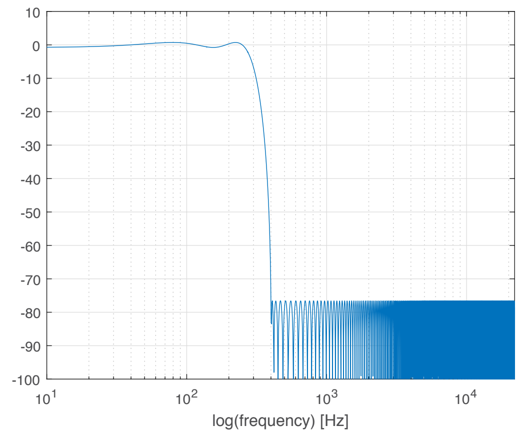
Lab 3 – Filtering in C

1. Process pre-recorded audio

- Use file `fireflyintro.wav`
- Use the impulse response in `lpf_260_400_44100_80dB.bin`
- Plot the magnitude frequency response of the filter in MATLAB.

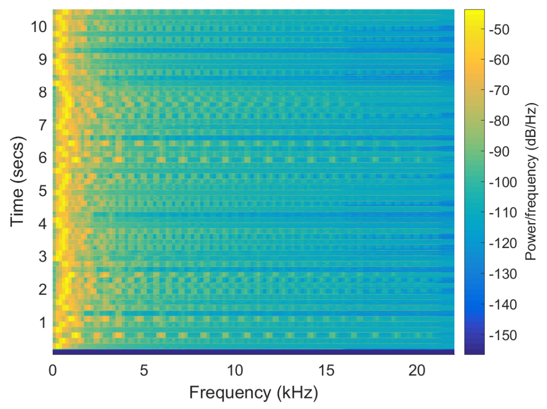
```
>> edit magnitude_plot.m

fid = fopen('lpf_260_400_44100_80db.bin','rb');
% Read in header
head = fread(fid,5,'int');
% Read in impulse response
h = fread(fid,inf,'float');
fclose(fid);
% Take a look to make sure we pulled in the right stuff
stem(h);

% Now make magnitude response plot
N = 2^14; % FFT size
% Make a frequency vector for plotting
f = [0:N-1]*44100/N;
% Compute the magnitude response
H = abs(fft(h,N)).^2;

figure(1);
plot(f,10*log10(H));
grid on;
xlim([0 44100/2]);
ylim([-100 10]);
xlabel('frequency [Hz]');

figure(2);
semilogx(f,10*log10(H));
grid on;
xlim([10 44100/2]);
ylim([-100 10]);
xlabel('log(frequency) [Hz]');
```



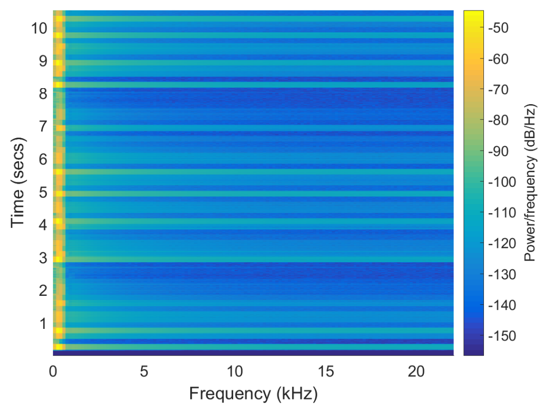


- Plot a spectrogram of the signal before and after filtering.

```
>> edit spectrogram_plot.m

[x,sample_rate] = audioread('fireflyintro.wav');

nfft = 2^8; % FFT size
overlap = round(0.8*nfft);
window = hamming(nfft);
spectrogram(x,window,overlap,nfft,sample_rate);

set(gca,'FontSize',16);
grid on;
print -dpdf spectrogram_plot(before).pdf
```



```
>> edit spectrogram_plot.m

[x,sample_rate] = audioread('fireflyintro-1.wav');

nfft = 2^8; % FFT size
overlap = round(0.8*nfft);
window = hamming(nfft);
spectrogram(x,window,overlap,nfft,sample_rate);

set(gca,'FontSize',16);
grid on;
print -dpdf spectrogram_plot(after).pdf
```



- Listen to the before and after audio and describe the difference.

After the file `fireflyintro.wav` was processed through the low-pass filter, only the lower frequencies before the 80dB cutoff frequency were preserved. Thus the filtered audio file sounded significantly lower in volume and lacked the full detail of the original audio file.

- When doing convolution on pre-recorded signals, explain the advantages of zero padding the input signal.

- Results in a longer result in the DFT domain.
- Yields an efficient method of interpolation in the DFT domain.
- Reduces discontinuity at the DFT domain boundary.

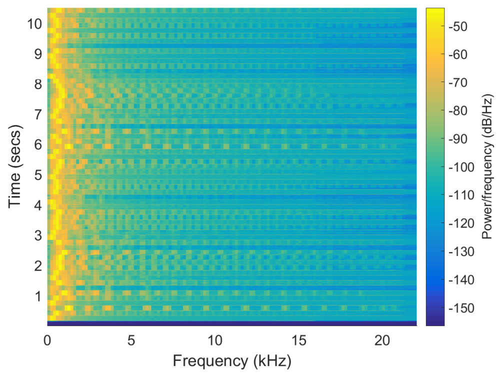
- How many zeroes are padded at the start and end of the signal?

```
Lh = 798 // length of impulse response
Lx = 463050 // length of input signal
Lz = Lx+2*(Lh-1) = 464644 // length of zero padded input
∴ (Lz-Lx)/2 = Lh-1 = 797 // number of zeroes at start and end of signal
```

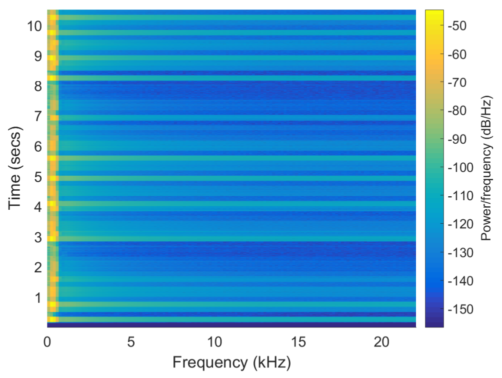
2. Process real-time audio using circular buffer

- Use file `fireflyintro.wav`
- Use the impulse response in `lpf_260_400_44100_80dB.bin`
- Plot a spectrogram of the signal before and after filtering.

```
>> edit spectrogram_plot.m

[x,sample_rate] = audioread('fireflyintro.wav');

nfft = 2^8; % FFT size
overlap = round(0.8*nfft);
window = hamming(nfft);
spectrogram(x,window,overlap,nfft,sample_rate);

set(gca,'FontSize',16);
grid on;
print -dpdf spectrogram_plot(before).pdf
```



```
>> edit spectrogram_plot.m

[x,sample_rate] = audioread('fireflyintro-2.wav');

nfft = 2^8; % FFT size
overlap = round(0.8*nfft);
window = hamming(nfft);
spectrogram(x,window,overlap,nfft,sample_rate);

set(gca,'FontSize',16);
grid on;
print -dpdf spectrogram_plot(after).pdf
```



- Listen to the before and after audio and describe the difference.
  - This should give the same result as in the first part.

After the file `fireflyintro.wav` was processed through the low-pass filter, only the lower frequencies before the 80dB cutoff frequency were preserved. Thus the filtered audio file sounded significantly lower in volume and lacked the full detail of the original audio file.

- When filtering real-time signals, explain the advantages of circular indexing.

- Uses minimal allocation of memory.
- Avoids shifting the data.

3. 2D spatial filtering for edge detection

- Write and test a C program to perform 2D convolution.
- Use the pre-recorded convolution method in which the entire image is loaded into memory and zero-padded all the way around.
- Modify the C program to convolve an input image *x* with the two point spread functions

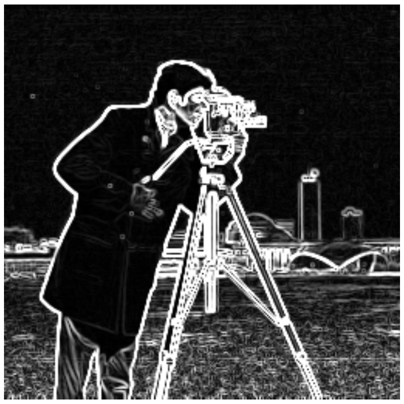
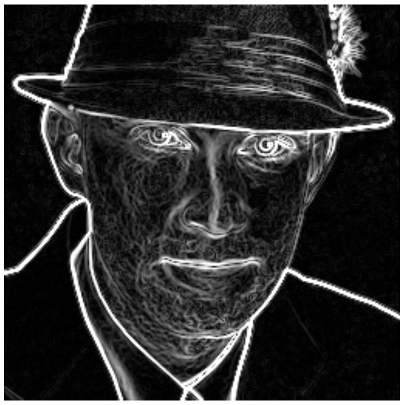
$$h_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}.$$

- Use `cameraman.png` as the input image.
- Combine the two resulting images into a single image using a root sum of squares combination.

$$y(m,n) = \sqrt{\left[y_x(m,n)\right]^2 + \left[y_y(m,n)\right]^2}, \text{ where } y_x = h_x * x \text{ and } y_y = h_y * x.$$

- Save the resulting image and view it in MATLAB.
- Explain what you see.
- Find another digital picture (one of your own, or one found on the Internet) and apply the edge detection processing to it.

  - Note: either convert the picture to grayscale before applying edge detection, or apply edge detection to each of the color channels independently.
- Prepare a document that includes the following:
  - Original and processed images
  - Code
  - Read Wikipedia's article on the "Sobel Operator" (`http://en.wikipedia.org/wiki/Sobel_operator`) and explain how edge detection works (please mention "convolution" in your discussion).
  - Explain why zero padding the input image simplifies 2D convolution.
  - How much zero padding is needed in the row and column dimension?

Original Images	**Processed Images**
	
	

The processed images are the result of the edge detection filter. The white pixels denote the detected edges.

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct
{
        int ndim; /* number of dimensions; audio=1 */
        int nchan; /* number of channels; monaural=1 stereo=2 */
        int dim0; /* length of first dimension; audio=L L=length */
```

```
	int dim1; /* length of second dimension; audio=Fs Fs=sample rate */
	int dim2; /* length of third dimension; audio=null */
} dsp_file_header;

int main(int argc,char *argv[])
{
	// Open Files
	FILE *fx,*fy;
	if(NULL == (fx = fopen("cameraman.bin","rb")))
	{
		printf("ERROR: Cannot open cameraman.bin for input.\n");
		return 0;
	}
	if(NULL == (fy = fopen("cameraman-3.bin","wb")))
	{
		printf("ERROR: Cannot open cameraman-3.bin for output.\n");
		return 0;
	}

	// Read In & Write Out Headers
	int Rx,Rh,Cx,Ch,Ry,Cy,Rz,Cz;
	dsp_file_header headx,heady;
	fread(&headx,sizeof(dsp_file_header),1,fx); // Read In Header
	Rx = headx.dim0;		// Number of Rows in Input Image
	Rh = 3;				// Number of Rows in Impulse Filter
	Cx = headx.dim1;		// Number of Columns in Input Image
	Ch = 3;				// Number of Columns in Impulse Filter
	Ry = Rx + (Rh-1);		// Length of Convolution Result for Rows
	Cy = Cx + (Ch-1);		// Length of Convolution Result for Columns
	Rz = Rx + 2*(Rh-1);		// Length of Double Zero Padded Image for Rows
	Cz = Cx + 2*(Ch-1);		// Length of Double Zero Padded Image for Columns
	memcpy(&heady,&headx,sizeof(dsp_file_header)); // Copy Input Header to Output Header
	heady.dim0 = Ry;		// Number of Rows in Output Image
	heady.dim1 = Cy;		// Number of Columns in Output Image
	fwrite(&heady,sizeof(dsp_file_header),1,fy); // Write Out Header

	// Error Checking
	if(headx.nchan > 1)
	{
		printf("ERROR: This program only processes single channel signals.\n");
		return 0;
	}

	// Memory Allocation
	float *x = (float*)calloc(sizeof(float),Rz*Cz);
	float *y = (float*)calloc(sizeof(float),Ry*Cy);
	float *yx = (float*)calloc(sizeof(float),Ry*Cy);
	float *yy = (float*)calloc(sizeof(float),Ry*Cy);

	// Read Data into Input Array
	fread(x+(Ry*Cy-Rx*Cx),sizeof(float),Rx*Cx,fx);

	// Impulse Filter
	int hx[3][3]; // Row Mask
	int hy[3][3]; // Columns Mask
	// Row Mask Matrix (Reversed)
	hx[0][0] = 1; hx[0][1] = 0; hx[0][2] = -1;
	hx[1][0] = 2; hx[1][1] = 0; hx[1][2] = -2;
	hx[2][0] = 1; hx[2][1] = 0; hx[2][2] = -1;
	// Columns Mask Matrix (Reversed)
	hy[0][0] = 1; hy[0][1] = 2; hy[0][2] = 1;
	hy[1][0] = 0; hy[1][1] = 0; hy[1][2] = 0;
	hy[2][0] = -1; hy[2][1] = -2; hy[2][2] = -1;

	// Processing
	int k,l,i,j;
	float Rtmp,Ctmp;
```

```
        for(k=0; k<Ry; k++) // Loop over rows in output image.
        {
                for(l=0; l<Cy; l++) // Loop over columns in output image.
                {
                        for(Rtmp=0.0,i=0; i<Rh; i++) // Loop over rows in impulse filter.
                        {
                                for(Ctmp=0.0,j=0; j<Ch; j++) // Loop over columns in impulse filter.
                                {
                                        // Multiply and Accumulate the Rows
                                        Rtmp += hx[i][j]*x[(k+i)*Cz+(l+j)];
                                        // Multiply and Accumulate the Columns
                                        Ctmp += hy[i][j]*x[(k+i)*Cz+(l+j)];
                                }
                        }
                        yx[k*Cy+l] = Rtmp; // Save result to output image for rows.
                        yy[k*Cy+l] = Ctmp; // Save result to output image for columns.
                }
        }
        y = sqrt(pow(yx,2)+pow(yy,2)); // Combine the rows output image with the columns output image.

        // Write Data into Output Array
        fwrite(y,sizeof(float),Ry*Cy,fy);

        // Close Files and Clear Memory
        fclose(fx);
        fclose(fy);
        free(x);
        free(y);
        free(yx);
        free(yy);
}
```

The Sobel filter works by analyzing the matrix of the image as a separate procedure in each dimension; i.e. vertical and horizontal. In each respective dimension, changes in contrast are detected by convolving the two point spread function of that dimension with the data value of that pixel from the original image. With the resulting edge detected values derived from each dimension, they are combined using the root sum of squares combination to form the final edge-filtered image.

Zero padding the input images simplifies 2D convolution by…

- Results in a longer result in the DFT domain.
- Yields an efficient method of interpolation in the DFT domain.
- Reduces discontinuity at the DFT domain boundary.

```
Rx = 256                  // Number of Rows in Input Image
Rh = 3                    // Number of Rows in Impulse Filter
Cx = 256                  // Number of Columns in Input Image
Ch = 3                    // Number of Columns in Impulse Filter
Ry = Rx+(Rh-1) = 258      // Length of Convolution Result for Rows
Cy = Cx+(Ch-1) = 258      // Length of Convolution Result for Columns
Rz = Rx+2*(Rh-1) = 260    // Length of Double Zero Padded Image for Rows
Cz = Cx+2*(Ch-1) = 260    // Length of Double Zero Padded Image for Columns
∴ Ry*Cy-Rx*Cx = 1028      // Number of Zeroes Padding the Input Image
```