NONLINEAR SIGNAL PROCESSING

Laboratory work #9

GOAL OF WORK:

- Eliminating echo from audio signal on base of Ceptrum analysis.
- Estimation of Fundamental Frequency in audio signals using complex Cepstrum
- Homomorphic filtering for cancellation of multiplicative noise in an image.

Description:

Cepstral analysis is a nonlinear signal processing technique that is applied most commonly in speech processing and homomorphic filtering.

Consider the different Matlab functions:

xhat = cceps(x) returns the complex cepstrum of the real data sequence x using the Fourier transform. The input is altered, by the application of a linear phase term, to have no phase discontinuity at $\pm\pi$ radians. That is, it is circularly shifted (after zero padding) by some samples, if necessary, to have zero phase at π radians.

[xhat,nd] = cceps(x) returns the number of samples nd of (circular) delay added to x prior to finding the complex cepstrum.

[xhat, nd,xhat1] = cceps(x) returns a second complex cepstrum, xhat1, computed using an alternative factorization algorithm. This method can be applied only to finite-duration signals.

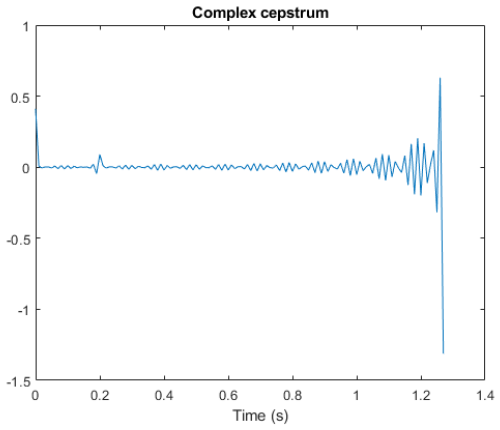
[...] = cceps(x,n) zero pads x to length n and returns the length n complex cepstrum of x.

Task 1:

- Use **cceps** function to show an echo in the signal and remove echo from signal.
- Generate a sine of frequency 45 Hz, sampled at 100 Hz. Add an echo with half the amplitude and 0.2 s later. Play (sound) given signal.
- Compute the complex cepstrum of the signal. Notice the echo at 0.2 s.
- Remove echo from signal using necessary filter type.
- Plot obtained signal signal in time anf frequency domain
- Play obtained signal.
- Load into program another audio signal. Add an echo with 0.4 amplitude and 0.3 s delay.

- Eliminate echo from this signal using homomorphic analysis.

```
% Generation of signal with echo
Fs = 100;
t = 0:1/Fs:1.27;

s1 = sin(2*pi*45*t);
s2 = s1 + 0.5*[zeros (1,20) s1(1:108)];
```



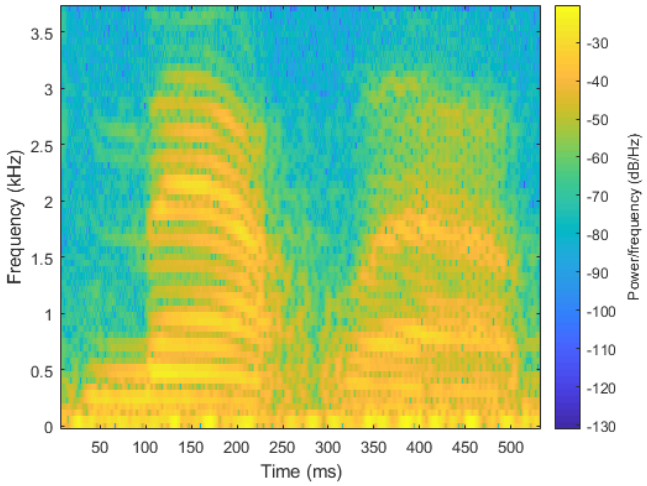
Task 2: Estimate the Fundamental Frequency for given audio signal using complex Cepstrum

Load the speech signal. The recording is of a woman saying "MATLAB". The sampling frequency is 7418 Hz. The following code loads the speech waveform, mtlb, and the sampling frequency, Fs, into the MATLAB workspace.

```
load mtlb
```

Use the spectrogram to identify a voiced segment for analysis.

```
segmentlen = 100;
noverlap = 90;
NFFT = 128;

spectrogram (mtlb, segmentlen, noverlap, NFFT,Fs,'yaxis')
```



Extract the segment from 0.1 to 0.25 seconds for analysis. The extracted segment corresponds roughly to the first vowel, /æ/, in "MATLAB".

```
dt = 1/Fs;
I0 = round(0.1/dt);
Iend = round(0.25/dt);
x = mtlb(I0:Iend);
```

Obtain the complex cepstrum.

```
c = cceps(x);
```

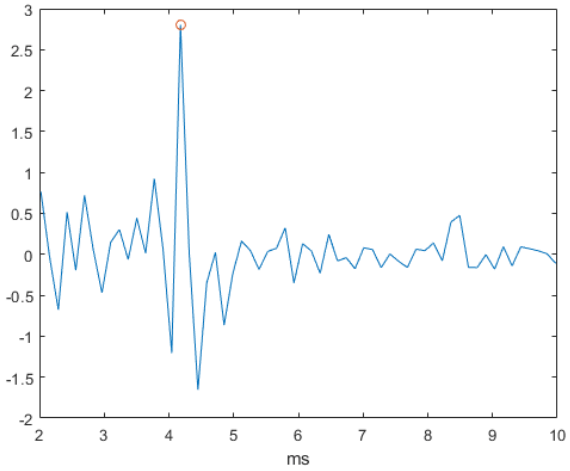
Select a time range between 2 and 10 ms, corresponding to a frequency range of approximately 100 to 500 Hz. Identify the tallest peak of the cepstrum in the selected range. Find the frequency corresponding to the peak. Use the peak as the estimate of the fundamental frequency.

```
t = 0:dt:length(x)*dt-dt;

trng = t(t>=2e-3 & t<=10e-3);
crng = c(t>=2e-3 & t<=10e-3);

[~,I] = max(crng);

fprintf('Complex cepstrum F0 estimate is %3.2f Hz.\n',1/trng(I))
```

Complex cepstrum F0 estimate is 239.29 Hz.

Plot the cepstrum in the selected time range and overlay the peak.

```
plot (trng*1e3, crng)
xlabel('ms')

hold on
plot(trng(I)*1e3, crng(I),'o')
hold off
```



Use a zero-crossing detector on a lowpass-filtered and rectified form of the vowel to estimate the fundamental frequency.

```
[b0, a0] = butter(2,325/(Fs/2));
xin = abs(x);
xin = filter (b0, a0, xin);
xin = xin-mean(xin);
x2 = zeros(length(xin),1);
x2(1: length(x)-1) = xin(2: length(x));
zc = length(find((xin>0 & x2<0) | (xin<0 & x2>0)));
F0 = 0.5*Fs*zc/length(x);
fprintf('Zero-crossing F0 estimate is %3.2f Hz.\n',F0)
```

Zero-crossing F0 estimate is 233.27 Hz.

The estimate of the fundamental frequency obtained with the complex cepstrum is 239.29 Hz and the estimate with the zero-crossing detector is 233.27 Hz.

Task 3: Homomorphic filtering for cancellation of multiplicative noise in image.

There are several image denoising/enhancement techniques that assume an additive noise model, i.e.

$$\tilde{I}(x,y) = I(x,y) + n(x,y),$$

where n is the noise signal. But there are relatively few techniques that work with other noise models, such as the multiplicative model

$$\tilde{I}(x,y) = I(x,y)\, n(x,y).$$

Homomorphic filtering is one such technique for removing multiplicative noise that has certain characteristics.

Homomorphic filtering is most commonly used for correcting non-uniform illumination in images. The illumination-reflectance model of image formation says that the intensity at any pixel, which is the amount of light reflected by a point on the object, is the product of the illumination of the scene and the reflectance of the object(s) in the scene, i.e.,

$$I(x,y) = L(x,y)\, R(x,y)$$

where I is the image, L is scene illumination, and R is the scene reflectance. Reflectance R arises from the properties of the scene objects themselves, but illumination L results from the lighting conditions at the time of image capture. To compensate for the non-uniform illumination, the key is to remove the illumination component L and keep only the reflectance component R. If we consider illumination as the noise signal (which we want to remove), this model is similar to the multiplicative noise model shown earlier.

Illumination typically varies slowly across the image as compared to reflectance which can change quite abruptly at object edges. This difference is the key to separating out the illumination component from the reflectance component. In homomorphic filtering we first transform the multiplicative components to additive components by moving to the log domain.

$$\ln(I(x,y)) = \ln(L(x,y)\, R(x,y))$$

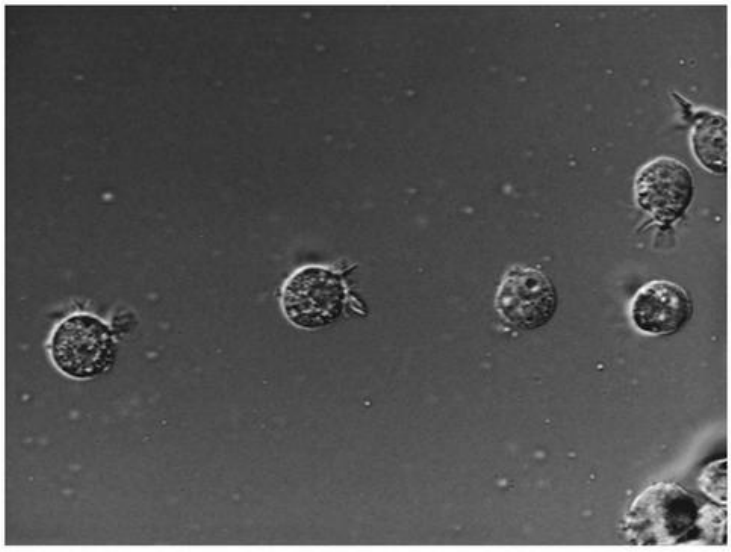
$$\ln(I(x,y)) = \ln(L(x,y)) + \ln(R(x,y))$$

Then we use a high-pass filter in the log domain to remove the low-frequency illumination component while preserving the high-frequency reflectance component. The basic steps in homomorphic filtering are shown in the diagram below:

Homomorphic Filtering

- Use an image from the Image Processing Toolbox.

```
I = imread('AT3_1m4_01.tif');
imshow(I)
```

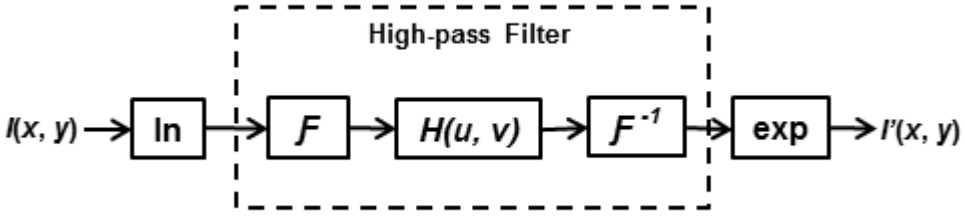


In this image the background illumination changes gradually from the top-left corner to the bottom-right corner of the image. Let's use homomorphic filtering to correct this non-uniform illumination.

- Convert the input image to the log domain. Before that we will also convert the image to floating-point type.

```
I = im2double(I);
I = log(1 + I);
```

- Provide high-pass filtering. We can do high-pass filtering in either the spatial or the spectral domain. Although they are both exactly equivalent, each domain offers some practical advantages of its own. We will do both it ways. Let's start with frequency-domain filtering. In frequency domain the homomorphic filtering process looks like:



First we will construct a frequency-domain high-pass filter. There are different types of high-pass filters you can construct, such as Gaussian, Butterworth, and Chebychev filters. We will construct a simple Gaussian high-pass filter directly in the frequency domain. In frequency domain filtering we have to careful about the *wraparound error* which comes from the fact that Discrete Fourier Transform treats a finite-length signal (such as the image) as an infinite-length periodic signal where the original finite-length signal represents one period of the signal. Therefore, there is *interference* from the non-zero parts of the adjacent copies of the signal. To avoid this, we will pad the image with zeros. Consequently, the size of the filter will also increase to match the size of the image.

```
M = 2*size(I,1) + 1;
N = 2*size(I,2) + 1;
```

Note that we can make the size of the filter (M,N) even numbers to speed-up the FFT computation, but we will skip that step for the sake of simplicity. Next, we choose a standard deviation for the Gaussian which determines the bandwidth of low-frequency band that will be filtered out.

```
sigma = 10;
```

And create the high-pass filter...

```
[X, Y] = meshgrid(1:N,1:M);
centerX = ceil(N/2);
centerY = ceil(M/2);
gaussianNumerator = (X - centerX).^2 + (Y - centerY).^2;
H = exp(-gaussianNumerator./(2*sigma.^2));
H = 1 - H;
```

Couple of things to note here. First, we formulate a low-pass filter and then subtracted it from 1 to get the high-pass filter. Second, this is a *centered* filter in that the zero-frequency is at the center.

```
imshow(H,'InitialMagnification',25)
```

We can rearrange the filter in the *uncentered* format using fftshift.

```
H = fftshift(H);
```

Next, we high-pass filter the log-transformed image in the frequency domain. First we compute the FFT of the log-transformed image with zero-padding using the fft2 syntax that allows us to simply pass in size of the padded image. Then we apply the high-pass filter and compute the inverse-FFT. Finally, we crop the image back to the original unpadded size.

```
If = fft2(I, M, N);
Iout = real(ifft2(H.*If));
Iout = Iout(1:size(I,1),1:size(I,2));
```

- Apply the exponential function to invert the log-transform and get the homomorphic filtered image.

```
Ihmf = exp(Iout) - 1;
```

- Let's look at the original and the homomorphic-filtered images together. The original image is on the left and the filtered image is on the right. If you compare the two images you can see that the gradual change in illumination in the left image has been corrected to a large extent in the image on the right.

```
imshowpair(I, Ihmf, 'montage')
```