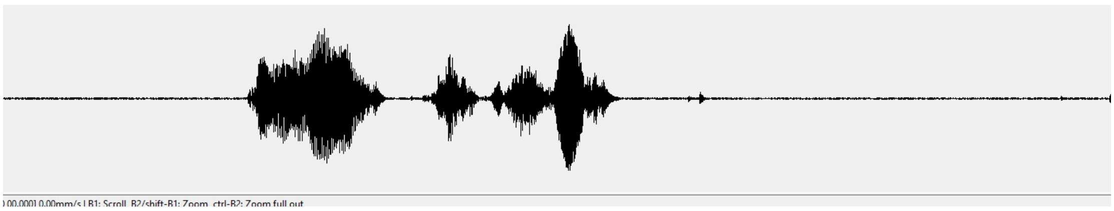
Take a recoded speech signal which you have done earlier .consider the voiced frame and unvoiced frame of your choice apply the following

1. No.of points in N-point DFT
2. Size of wind
3. Shape of window

Sol:)

Python code for the voiced and unvoiced frame of “I am abhishek”

```
import numpyimport scipy.io.wavfilefrom scipy.fftpack import dct

sample_rate, signal = scipy.io.wavfile.read('abhishekponnamofficial07(module3).wav')  signal = signal[0:int(3.5 * sample_rate)]
```



Time

```
frame_length, frame_step = frame_size * sample_rate, frame_stride * sample_rate  # Convert from seconds to samplessignal_length = len(emphasized_signal)frame_length = int(round(frame_length))frame_step = int(round(frame_step))num_frames = int(numpy.ceil(float(numpy.abs(signal_length
```

```
- frame_length)) / frame_step))  # Make sure that we have at least 1 framepad_signal_length = num_frames * frame_step + frame_lengthz = numpy.zeros((pad_signal_length - signal_length))pad_signal = numpy.append(emphasized_signal, z) # Pad Signal to make sure that all frames have equal number of samples without truncating any samples from the original signalindices = numpy.tile(numpy.arange(0, frame_length), (num_frames, 1)) + numpy.tile(numpy.arange(0, num_frames * frame_step, frame_step), (frame_length, 1)).Tframes = pad_signal[indices.astype(numpy.int32, copy=False)]


frames *= numpy.hamming(frame_length)

mag_frames = numpy.absolute(numpy.fft.rfft(frames, NFFT))  # Magnitude of the FFTpow_frames = ((1.0 / NFFT) * ((mag_frames) ** 2))
```

```
low_freq_mel = 0high_freq_mel = (2595 * numpy.log10(1 + (sample_rate / 2) / 700))  # Convert Hz to Melmel_points = numpy.linspace(low_freq_mel, high_freq_mel,
```

```
nfilt + 2)  # Equally spaced in Mel scalehz_points = (700 * (10**(mel_points / 2595) - 1))  # Convert Mel to Hzbin = numpy.floor((NFFT + 1) * hz_points / sample_rate)

fbank = numpy.zeros((nfilt, int(numpy.floor(NFFT / 2 + 1))))for m in range(1, nfilt + 1):

    f_m_minus = int(bin[m - 1])   # left
f_m = int(bin[m])             # center
f_m_plus = int(bin[m + 1])    # right

    for k in range(f_m_minus, f_m):

        fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])

    for k in range(f_m, f_m_plus):

        fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])filter_banks = numpy.dot(pow_frames, fbank.T)filter_banks = numpy.where(filter_banks == 0, numpy.finfo(float).eps, filter_banks)  # Numerical Stabilityfilter_banks = 20 * numpy.log10(filter_banks)  # dB

mfcc = dct(filter_banks, type=2, axis=1, norm='ortho')[:, 1 : (num_ceps + 1)]
```

```
(nframes, ncoeff) = mfcc.shapen =
numpy.arange(ncoeff)lift = 1 + (cep_lifter /
2) * numpy.sin(numpy.pi * n /
cep_lifter)mfcc *= lift
```

```
filter_banks -= (numpy.mean(filter_banks,
axis=0) + 1e-8)

mfcc -= (numpy.mean(mfcc, axis=0) + 1e-8)
```

Signal windowing

1, rectangular window

$$w(n)=₩left₩{₩begin{matrix} 1&&0₩leq n₩leq L-1₩₩ 0&&Other ₩end{matrix}₩right.$$

2, Hamming window (Hamming)

$$w(n)=₩left₩{₩begin{matrix} ₩frac{1}{2}(1-cos(₩frac{2₩pi n}{L-1}))&&0₩leq n₩leq L-1₩₩ 0&&Other₩end{matrix}₩right.$$

3. Haining window (Hanning)

$$w(n)=₩left₩{₩begin{matrix} 0.54-0.46cos(₩frac{2₩pi n}{L-1})&&0₩leq n₩leq L-1₩₩ 0&&Other₩end {matrix}₩right.$$

Usually, the signal is truncated and framing needs to be windowed, because the truncation has frequency domain energy leakage, and the window function can reduce the impact of truncation.

Signal framing

In the framing, there will be a partial overlap between adjacent frames, the frame length (wlen) = overlap + frame shift (inc), if the adjacent two frames do not overlap, then due to the shape of the window function, There is a loss in the edge of the intercepted speech frame, so the overlap is set. Inc is frame shift, indicating the offset of the previous frame of the next frame, fs is the sampling rate, and fn is the number of frames of a speech signal.

$$\frac{N-overlap}{inc}=\frac{N-wlen+inc}{inc}$$

The theoretical basis of signal framing, where x is the speech signal and w is the window function:

$$y(n) = \sum_{n=-N/2+1}^{N/2} x(m)w(n-m)$$

Windowing truncates similar sampling. In order to ensure that adjacent frames are not too different, usually there is a frame shift between frames, which is actually the effect of interpolation smoothing.

FRAMING CODE

```
import numpy as npimport waveimport
os#import math

 def enframe(signal, nw, inc):

    '''Convert audio signals into frames.

         Parameter meaning:

         Signal: original audio model

         Nw: the length of each frame
(here, the length of the sampling point,
that is, the sampling frequency
multiplied by the time interval)

         Inc: interval of adjacent frames
(as defined above)

    '''

    signal_length=len(signal) #Total
signal length

    if signal_length<=nw: #If the signal
length is less than the length of one
frame, the number of frames is defined as
1
```

```
        nf=1

    else: #Otherwise, calculate the total
length of the frame


nf=int(np.ceil((1.0*signal_length-
nw+inc)/inc))

    pad_length=int((nf-1)*inc+nw) #All
frames add up to the total flattened
length

    zeros=np.zeros((pad_length-
signal_length,)) #Not enough length to
fill with 0, similar to the extended
array operation in FFT


pad_signal=np.concatenate((signal,zeros))
#The signal after the padding is recorded
as pad_signal


indices=np.tile(np.arange(0,nw),(nf,1))+n
p.tile(np.arange(0,nf*inc,inc),(nw,1)).T
#Equivalent to extracting the time points
of all frames to obtain a matrix of nf*nw
length


indices=np.array(indices,dtype=np.int32)
#Convert indices to matrix
```

```
    frames=pad_signal[indices] #Get the
frame signal

#         Win=np.tile(winfunc(nw),(nf,1))
#windowWindow function, here defaults to
1

#         Return frames*win #return frame
signal matrix

    return framesdef wavread(filename):

    f = wave.open(filename,'rb')

    params = f.getparams()

    nchannels, sampwidth, framerate,
nframes = params[:4]

    strData = f.readframes(nframes)#Read
audio, string format

    waveData =
np.fromstring(strData,dtype=np.int16)#Con
vert a string to an int    f.close()

    waveData =
waveData*1.0/(max(abs(waveData)))#Wave
amplitude normalization

    waveData =
np.reshape(waveData,[nframes,nchannels]).
T
```

```
    return waveData

filepath = "./data/" #Add path

dirname= os.listdir(filepath) #Get all
the file names under the folder

filename = filepath+dirname[0]

data = wavread(filename)

nw = 512

inc = 128

Frame = enframe(data[0], nw, inc)
```

VOICED FRAMING WITHOUT WINDOWING

```
def enframe(signal, nw, inc, winfunc):

    '''Convert audio signals into frames.

        Parameter meaning:

        Signal: original audio model
```

```
        Nw: the length of each frame
(here, the length of the sampling point,
that is, the sampling frequency
multiplied by the time interval)

        Inc: interval of adjacent frames
(as defined above)

    '''

    signal_length=len(signal) #Total
signal length

    if signal_length<=nw: #If the signal
length is less than the length of one
frame, the number of frames is defined as
1

        nf=1

    else: #Otherwise, calculate the total
length of the frame


nf=int(np.ceil((1.0*signal_length-
nw+inc)/inc))

    pad_length=int((nf-1)*inc+nw) #All
frames add up to the total flattened
length

    zeros=np.zeros((pad_length-
signal_length,)) #Not enough length to
```

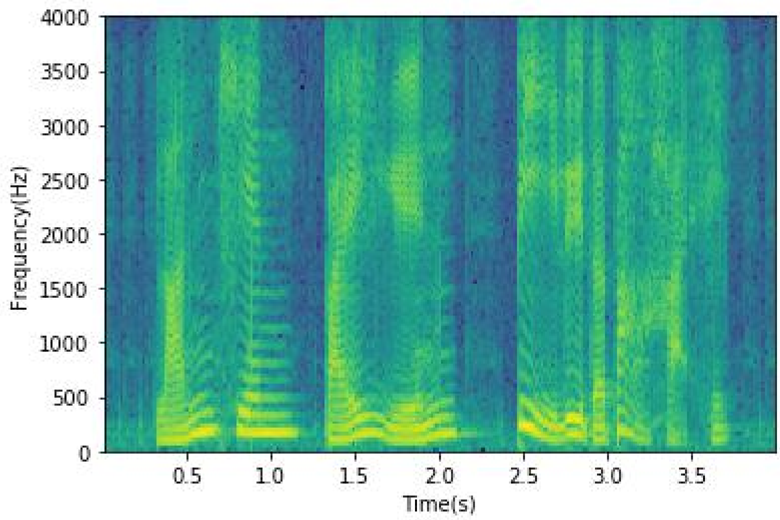
```
fill with 0, similar to the extended
array operation in FFT


pad_signal=np.concatenate((signal,zeros))
#The signal after the padding is recorded
as pad_signal


indices=np.tile(np.arange(0,nw),(nf,1))+n
p.tile(np.arange(0,nf*inc,inc),(nw,1)).T
#Equivalent to extracting the time points
of all frames to obtain a matrix of nf*nw
length


indices=np.array(indices,dtype=np.int32)
#Convert indices to matrix

    frames=pad_signal[indices] #Get the
frame signal

    win=np.tile(winfunc,(nf,1))  #Window
window function, here defaults to 1

    return frames*win   #Return frame
signal matrix
```

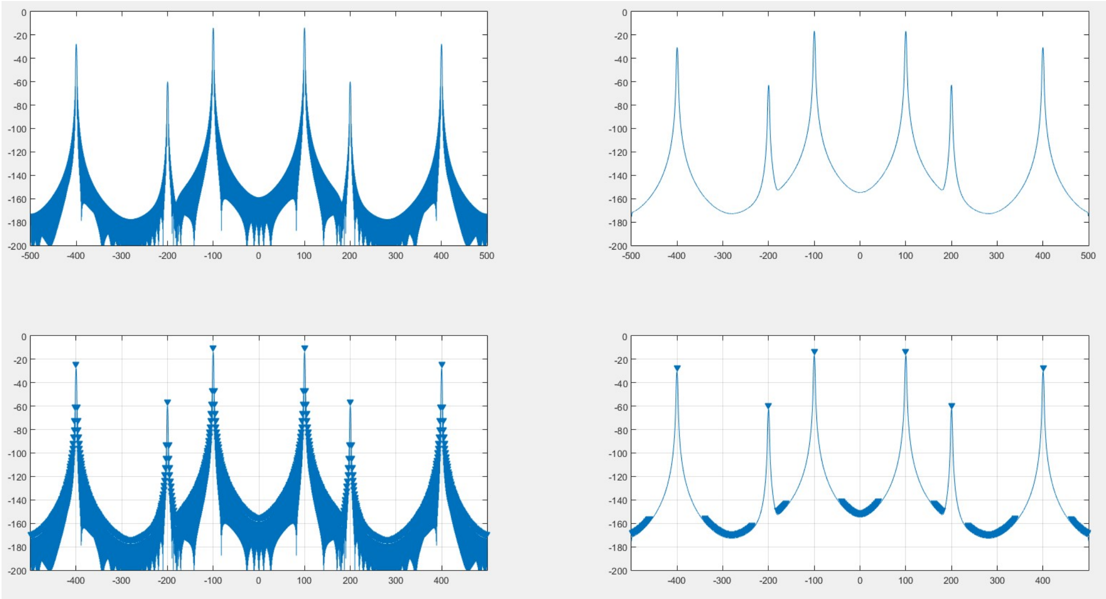
WINDOWED FRAMING

```
import numpy as npimport
matplotlib.pyplot as pltfrom scipy.io
```

```
import wavfilefrom python_speech_features
import mfcc, logfbank

#  Read input audio file

sampling_freq, audio =
wavfile.read("input_freq.wav")

#  Extract MFCC and filter bank features

mfcc_features = mfcc(audio, sampling_freq)

filterbank_features = logfbank(audio,
sampling_freq)

print('\nMFCC:\nNumber of windows =',
mfcc_features.shape[0])print('Length of
each feature =',
mfcc_features.shape[1])print('\nFilter
bank:\nnumber of windows =',
filterbank_features.shape[0])print('Lengt
h of each feature =',
filterbank_features.shape[1])

#  Draw a feature map to visualize the
MFCC. Transpose the matrix so that the
time domain is horizontal

mfcc_features = mfcc_features.T

plt.matshow(mfcc_features)
```

```python
plt.title('MFCC')#  Visualize filter bank features. Transpose the matrix so that the time domain is horizontal

filterbank_features = filterbank_features.T

plt.matshow(filterbank_features)

plt.title('Filter bank')



plt.show()
```





left is the original spectrum generated with a hanning window. Right is the spectrum convoluted by the DFT of a hanning window.