

# Solution Exercise 1: Short-time features

```
1 import numpy as np
2 import scipy.signal.windows as windows
3 import matplotlib.pyplot as plt
4 import sys
5 import math
6
7 def compute_short_time_energy(signal, frame_length, hop_size, window_type='
    rectangular'):
8     """Compute Short-Time Energy (STE) with a given window type."""
9     num_frames = int(np.floor((len(signal)-frame_length)/hop_size)+1)
10    print(f'num_frames: ', num_frames)
11    energy = np.zeros(num_frames)
12
13    if window_type == 'rectangular':
14        window = np.ones(frame_length)
15    elif window_type == 'hamming':
16        window = windows.hamming(frame_length)
17    elif window_type == 'hann':
18        window = windows.hann(frame_length)
19    else:
20        raise ValueError("Unsupported window type. Choose 'rectangular' or '
            hamming'.")
21
22    for i in range(num_frames):
23        start = i * hop_size
24        end = min(start + frame_length, len(signal))
25        frame = np.zeros(frame_length)
26        frame[:end-start] = signal[start:end] # Zero-padding if needed
27        frame = frame * window
28        energy[i] = np.sum(frame ** 2)
29    return energy
```

# Solution Exercise 1: Short-time features

```
1 def compute_short_time_magnitude(signal, frame_length, hop_size, window_type='
    rectangular'):
2     """Compute Short-Time Energy (STE) with a given window type."""
3     num_frames = int(np.floor((len(signal)-frame_length)/hop_size)+1)
4     print(f'num_frames: ', num_frames)
5     magnitude = np.zeros(num_frames)
6
7     if window_type == 'rectangular':
8         window = np.ones(frame_length)
9     elif window_type == 'hamming':
10        window = windows.hamming(frame_length)
11    elif window_type == 'hann':
12        window = windows.hann(frame_length)
13    else:
14        raise ValueError("Unsupported window type. Choose 'rectangular' or '
            hamming'.")
15
16    for i in range(num_frames):
17        start = i * hop_size
18        end = min(start + frame_length, len(signal))
19        frame = np.zeros(frame_length)
20        frame[:end-start] = signal[start:end] # Zero-padding if needed
21        frame = np.abs(frame) * window # Element-wise multiplication
22        magnitude[i] = np.sum(frame)
23    return magnitude
```

# Solution Exercise 1: Short-time features

```
1 def compute_short_time_zero_crossing(signal, frame_length, hop_size, window_type
2   = 'rectangular'):
3     """Compute Short-Time Zero Crossing (STZ) with a given window type."""
4     num_frames = int(np.floor((len(signal)-frame_length)/hop_size)+1)
5     zc = np.zeros(num_frames)
6
7     if window_type == 'rectangular':
8         window = np.ones(frame_length)
9     elif window_type == 'hamming':
10        window = windows.hamming(frame_length)
11    elif window_type == 'hann':
12        window = windows.hann(frame_length)
13    else:
14        raise ValueError("Unsupported window type. Choose 'rectangular' or '
15        hamming'.")
16
17    for i in range(num_frames):
18        start = i * hop_size
19        end = min(start + frame_length, len(signal))
20        frame = np.zeros(frame_length)
21        frame[:end-start] = signal[start:end] # Zero-padding if needed
22        frame = frame*window
23        frame = 0.5*np.abs(np.diff(np.sign(frame)))
24        zc[i] = np.sum(frame)
25    return zc
```

# Solution Exercise 1: Short-time features

```
1 def main():
2     if len(sys.argv) < 2:
3         print("Usage: python script.py <wavfile>")
4         sys.exit(1)
5
6     # Load the WAV file
7     wav_file = sys.argv[1]
8     import librosa
9     signal, sample_rate = librosa.load(wav_file, sr=None)
10    print(signal.shape)
11    if signal.ndim > 1:
12        signal = signal[:, 0] # Convert to mono if stereo
13
14    frame_length = int(0.05 * sample_rate) # 25 ms window
15    hop_size = 1 #int(0.01 * sample_rate) # 10 ms hop size
16
17    # Compute STE for rectangular and Hamming windows
18    ste_mag = compute_short_time_magnitude(signal, frame_length, hop_size, '
19        hamming')
20    ste_eng = compute_short_time_energy(signal, frame_length, hop_size, 'hamming
21        ')
22    ste_zcr = compute_short_time_zero_crossing(signal, frame_length, hop_size, '
23        rectangular')
24
25    t = np.linspace(0, ste_eng.shape[0]/sample_rate, len(ste_eng))
26
27    plt.figure(figsize=(10, 5))
```

# Solution Exercise 1: Short-time features

```
1 time = np.linspace(0, len(signal)/sample_rate, len(signal))
2 plt.subplot(4, 1, 1)
3 plt.plot(time, signal, color='black')
4 plt.title('Speech Signal')
5 plt.xlabel('time in sec')
6 plt.ylabel('Amplitude')
7 #plt.xlim(0,math.ceil(len(signal/sample_rate)))
8 plt.ylim(np.min(signal),np.max(signal))
9 plt.legend()
10
11 plt.subplot(4, 1, 2)
12 plt.plot(t, ste_eng, color='red')
13 plt.title('Short-Time Energy with Hamming Window')
14 plt.xlabel('Frame Index')
15 plt.ylabel('Energy')
16 plt.xlim(0,math.ceil(ste_mag.shape[0]/sample_rate))
17 plt.ylim(0,0.5)
18 plt.legend()
19
20 # Plot results
21 plt.subplot(4, 1, 3)
22 plt.plot(t, ste_mag, color='blue')
23 plt.title('Short-Time Magnitude with Hamming Window')
24 plt.xlabel('Frame Index')
25 plt.ylabel('Magnitude')
26 plt.xlim(0,math.ceil(ste_mag.shape[0]/sample_rate))
27 plt.ylim(0,np.max(ste_mag))
28
29 # Plot results
30 plt.subplot(4, 1, 4)
31 plt.plot(t, ste_zcr, color='blue')
32 plt.title('Short-Time Zero Crossing with Rectangular Window')
33 plt.xlabel('Frame Index')
34 plt.ylabel('ZCR')
35 plt.xlim(0,math.ceil(ste_zcr.shape[0]/sample_rate))
```

## Solution Exercise 2: Pitch Detection with Autocorrelation

```
1     import numpy as np
2     import librosa
3     import matplotlib.pyplot as plt
4     import scipy.signal.windows as windows
5     from matplotlib.ticker import MultipleLocator
6
7
8     def autocorrelation(frame):
9         """
10        Compute the autocorrelation of a frame.
11        """
12        frame = frame - np.mean(frame) # Remove DC component
13        result = np.correlate(frame, frame, mode='full')
14        return result[result.size // 2:] # positive value only
15
16     def compute_pitch(signal, sr, frame_size=1024, hop_size=512, fmin=50, fmax=500):
17         """
18        Compute pitch using short-time autocorrelation function.
19        """
20        pitches = []
21        times = []
22
23        for i in range(0, len(signal) - frame_size, hop_size):
24            frame = signal[i:i + frame_size]
25            frame = frame * windows.hamming(frame_size)
26            acf = autocorrelation(frame)
27
28            # Find the first peak after lag 0
29            min_lag = sr // fmax
30            max_lag = sr // fmin
```

## Solution Exercise 2: Pitch Detection with Autocorrelation

```
1      acf[:min_lag] = 0 # Ignore low lags
2      peak_index = np.argmax(acf[:max_lag])
3
4      if acf[peak_index] > 0:
5          pitch = sr / peak_index
6      else:
7          pitch = 0 # Unvoiced frame
8
9      pitches.append(pitch)
10     times.append(i / sr)
11
12     return np.array(times), np.array(pitches)
```

## Solution Exercise 2: Pitch Detection with Autocorrelation

```
1      # Load an example audio file
2      signal, sr = librosa.load('./audio/ih-pout-410Hz-8kHz.wav', sr=None)
3      # Compute pitch
4      times, pitches = compute_pitch(signal, sr, frame_size=240, hop_size=80)
5      # Plot pitch track
6      plt.figure(figsize=(10, 4))
7      plt.subplot(2, 1, 1)
8      plt.plot(times, pitches, label='Estimated Pitch', color='b', marker='o')
9      print(times)
10     plt.xlabel('Time (s)')
11     plt.ylabel('Pitch (Hz)')
12     plt.title('Pitch Estimation using Autocorrelation')
13     plt.legend()
14     plt.grid(True)
15     plt.subplot(2, 1, 2)
16     plt.plot(np.linspace(0, len(signal)/sr, len(signal)), signal, label='Speech
        Signal', color='r')
17     plt.xlabel('Time (s)')
18     plt.ylabel('Pitch (Hz)')
19     plt.legend()
20     plt.grid(True)
21     plt.show()
```



## Solution Exercise 3: VUS detector

```
1 import numpy as np
2 import librosa
3 import matplotlib.pyplot as plt
4 import scipy.signal.windows as windows
5 from matplotlib.ticker import MultipleLocator
6 from scipy.interpolate import interp1d
7 import scipy.signal as signal
8 from matplotlib.ticker import MaxNLocator
9 def vus(s, frame_size, hop_size):
10     # Remove DC component
11     s = s - np.mean(s)
12     # Signal length
13     D = len(s)
14     # Frame Length
15     L = frame_size
16     # Frame shift
17     U = hop_size
18     # Window type
19     win = windows.hamming(L)
20     # Number of frames
21     Nfr = int(np.floor((D-L)/U)+1)
22     # Memory allocation (for speed)
23     En = np.zeros(Nfr)
24     ZCr = np.zeros(Nfr)
25     T = np.zeros(Nfr) # Next analysis time instants
26     for i in range(Nfr):
27         start = i * U
28         end = start + L
29         frame = np.zeros(L)
30         frame[start:end-start] = s[start:end]*win
31         En[i] = np.sum(frame**2)/L # it is defined as an average energy in the
                                   # frame.
32         ZCr[i] = np.sum(0.5*np.abs(np.diff(np.sign(frame))))
33         T[i] = L/2 + i*U # Next analysis time instant
```

## Solution Exercise 3: VUS detector

```
1      # THRESHOLDS
2      Ethres = np.mean(En)/2
3      ZCRthres = (3/2)*np.mean(ZCr) - 0.3*np.std(ZCr)
4
5      VUS = np.zeros(Nfr)
6
7      # Classification for each frame
8      for i in range(Nfr):
9          if En[i] > Ethres:
10             # Voiced
11             VUS[i] = 1.0
12         elif ZCr[i] < ZCRthres:
13             # SILENCE
14             VUS[i] = 0.0
15         elif ZCr[i] > ZCRthres:
16             # UNVOICED
17             VUS[i] = 0.5
18
19     return T, VUS
```

## Solution Exercise 3: VUS detector

```
1      s, sr = librosa.load('./audio/H.22.16k.wav', sr=None)
2      s = s - np.mean(s)
3      frame_rate = 0.01
4      frame_length = 0.03
5      frame_size = int(frame_length*sr)
6      hop_size = int(frame_rate*sr)
7      # Define frames
8      # Peak normalize the audio to range [-1, 1]
9      s = s / np.max(np.abs(s))
10     T, VUS_values = vus(s, frame_size, hop_size)
11     T, index = np.unique(T, return_index=True)
12     new_T = np.arange(0, len(s), 1) # New points for interpolation (1:1:D in MATLAB
13                                     )
14     interpolator = interp1d(T, VUS_values[index], kind='linear', fill_value=0,
15                             bounds_error=False)
16     VUS_i = interpolator(new_T)
17     t = np.linspace(0, len(s)/sr, len(s))
18     plt.plot(t, s, label='Speech Signal', color='r')
19     plt.plot(t, VUS_i, label='VUS', color='b')
20
21     plt.xlabel('Time (s)')
22     plt.ylabel('Amplitude')
23     plt.ylim(-1.0, +1.0)
24     plt.legend()
25     plt.grid(True)
26     plt.title("VUS discriminator")
27     plt.show()
```

## Solution Exercise 4: Pitch Tracker

```
1  import numpy as np
2  import librosa
3  import matplotlib.pyplot as plt
4  import scipy.signal.windows as windows
5  from matplotlib.ticker import MultipleLocator
6  from scipy.interpolate import interp1d
7  from scipy.interpolate import CubicSpline
8  import scipy.signal as signal
9  from matplotlib.ticker import MaxNLocator
10
11 def vus(s, frame_size=1024, hop_size=512):
12     # Remove DC component
13     s = s - np.mean(s)
14     # Signal length
15     D = len(s)
16     # Frame Length
17     L = frame_size
18     # Frame shift
19     U = hop_size
20     # Window type
21     win = windows.hamming(L)
22     # Number of frames
23     Nfr = int(np.floor((D-L)/U)+1)
24     # Memory allocation (for speed)
25     En = np.zeros(Nfr)
26     ZCr = np.zeros(Nfr)
27     T = np.zeros(Nfr) # Next analysis time instants
28     for i in range(Nfr):
29         start = i * U
30         end = start + L
31         frame = np.zeros(L)
32         frame[:end-start] = s[start:end]*win
33         En[i] = np.sum(frame**2)/L
34         ZCr[i] = np.sum(0.5*np.abs(np.diff(np.sign(frame))))
```

## Solution Exercise 4: Pitch Tracker

```
1  T[i] = L/2 + i*U # Next analysis time instant
2
3  # THRESHOLDS
4  Ethres = np.mean(En)/1.2
5  ZCRthres = (3/2)*np.mean(ZCr) - 0.3*np.std(ZCr)
6  VUS = np.zeros(Nfr)
7  # Classification for each frame
8  for i in range(Nfr):
9      if En[i] > Ethres:
10         # Voiced
11         VUS[i] = 1.0
12     elif ZCr[i] < ZCRthres:
13         # SILENCE
14         VUS[i] = 0.0
15     elif ZCr[i] > ZCRthres:
16         # UNVOICED
17         VUS[i] = 0.5
18  return VUS
```

## Solution Exercise 4: Pitch Tracker

```
1  def acf_peak_picking(frame, sr, fmin=70, fmax=500):
2      L = len(frame)
3
4      np.set_printoptions(precision=4, suppress=False)
5      corr = np.correlate(frame, frame, mode='full')
6      pos_corr = corr[corr.size // 2:] # positive axes only
7
8      up_thresh = 1/fmin
9      low_thresh = 1/fmax
10
11     # Find peaks in the autocorrelation function
12     peaks, _ = signal.find_peaks(pos_corr)
13     pks_value = pos_corr[peaks]
14     locs = peaks/sr # Convert location of peaks to time values
15
16     valid_locs = locs[(locs >= low_thresh) & (locs <= up_thresh)]
17     valid_peaks = pks_value[(locs >= low_thresh) & (locs <= up_thresh)]
18     argmax = np.argmax(valid_peaks)
19     f_0 = 1/valid_locs[argmax]
20     return f_0
```

## Solution Exercise 4: Pitch Tracker

```
1 def compute_pitch(s, sr, frame_size=240, hop_size=80, fmin=50, fmax=500):
2     """
3     Compute pitch using short-time autocorrelation function.
4     """
5
6     VUS = vus(s, frame_size=frame_size, hop_size=hop_size)
7     s = s-np.mean(s) # Remove DC component
8     # Compute pitch
9     Nfr = int(np.floor((len(s)-frame_size)/hop_size)+1)
10    f_acf = np.zeros(Nfr)
11    T = np.zeros(Nfr)
12
13    for i in range(Nfr):
14        start = i*hop_size
15        end = start + frame_size
16        frame = np.zeros(frame_size)
17        frame = s[start:end]*windows.hamming(frame_size)
18        if VUS[i] == 1.0:
19            f_acf[i] = acf_peak_picking(frame, sr)
20        else:
21            f_acf[i] = 0
22        T[i] = frame_size/2 + i*hop_size
23    return T, f_acf
```

## Solution Exercise 4: Pitch Tracker

```
1 import soundfile as sf
2 s, sr = sf.read('./audio/H.22.16k.wav')
3
4 # Peak normalize the audio to range [-1, 1]
5 s = s / np.max(np.abs(s))
6 T, f_acf = compute_pitch(s, sr, frame_size=240, hop_size=80)
7 T, index= np.unique(T, return_index=True)
8 new_T = np.arange(0, len(s), 1) # New points for interpolation (1:1:D in MATLAB
9 )
10 interpolator = interp1d(T, f_acf[index], kind='cubic', fill_value='extrapolate')
11 f_acf_i = interpolator(new_T)
12
13 # Classify the speech signal
14 f_acf_pos = f_acf[f_acf > 0]
15 f_acf_male = f_acf_pos[(f_acf_pos >= 70) & (f_acf_pos <= 160)]
16 f_acf_female = f_acf_pos[(f_acf_pos > 160) & (f_acf_pos <= 275)]
17 f_acf_child = f_acf_pos[(f_acf_pos > 275) & (f_acf_pos <= 500)]
18 result = 'Adult Male'
19 max_len = len(f_acf_male)
20 if max_len < len(f_acf_female):
21     result = 'Adult Female'
22     max_len = len(f_acf_female)
23 if max_len < len(f_acf_child):
24     result = 'Child'
25 print(f'The voice is ', result)
```



## Solution Exercise 4: Pitch Tracker

```
1 # Plot pitch track
2 plt.figure(figsize=(10, 4))
3 plt.subplot(2, 1, 1)
4 t = np.linspace(0, len(s)/sr, len(s))
5 plt.plot(t, f_acf_i, label='Estimated Pitch', color='b')
6 plt.xlabel('Time (s)')
7 plt.ylabel('Pitch (Hz)')
8 plt.title('Pitch Estimation using Autocorrelation')
9 plt.legend()
10 plt.ylim(np.min(f_acf_i), np.max(f_acf_i)+100)
11 ax = plt.gca()
12 # Customize the y-axis tick locator
13 # ax.yaxis.set_major_locator(MaxNLocator(integer=True, prune='both', steps=[1, 2,
14 #                                     5, 10]))
15 ax.yaxis.set_major_locator(MultipleLocator(50))
16 plt.grid(True)
17 plt.subplot(2, 1, 2)
18 plt.plot(t, s, label='Speech Signal', color='r')
19 plt.xlabel('Time (s)')
20 plt.ylabel('Amplitude')
21 plt.legend()
22 plt.grid(True)
23 plt.show()
```