Homework 10

Solutions

1. (Hann window)

(a) By definition of the DFT

$$\hat{y}[k] = \sum_{j=1}^{N} y\,[j] \exp\left(-\frac{i2\pi kj}{N}\right) \tag{1}$$

$$= \sum_{j=1}^{N} x[j] \exp\left(\frac{i2\pi mj}{N}\right) \exp\left(-\frac{i2\pi kj}{N}\right) \tag{2}$$

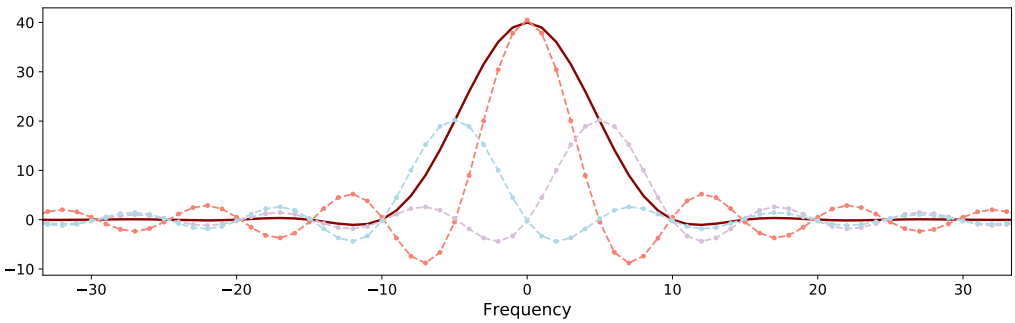
$$= \sum_{j=1}^{N} x[j] \exp\left(-\frac{i2\pi (k-m)j}{N}\right). \tag{3}$$

(b) For all indices j

$$h\,[j] = \frac{1}{2}\left(1 + \cos\left(\frac{\pi j}{w}\right)\right) \vec{\pi}[j] \tag{4}$$

$$= \frac{1}{2}\left(1 + \frac{1}{2}\exp\left(\frac{i2\pi Nj}{2wN}\right) + \frac{1}{2}\exp\left(-\frac{i2\pi Nj}{2wN}\right)\right) \vec{\pi}[j]. \tag{5}$$

(c) In the frequency domain we can interpret it as a sinc whose side lobes are cancelled out by two smaller shifted sincs. As a result, the window produces less distortion when multiplied with a signal as discussed in the lecture. See the diagram below:



2. (STFT inverse)

(a) In matrix form we have that the diagonal matrix containing the window is just an identity matrix, $\text{diag}\left(w_{[\ell]}\right) = I_{[\ell]}$. Let us separate the input vector x into segments of length $\ell/2$. We

have

$$\mathrm{STFT}_{[\ell]}(x) := \begin{bmatrix} F_{[\ell]} & 0 & 0 & 0 & 0 & \cdots \\ & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & F_{[\ell]} & 0 & 0 & \cdots \\ 0 & 0 & & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & F_{[\ell]} & \cdots \\ 0 & 0 & 0 & 0 & & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} I_{[\ell]} & 0 & 0 & \cdots \\ & 0 & 0 & \cdots \\ 0 & I_{[\ell]} & 0 & \cdots \\ 0 & & 0 & \cdots \\ 0 & 0 & I_{[\ell]} & \cdots \\ 0 & 0 & & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} x_{:\ell/2} \\ x_{\ell/2:\ell} \\ x_{\ell:3\ell/2} \\ x_{3\ell/2:2\ell} \\ x_{2\ell:5\ell/2} \\ x_{5\ell/2:3\ell} \\ \cdots \end{bmatrix}. \quad (6)$$

We can invert the first block matrix containing the DFT matrices by applying a block-diagonal matrix B containing inverse DFT matrices $\frac{1}{\ell}F^*_{[\ell]}$. This yields

$$B\,\mathrm{STFT}_{[\ell]}(x) = \begin{bmatrix} I_{[\ell]} & 0 & 0 & \cdots \\ & 0 & 0 & \cdots \\ 0 & I_{[\ell]} & 0 & \cdots \\ 0 & & 0 & \cdots \\ 0 & 0 & I_{[\ell]} & \cdots \\ 0 & 0 & & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} x_{:\ell/2} \\ x_{\ell/2:\ell} \\ x_{\ell:3\ell/2} \\ x_{3\ell/2:2\ell} \\ x_{2\ell:5\ell/2} \\ x_{5\ell/2:3\ell} \\ \cdots \end{bmatrix} \quad (7)$$

$$= \begin{bmatrix} x_{:\ell/2} \\ x_{\ell/2:\ell} \\ x_{\ell/2:\ell} \\ x_{\ell:3\ell/2} \\ x_{\ell:3\ell/2} \\ x_{3\ell/2:2\ell} \\ x_{3\ell/2:2\ell} \\ \cdots \end{bmatrix}. \quad (8)$$

Subsampling the resulting vector by selecting the odd-numbered subvectors of length $\ell/2$ recovers x exactly.

(b) Multiplication by a rectangular window causes artifacts due to the side lobes of the DFT of the window as explained in the lecture.

3. (Haar wavelet)

(a) The vectors

$$\{\varphi_{2^k,p} : 0 \le p \le 2^{n-k} - 1\}$$

form an orthonormal basis for V_k. It has dimension 2^{n-k}.

(b) The vectors

$$\{\mu_{2^{k+1},p} : 0 \le p \le 2^{n-k-1} - 1\}$$

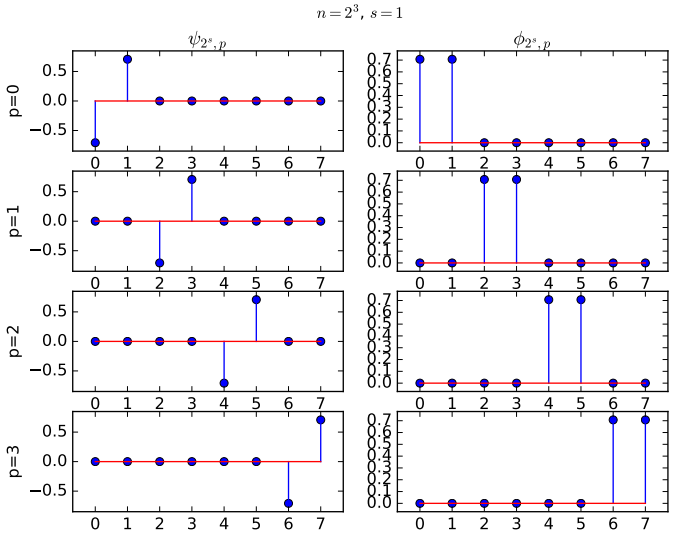
form an orthonormal basis for W_{k+1}. It has dimension 2^{n-k-1}.

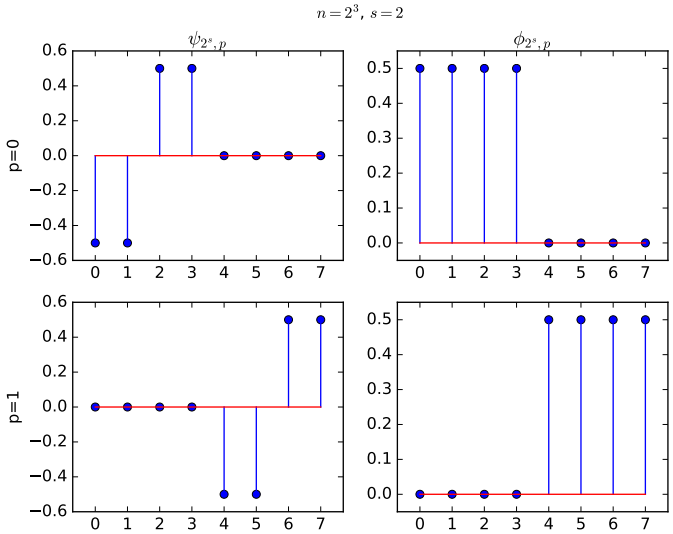
(c) The vectors

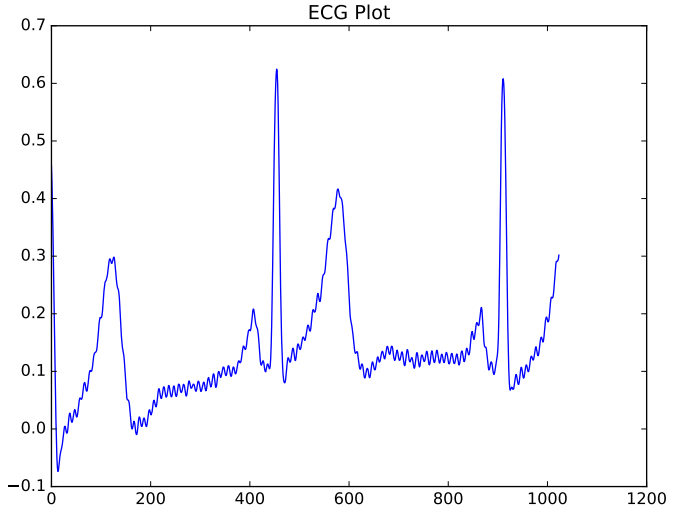
$$\{\mu_{2^j,p} : 1 \le j \le k, 0 \le p \le 2^{n-j} - 1\}$$

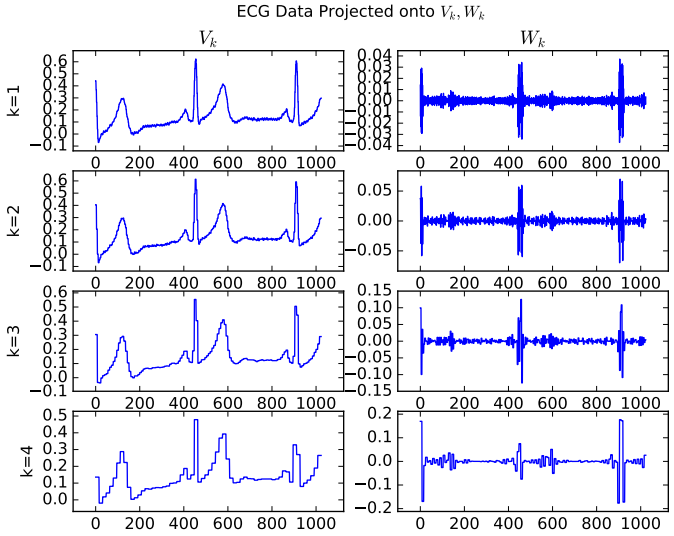
form an orthonormal basis for $W_{\le k}$. It has dimension $2^n - 2^{n-k}$.

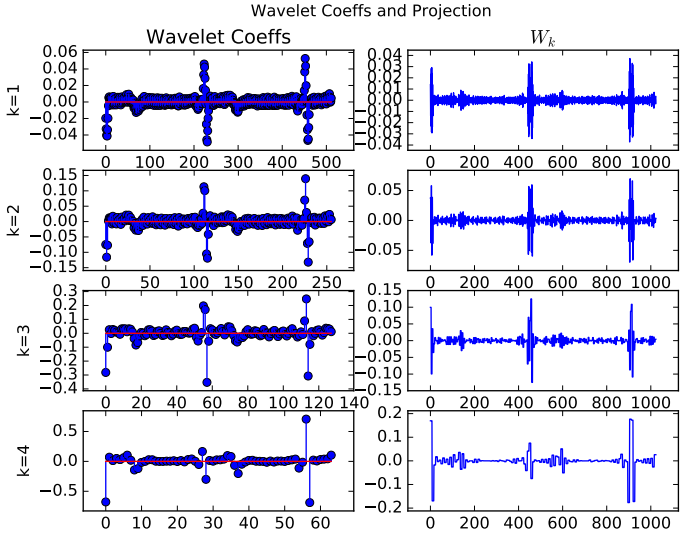
(h)


$n = 2^3,\ s = 1$
$\psi_{2^s,p}$
$\phi_{2^s,p}$
p=0
p=1
p=2
p=3


$n = 2^3,\ s = 2$
$\psi_{2^s,p}$
$\phi_{2^s,p}$
p=0
p=1


ECG Plot





4. Denoising via STFT

audio_denoising

April 24, 2020

In [1]:
```
import os
import glob

import numpy as np

import matplotlib.pyplot as plt
import IPython.display as ipd
import IPython
import matplotlib.colors as colors

from scipy import signal
from copy import deepcopy

import utils
```

In [2]:
```
dataurl = 'http://www.openslr.org/resources/1/waves_yesno.tar.gz'
```

In [3]:
```
utils.download_and_extract_data(dataurl)
```

In [4]:
```
path_to_sound_data = './waves_yesno/'
```

In [5]:
```
### STFT hyperparameters

nperseg = 512
window_size = 5
```

In [6]:
```
noise_std_array = [1e-1];  ## all noise levels to consider. feel free to play around
```

In [7]:
```
### for plotting

ini = 34500-1
end = 37500+1

ini2 = 34875
end2 = 35025

# ini2 = 33975-20
# end2 = 34025+20
```

0.0.1 Creating Dataset

```
In [8]: train_dataset, train_fs_array, train_max_array, val_dataset, val_fs_array, val_max_arra
```

train_dataset and val_dataset contains time domain signals normalized so that maximum amplitude is 1

```
In [9]: max( x.max() for x in train_dataset )
```

```
Out[9]: 1.0
```

The actual maximum amplitude of signal is saved in train_max_array and val_max_array. We will use this maximum value to un-normalize the signals before plotting or listening to it.

actual_signal[k] = train_dataset[k] * train_max_array[k]

The sampling rate of each sigal is given in train_fs_array and val_fs_array. In our case, all of them have same sampling rate. We will use sampling rate for stft function

0.0.2 Get Noise Function

```
In [10]: def get_noise(data, noise_std = 0.1):
             noise = np.random.randn(*data.shape);
             noise = noise * noise_std;
             return noise
```

1 STFT Hard and Block Thresholding

```
In [11]: def plot_stft(stft_array, save_str=''):

             """ utility function to plot stft """
             f, t, Zxx = stft_array;
             Zxx_abs = np.abs(Zxx)
             Zxx_abs[Zxx_abs<1e-5] = 1e-5
             plt.figure()
             plt.pcolormesh(t, f,Zxx_abs, norm=colors.LogNorm(vmin=Zxx_abs.min(), vmax=Zxx_abs
             tick_size = 18
             label_size = 18
             cb = plt.colorbar()
             plt.tick_params(labelsize=tick_size)
             cb.ax.tick_params(labelsize=tick_size)
             plt.ylabel('Frequency (Hz)',fontsize=label_size)
             plt.xlabel('Time (s)',fontsize=label_size)
             plt.savefig('plots/stft_denoising_'+save_str+'_stft.pdf',bbox_inches="tight")
             plt.show()
```

```
In [12]: def get_block_L2_norm(mat, window_size):

             """ mat: an nxn matrix
                 window_size: postivie integer (assume odd)
                 return: nxn matrix where the (i,j)th entry is the L2 norm of a window_size x
```

```
            neighbourhood centered at (i, j)

        to obtain an nxn output, assume that edges are zero padded.

        hint: implement using a convolution

        sample output for get_block_L2_norm(np.ones([5, 5]), 3):

      [[2.        , 2.44948974, 2.44948974, 2.44948974, 2.        ],
       [2.44948974, 3.        , 3.        , 3.        , 2.44948974],
       [2.44948974, 3.        , 3.        , 3.        , 2.44948974],
       [2.44948974, 3.        , 3.        , 3.        , 2.44948974],
       [2.        , 2.44948974, 2.44948974, 2.44948974, 2.        ]]
    """


    kernel = np.ones([window_size, window_size]);

    out = signal.convolve2d(mat**2, kernel, boundary='fill', mode='same')

    return np.sqrt(out)
```

1.0.1 Implement Hard and Block Thresholding

```
In [13]: def stft_denoising(source, noise_std, fs, nperseg, thresh,
                            window_size, block_thresh = None, plot_res = True, ind=0):

    """
    implements hard and block thresholding.

    thresh - threshold for hard thresholding. Thresholding is implemented per coeffic
    block_thresh - threshold for block thresholding. Thresholding is implemented base

    """

    noisy = source + get_noise(source, noise_std);

    if block_thresh is None:
        block_thresh = thresh;

    source_stft = signal.stft(source, fs = fs, nperseg=nperseg);
    noisy_stft = signal.stft(noisy, fs = fs, nperseg=nperseg);

    ## remove this for question
    denoised_stft = deepcopy(noisy_stft);
    block_denoised_stft = deepcopy(noisy_stft);

    Zxx = noisy_stft[2];
```

```
    abs_Zxx_block_L2_norm = get_block_L2_norm(np.abs(Zxx), window_size);

    denoised_stft = list(denoised_stft)
    denoised_stft[2] = np.where(np.abs(Zxx) >= thresh, Zxx, 0);

    block_denoised_stft = list(block_denoised_stft)
    block_denoised_stft[2] = np.where(abs_Zxx_block_L2_norm >= block_thresh, Zxx, 0);

    ## remove this for question

    if plot_res:
        plot_stft(source_stft,save_str='_clean_'+str(ind));
        plot_stft(noisy_stft,save_str='_noisy_'+str(ind));
        plot_stft(denoised_stft,save_str='_denoised_'+str(ind));
        plot_stft(block_denoised_stft,save_str='_block_denoised_'+str(ind));

    _, source_istft =  signal.istft(source_stft[2], fs=fs, nperseg=nperseg);
    _, noisy_istft =   signal.istft(noisy_stft[2], fs=fs, nperseg=nperseg);
    _, denoised_istft =   signal.istft(denoised_stft[2], fs=fs, nperseg=nperseg);
    _, block_denoised_istft =   signal.istft(block_denoised_stft[2], fs=fs, nperseg=np


    return np.real(source_istft), np.real(noisy_istft), np.real(denoised_istft), np.re
```

1.0.2 Choosing Threshold Based on Train Error

We will compute the error for different values of threshold and pick the threshold based on the lowest error on the training set.

We don't really do any training here. The denoising method only has one hyperparameter - the threshold. We're picking it based on the training set.

The next cells should run off hand if you have filled get_block_L2_norm() and stft_denoising()

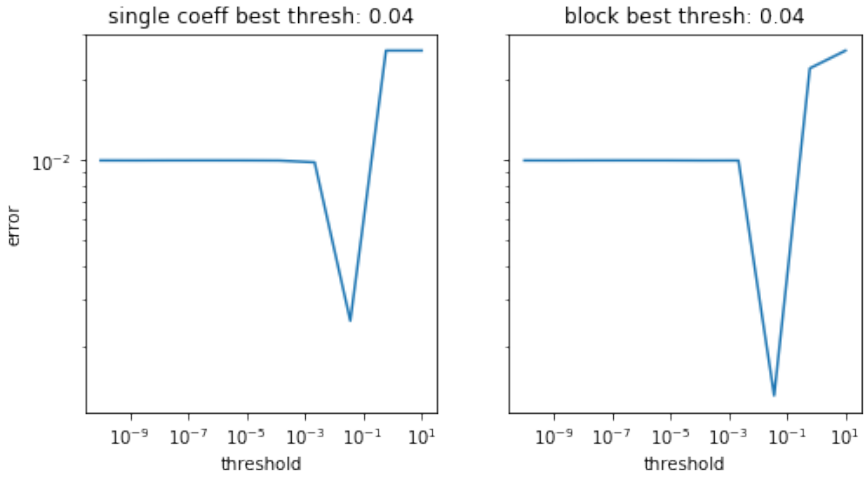
```
In [14]: np.sqrt(1e-2)
```

```
Out[14]: 0.1
```

```
In [15]: # threshold_array = [1e-10, 1e-8, 1e-6, 1e-4, 1e-2, 1e0, 1e2]

         threshold_array = np.logspace(-10, 1, num = 10)
```

```
In [16]: error_dict = {}
         block_error_dict = {}
         best_threshold_dict = {}
         block_best_threshold_dict = {}
         for noise_std in noise_std_array:

             error_dict[noise_std] = np.zeros_like(threshold_array);
```

```
        block_error_dict[noise_std] = np.zeros_like(threshold_array);

        for thresh_i, thresh in enumerate(threshold_array):
            total_error = 0.0;
            block_total_error = 0.0;
            total_length = 0.0;
            for i, x in enumerate(train_dataset):
                rec_source, rec_noisy, rec_denoised, rec_denoised_block = stft_denoising(x
                total_error += np.linalg.norm( rec_denoised - rec_source)**2;
                block_total_error += np.linalg.norm( rec_denoised_block - rec_source)**2;
                total_length += len(rec_source);

            error_dict[noise_std][thresh_i] = (total_error/total_length);
            block_error_dict[noise_std][thresh_i] = (block_total_error/total_length);

        print('noise std: ', noise_std);
        fig, axes = plt.subplots(1, 2, sharex=True, sharey=True, figsize = (8, 4));

        axes[0].loglog(threshold_array, error_dict[noise_std])
        axes[0].set_xlabel('threshold')
        axes[0].set_ylabel('error')

        best_threshold_dict[noise_std] = threshold_array[np.argmin(error_dict[noise_std])]

        axes[0].set_title('single coeff best thresh: '+str( round(best_threshold_dict[nois

        axes[1].loglog(threshold_array, block_error_dict[noise_std])
        axes[1].set_xlabel('threshold')
#       axes[1].set_ylabel('error')

        block_best_threshold_dict[noise_std] = threshold_array[np.argmin(block_error_dict

        axes[1].set_title('block best thresh: '+str( round(block_best_threshold_dict[noise

        plt.show()

        print('='*50+'\n')


noise std:  0.1
```



==========

1.0.3 Comparison Between STFT Hard and Block Threshold

In [17]:
```
val_idx = 2

source = val_dataset[val_idx]
fs = val_fs_array[val_idx]
max_val = val_max_array[val_idx]
```

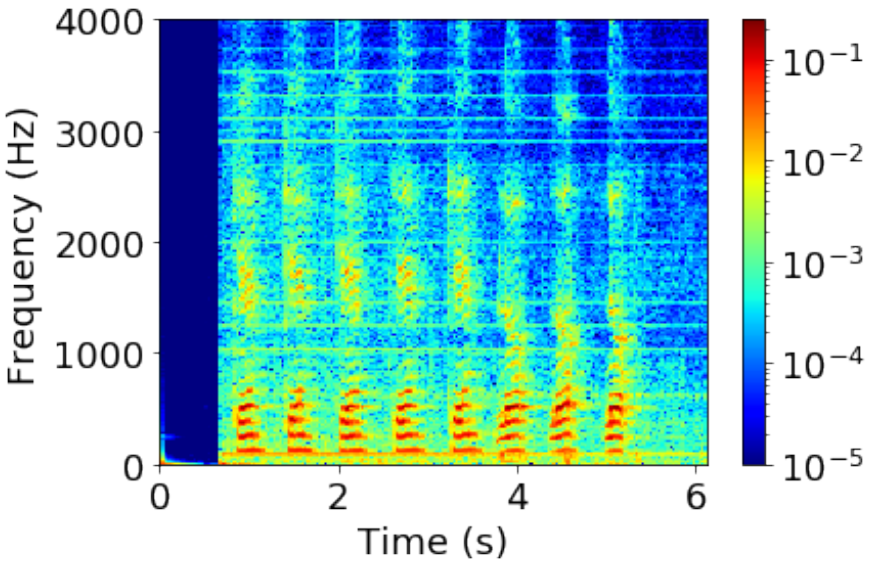
In [18]:
```
fig_size=(20,6)
```

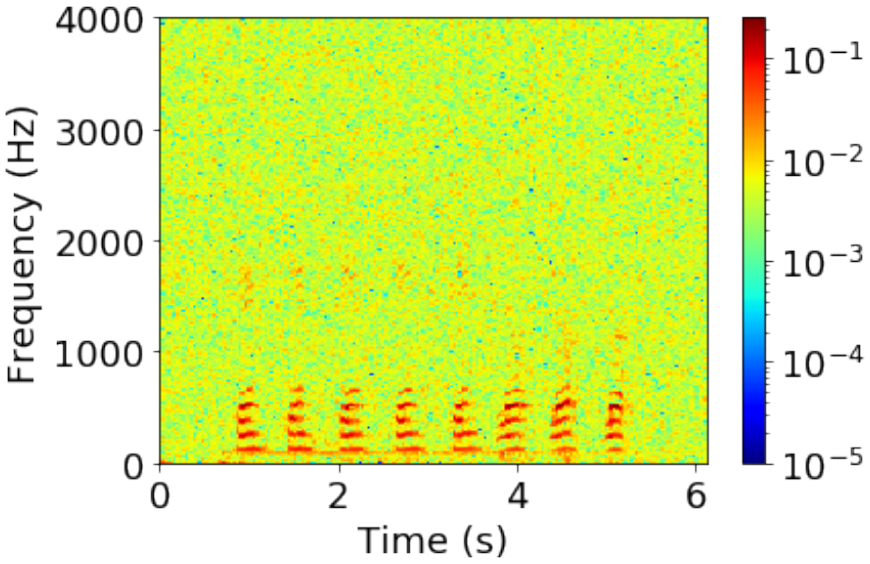
In [19]:
```
for ind_fig, noise_std in enumerate(noise_std_array):

    print('Noise Std: ', noise_std)

    noise_sample = get_noise(source, noise_std = noise_std);


    istft_source, istft_noisy, istft_denoised, istft_denoised_block = stft_denoising(s
                                                                      thresh = best_threshold
                                                                      window_size = window_s
                                                                      block_thresh = block_b

    istft_denoised_block *= max_val
    istft_denoised *= max_val;
```

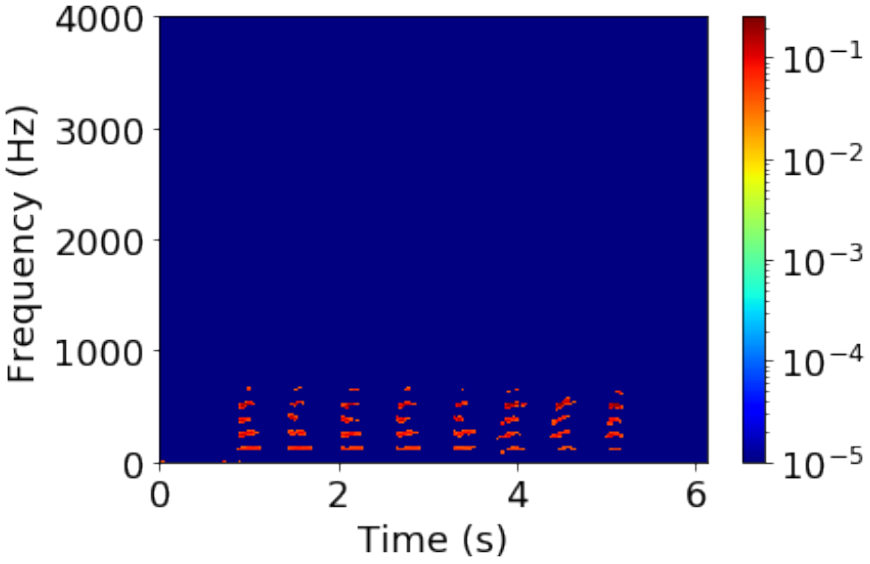
```
istft_noisy *= max_val;
istft_source *= max_val;

print('STFT Denoised: ')
IPython.display.display(ipd.Audio(istft_denoised, rate=fs))

print('Block STFT Denoised: ')
IPython.display.display(ipd.Audio(istft_denoised_block, rate=fs))

label_size = 18
font_size = 18

t_indices = np.arange(len(istft_source))/fs

plt.figure(figsize = fig_size)
plt.plot(np.real(istft_source))
plt.xlabel('Time (s)',fontsize=font_size)
plt.tick_params(labelsize=label_size)

plt.figure(figsize = fig_size)
plt.plot(t_indices[ini:end],np.real(istft_denoised[ini:end]))
plt.xlabel('Time (s)',fontsize=font_size)
plt.tick_params(labelsize=label_size)
plt.savefig('plots/stft_stft_denoised_' + str(ind_fig) + '.pdf',bbox_inches="tight

plt.figure(figsize = fig_size)
plt.plot(t_indices[ini:end],np.real(istft_denoised_block[ini:end]))
plt.xlabel('Time (s)',fontsize=font_size)
plt.tick_params(labelsize=label_size)
plt.savefig('plots/stft_block_denoised_' + str(ind_fig) + '.pdf',bbox_inches="tigh



plt.figure(figsize = fig_size)
plt.plot(t_indices[ini2:end2],np.real(istft_source[ini2:end2]),'--o',markersize=6
plt.plot(t_indices[ini2:end2],np.real(istft_denoised[ini2:end2]),'x',color='tomat
plt.plot(t_indices[ini2:end2],np.real(istft_noisy[ini2:end2]),'.',color='darkgree
plt.xlabel('Time (s)',fontsize=font_size)
plt.tick_params(labelsize=label_size)
plt.legend(fontsize=font_size)
plt.savefig('plots/stft_stft_denoised_' + str(ind_fig) + '_zoom.pdf',bbox_inches='

plt.figure(figsize = fig_size)
plt.plot(t_indices[ini2:end2],np.real(istft_source[ini2:end2]),'--o',markersize=6
plt.plot(t_indices[ini2:end2],np.real(istft_denoised_block[ini2:end2]),'x',color=
plt.plot(t_indices[ini2:end2],np.real(istft_noisy[ini2:end2]),'.',color='darkgree
plt.xlabel('Time (s)',fontsize=font_size)
plt.tick_params(labelsize=label_size)
```

```
            plt.legend(fontsize=font_size)
            plt.savefig('plots/stft_block_denoised_' + str(ind_fig) + '_zoom.pdf',bbox_inches=

            plt.show()


            print('='*50 + '\n')
```

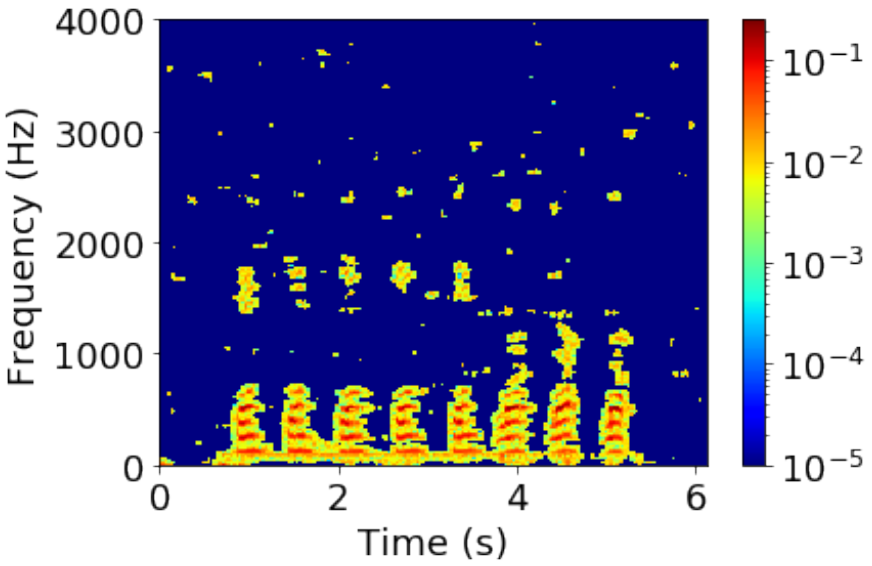
```
Noise Std:  0.1
```




4000
3000
2000
1000
0
Frequency (Hz)
0
2
4
6
Time (s)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}


4000
3000
2000
1000
0
Frequency (Hz)
0
2
4
6
Time (s)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}

9

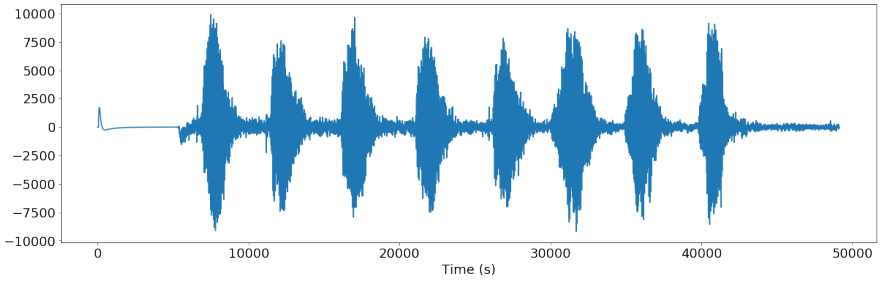


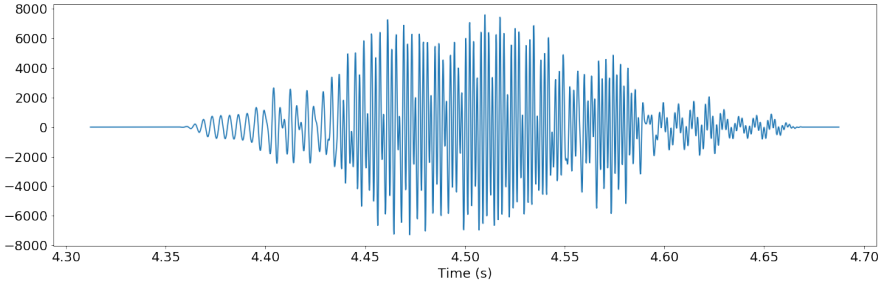

```
STFT Denoised:
```

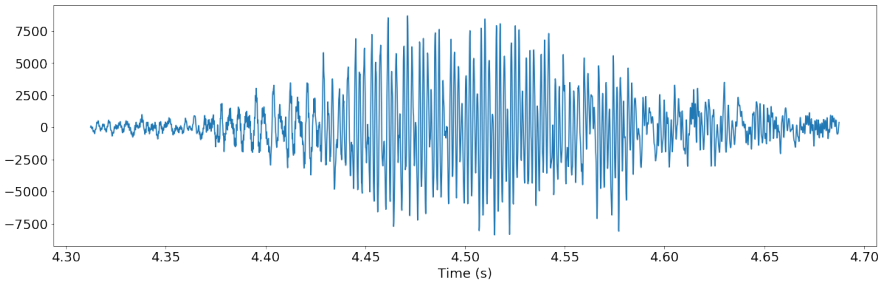
```
<IPython.lib.display.Audio object>
```

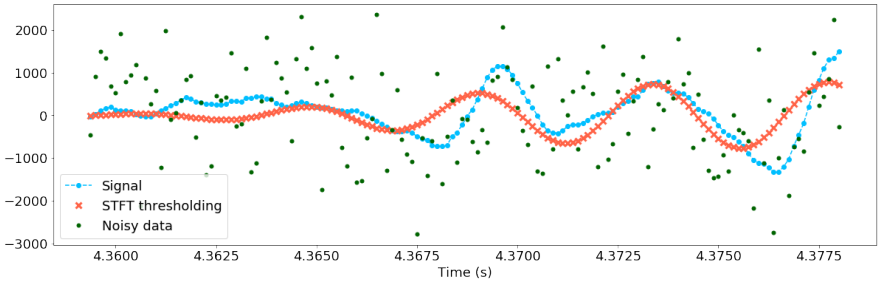
```
Block STFT Denoised:
```

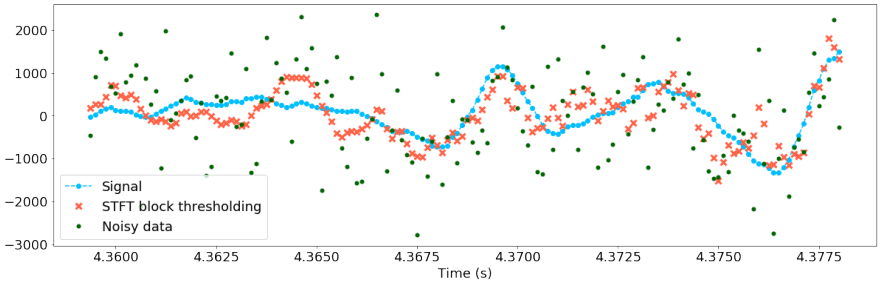
```
<IPython.lib.display.Audio object>
```




8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
4.30
4.35
4.40
4.45
4.50
4.55
4.60
4.65
4.70
Time (s)


7500
5000
2500
0
−2500
−5000
−7500
4.30
4.35
4.40
4.45
4.50
4.55
4.60
4.65
4.70
Time (s)


2000
1000
0
−1000
−2000
−3000
Signal
STFT thresholding
Noisy data
4.3600
4.3625
4.3650
4.3675
4.3700
4.3725
4.3750
4.3775
Time (s)


2000
1000
0
−1000
−2000
−3000
4.3600
4.3625
4.3650
4.3675
4.3700
4.3725
4.3750
4.3775
Time (s)
Signal
STFT block thresholding
Noisy data

==========