ECE 3640 - Discrete-Time Signals and Systems

Noise Cancellation by Spectral Subtraction

Jake Gunther

Spring 2015

Department of Electrical & Computer Engineering

background

scenario:

- a sensor is used to measure a desired signal $x(t)$
- the senor can not be placed at the signal source
- measured signal $y(t)$ is corrupted by additive white noise $n(t)$

$$y(t) = x(t) + n(t)$$

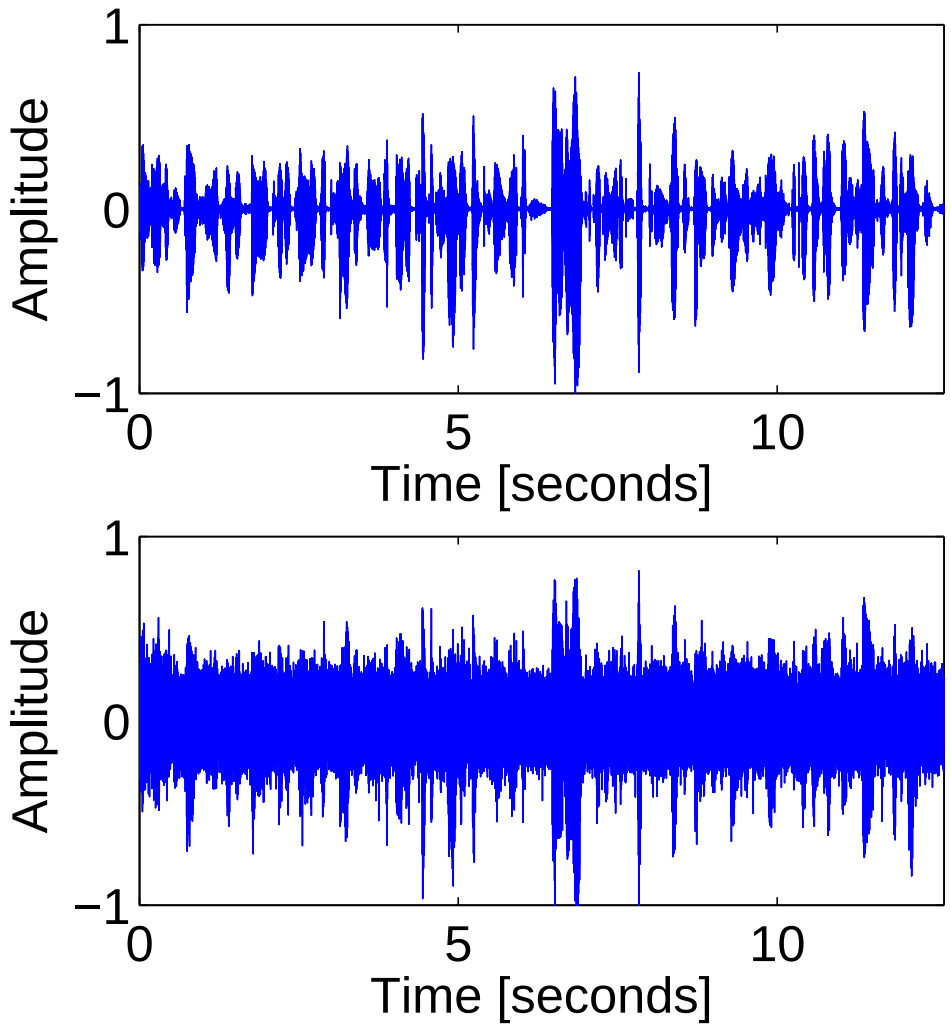
objective:

- recover $x(t)$ given the measurements $y(t)$

to get started:

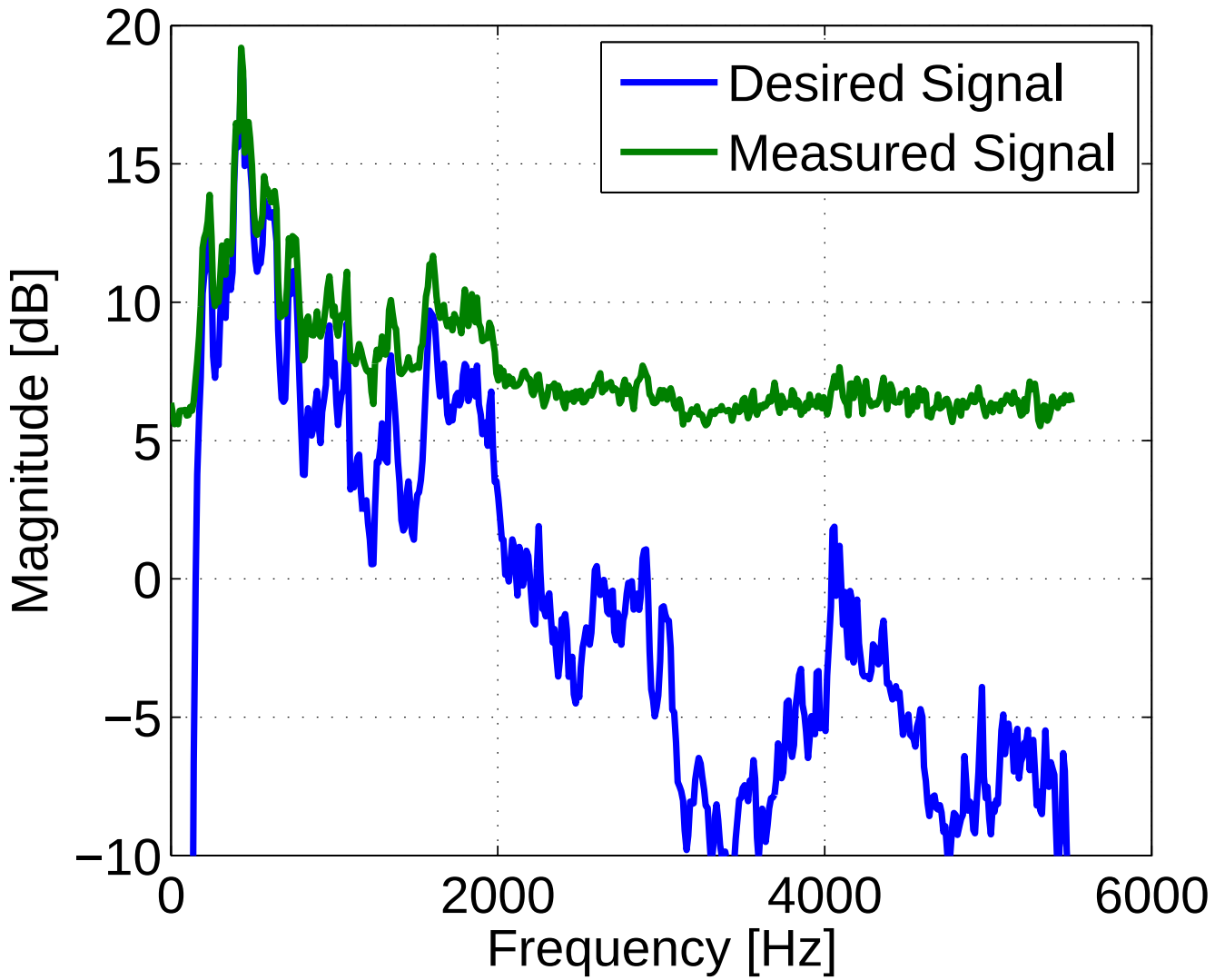
- using real data or simulated data, plot the signal in time and frequency to investigate the effects of the noise

time domain



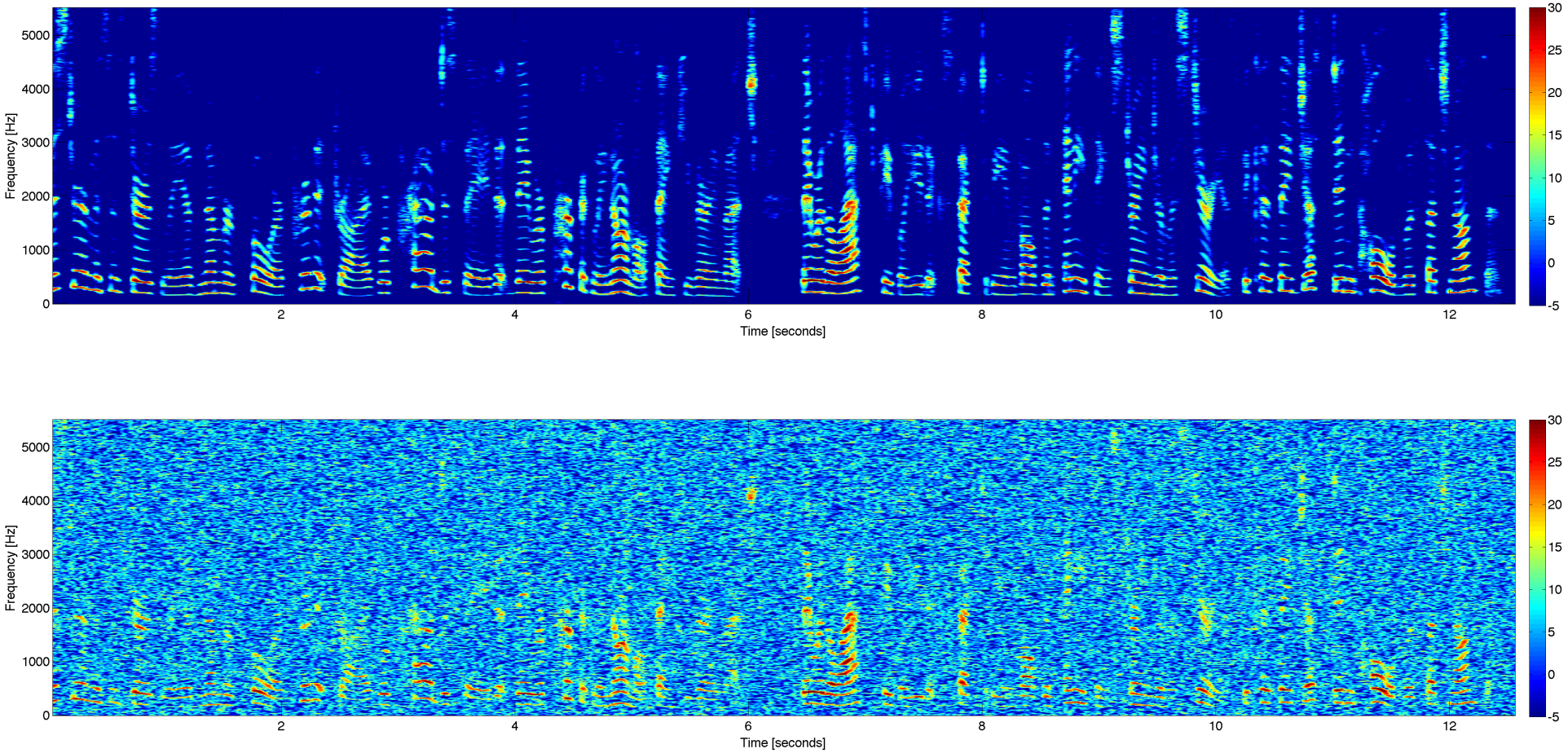
- noise fills in the signal

frequency domain



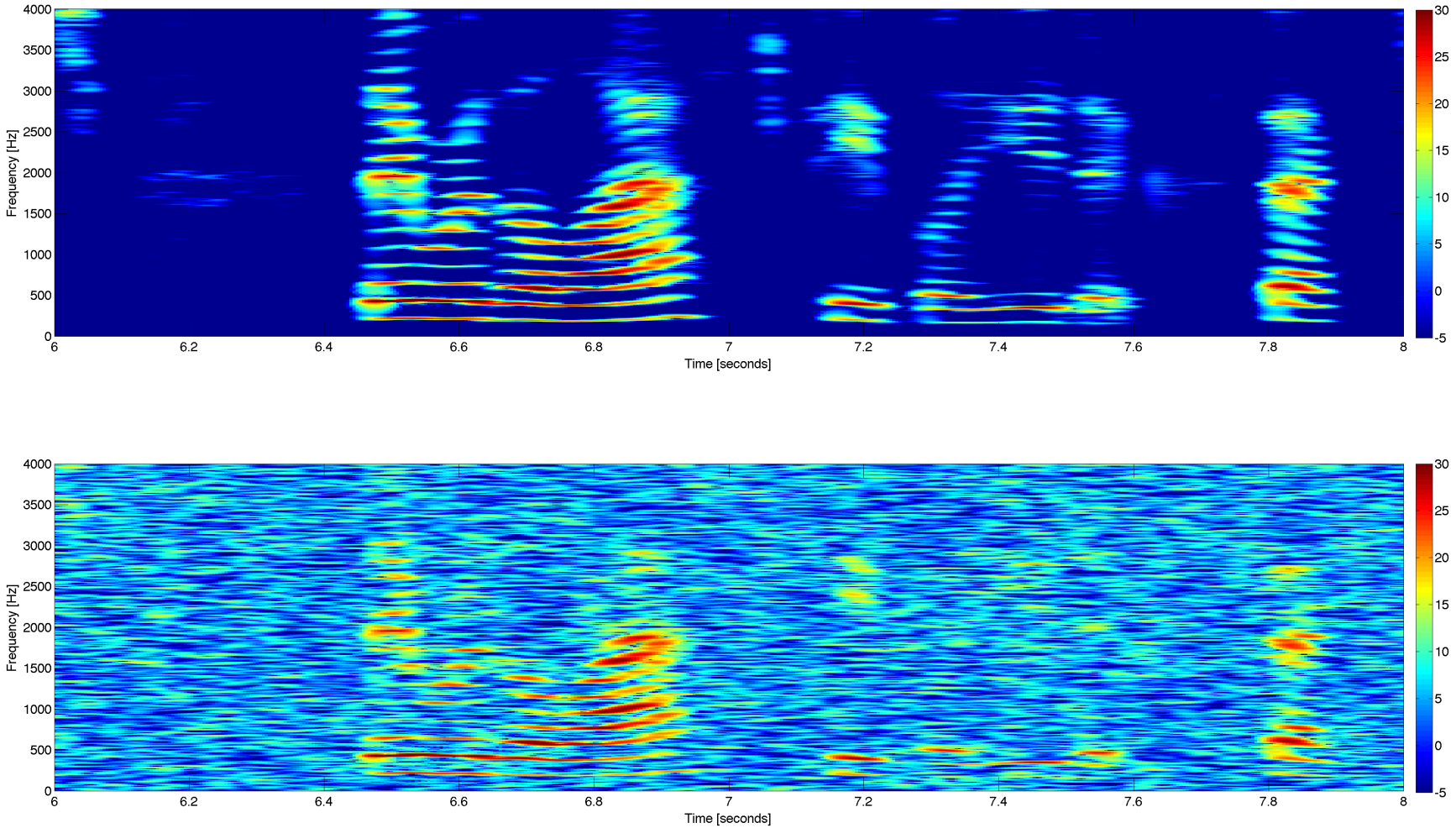
- noise fills up the signal spectrum (high frequencies are underwater)

spectrogram (T-F) domain



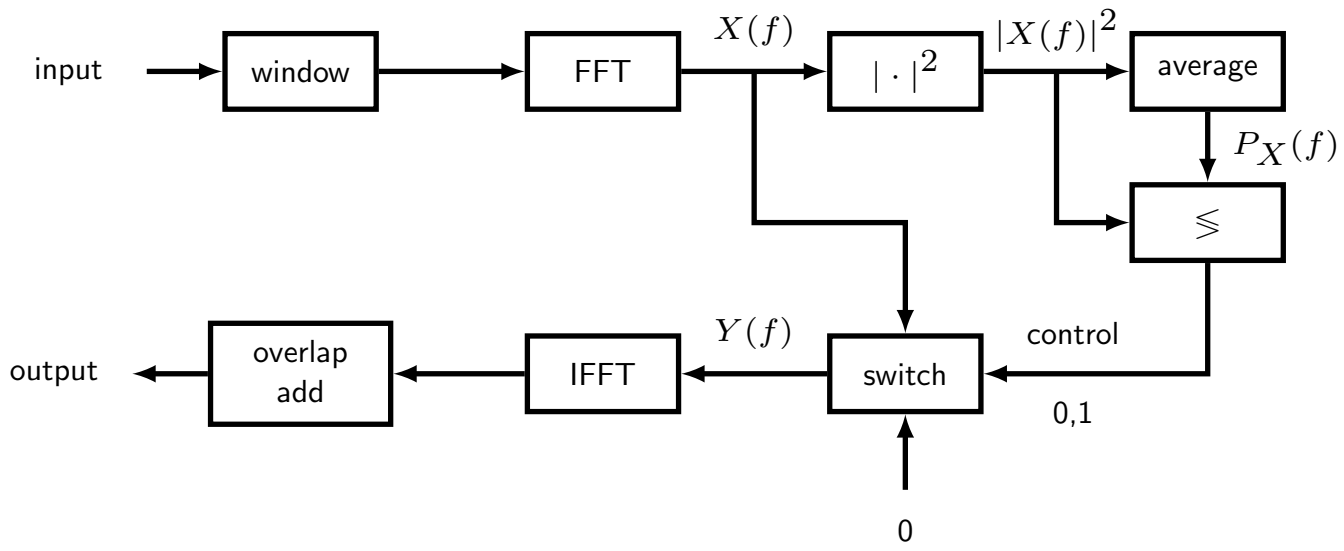
- top: desired signal
- bottom: measured signal (noisy)
- desired signal could be recovered by setting noise areas to zero

spectrogram (T-F) domain (zoomed)



- top: desired signal
- bottom: measured signal (noisy)
- desired signal could be recovered by setting noise areas to zero

algorithm: spectral subtraction



- switch controlled by comparison

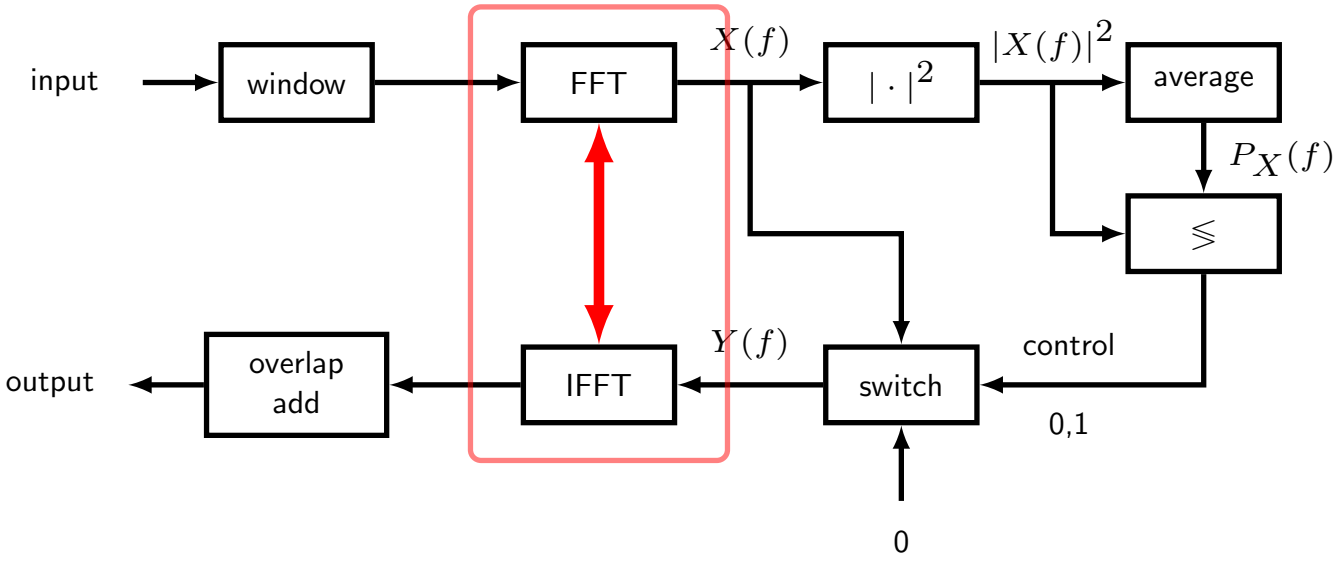
$$Y(f) = \begin{cases} X(f), & \text{if } |X(f)|^2 \geq 10 P_X(f), \\ 0, & \text{otherwise} \end{cases}$$

- use IIR-LPF for average

$$P_X(f) = \lambda P_X(f) + (1-\lambda)|X(f)|^2$$

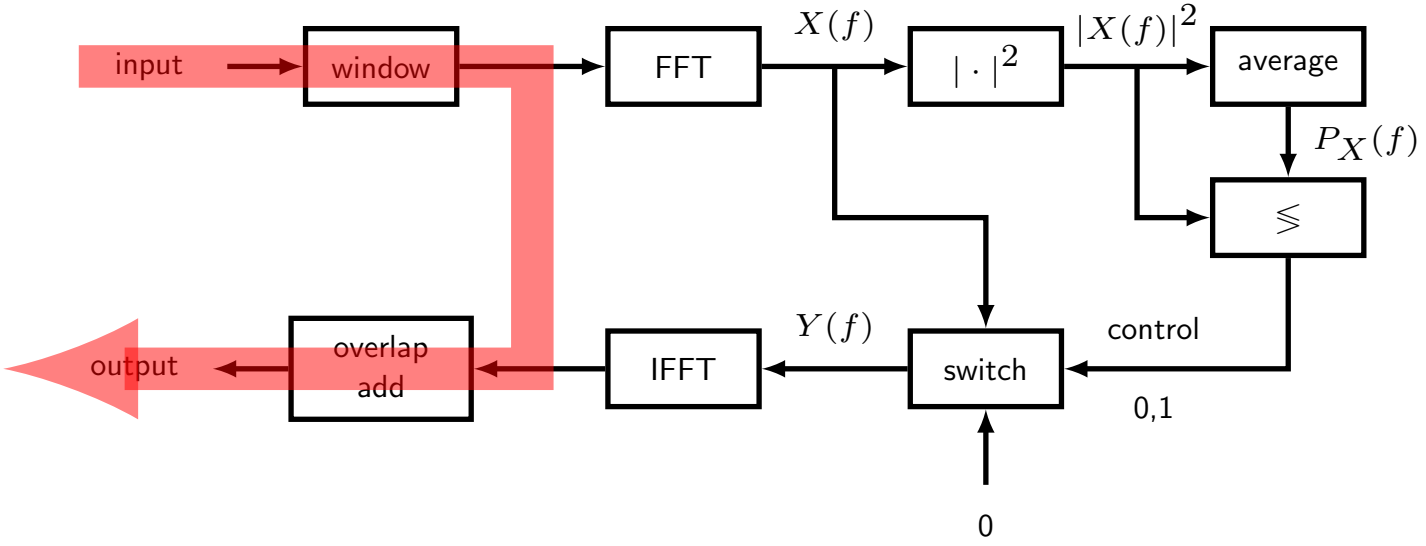
- use 256-point FFT
- use 128-point window overlap
- use rectangular window
- use FFTW3 package for FFT and IFFT
- write a C program to apply this algorithm to the file `harry8noise.wav`

coding step: write FFT/IFFT test program



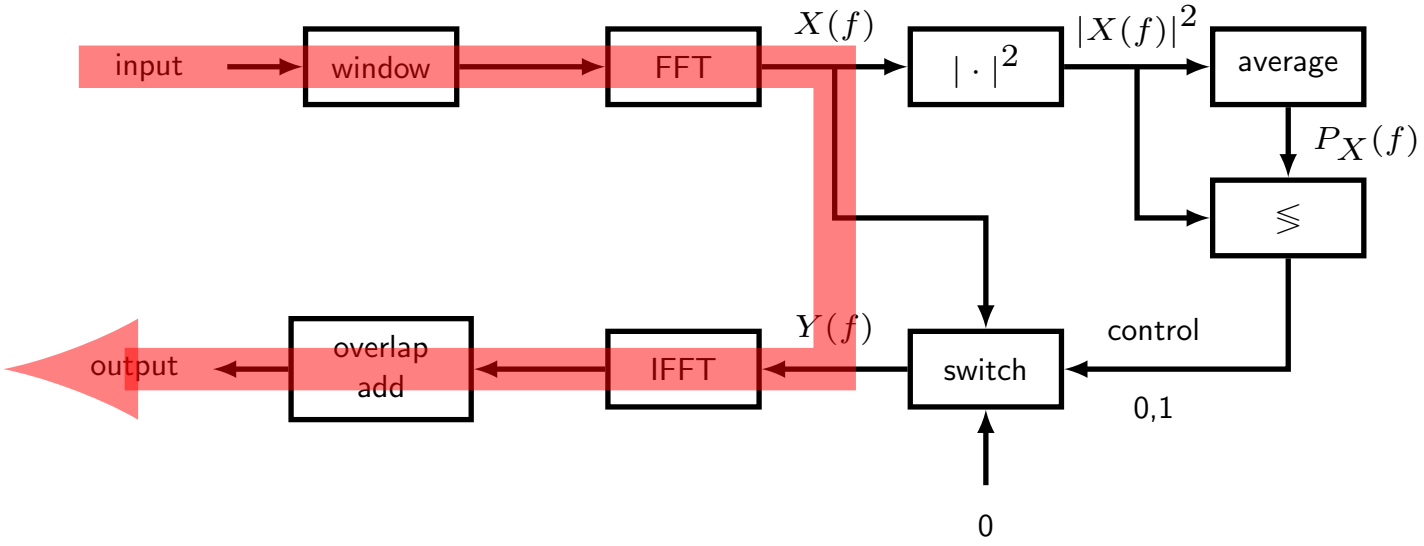
- learn how to use the FFTW3 package
- use real-to-complex for FFT
- use complex-to-real for IFFT
- use float (rather than double) FFTs
  on Unix/Linux type: `./configure --enable-float && make`
- use a test signal for which the FFT is known or a random signal and compare outputs with FFT in Matlab

coding step: buffered I/O, windowing and overlap-add



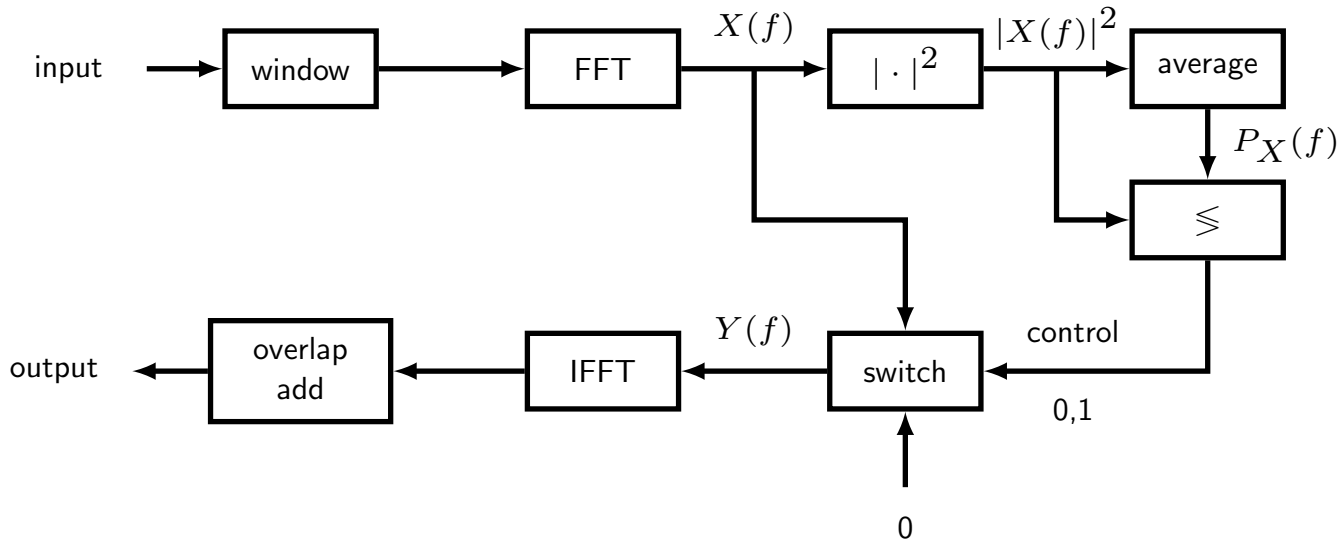
- implement buffered file input and output (read in only what is needed, write out what is produced
- this automatically accomplishes windowing
- implement overlap-add in the output buffer
- program output should be identical to the program input

coding step: add FFT and IFFT processing



- add FFT and IFFT to your program
- short-circuit the output of the FFT to the input of the IFFT
- save computed spectral magnitude to an image file and view in Matlab (this is a spectrogram)
- program output should be identical to the program input

coding step: add frequency-domain processing steps



- add and test spectral averaging
- add and test switch and control logic
- listen to the program output in Matlab

assignment

turn in the following items:

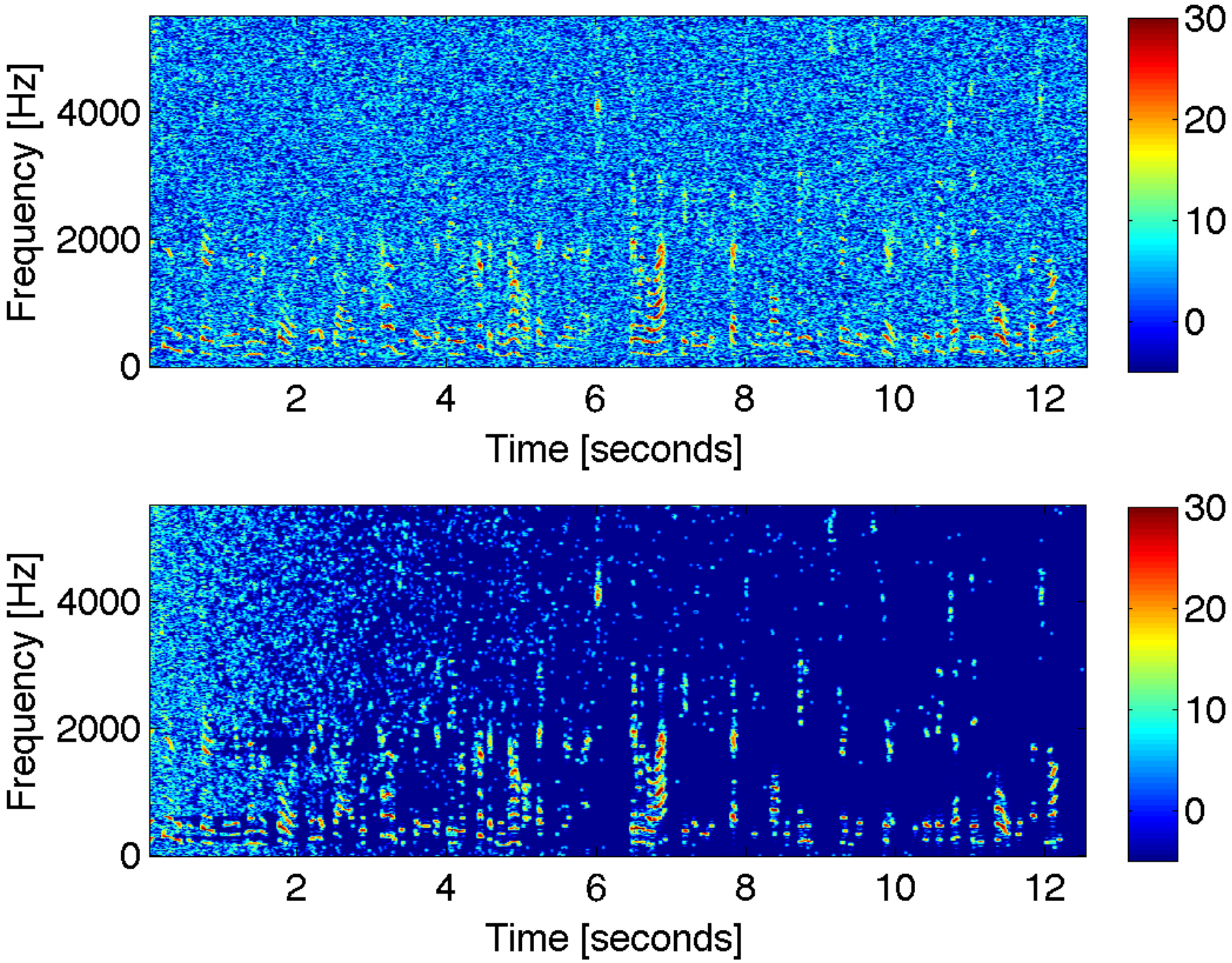
1. your C code

2. a plot showing spectrograms of the noisy signal and of the cleaned signal (for the spectrograms use a 1024-point FFT size, 95% overlap, and a hamming window)

3. a plot showing the noisy and cleaned signals in the time-domain (scale the time axis to be in units of seconds)

hint: example plots appear on the following slides

spectrograms in dB: noisy (top) and cleaned (bottom) signals



time-domain plots: noisy (top) and cleaned (bottom) signals