Lab topic 3

Pseudo-random numbers

- Homework reading: Find an online source to understand how random numbers are generated on a computer
- Example: https://en.wikipedia.org/wiki/Pseudorandom_number_generator

Generating WGN

- Read the help for the **randn** function in Matlab and generate a realization of WGN with 10,000 samples and :
  - $\mu = 0, \sigma = 1$
  - $\mu = 0, \sigma = 2$
  - $\mu = 0, \sigma^2 = 2$
  - $\mu = 2,\ \sigma^2 = 2$
- Learn to use the **histogram** function of Matlab to make a histogram of each realization
- Obtain the sample mean and standard deviations for each realization
  - Matlab: **mean** and **std**

Colored Gaussian Noise

- See **NOISE/colGaussNoiseDemo.m**
- In this script, we design an FIR filter for the target PSD

$$S_n(f) = \begin{cases} (f-100)(300-f), & f \in [100,300] \\ 0, & otherwise \end{cases}$$

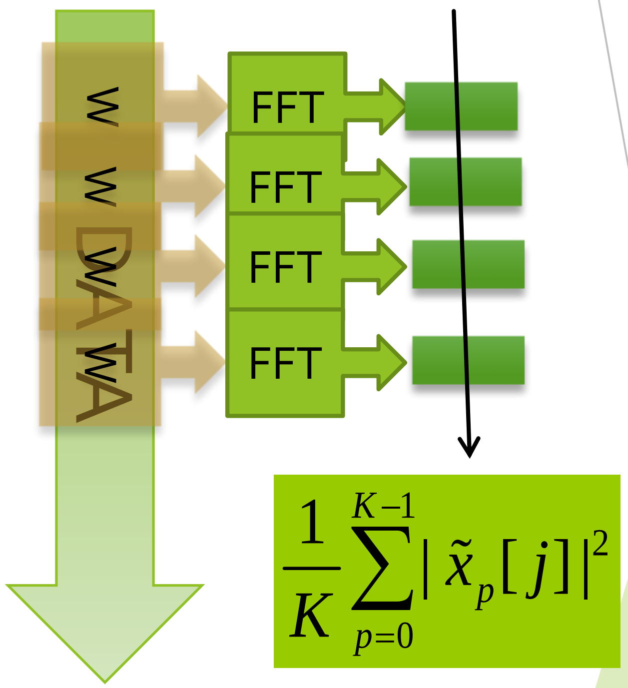
- And use the Wiener-Khinchin theorem to generate 16384 samples of colored Gaussian noise with the above PSD using a sampling frequency of 1024 Hz
- Run the script
  - Examine the noise time series by zooming in: Does it look like a WGN realization? How does it differ?
  - Plot the histogram of the noise realization: Is it still a Normal PDF?
  - Can you explain why the variance of the output noise is less than that of the input noise?

Welch's method

- Welch's method of overlapping windows
  - Already a built-in function in Matlab: `psd` (old) `pwelch` (new)
- Compute FFTs of K short overlapping windowed segments
- Calculate modulus squared of the FFT

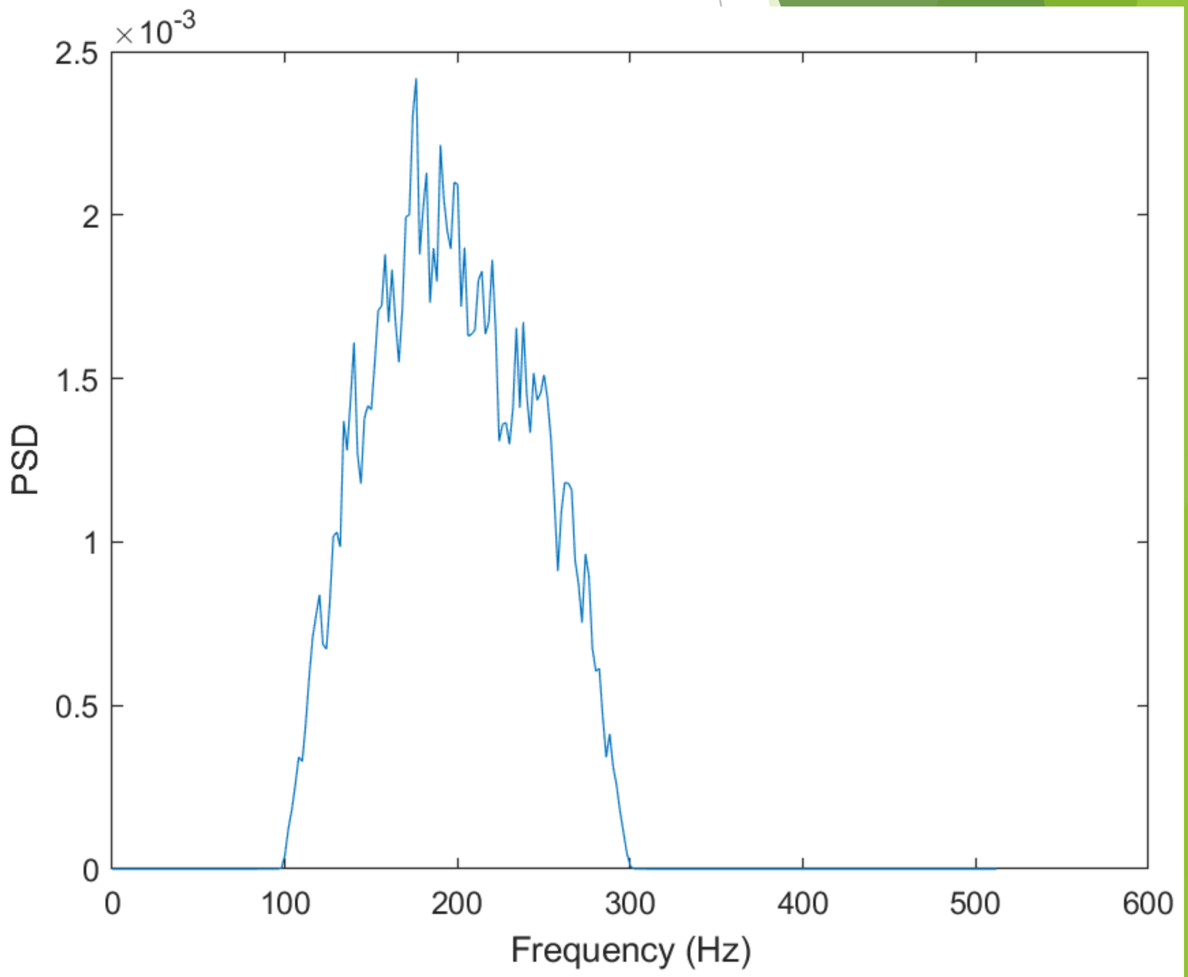
$$\text{PSD estimate: } S_n[j] = \frac{1}{K}\sum_{p=0}^{K-1} |\tilde{x}_p[j]|^2$$

- Matlab: `pwelch(x,nWin,[],[],fs)`
  - `x` : data vector
  - `nWin`: number of samples in each short segment
  - `fs` : sampling frequency of the data"



Estimating PSD

- The script **NOISE/colGaussNoiseDemo.m** generates a realization of colored Gaussian noise with a prescribed PSD
- *Estimate* its PSD using the `pwelch` function in Matlab



Arbitrary colored Gaussian noise

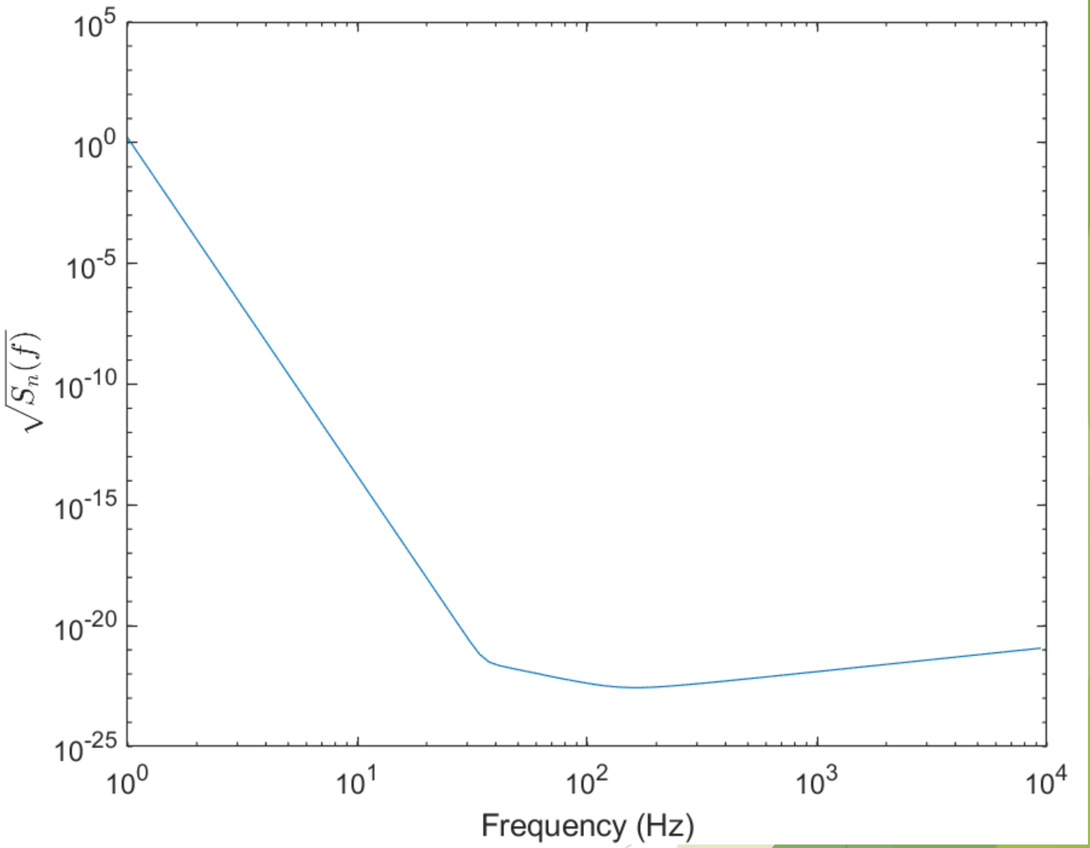
- If you have understood the main idea behind **NOISE/colGaussNoiseDemo.m,** write a function:
  - Input:
    - Number of samples
    - N-by-2 matrix containing $[f, \sqrt{S_n(f)}\ ]$ values on each row
    - FIR filter order
    - Sampling frequency (f_s)
  - Output:
    - Realization of Colored Gaussian noise with the given number of samples and sampling frequency

f (Hz)	$\sqrt{S_n(f)}$
$f_1 = 0$	s_1
f_2	s_2
$\vdots$	$\vdots$
$f_M = f_s/2$	s_M

Simulating LIGO noise

Objective

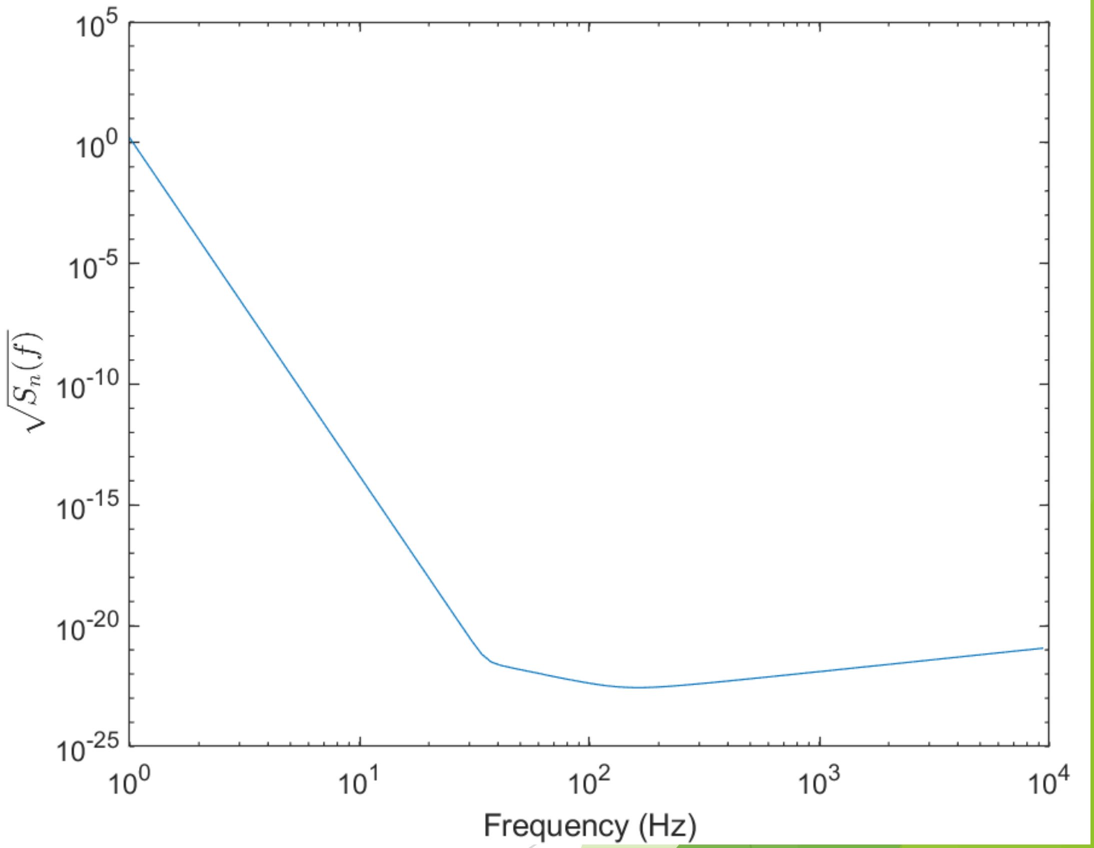
- Simulate the noise of an interferometric detector
- We will pick the sensitivity curve of the initial LIGO detector as an example for the target PSD
- The same steps can be used for any other design sensitivity curve (e.g., advanced LIGO, LISA etc)



Initial LIGO design sensitivity

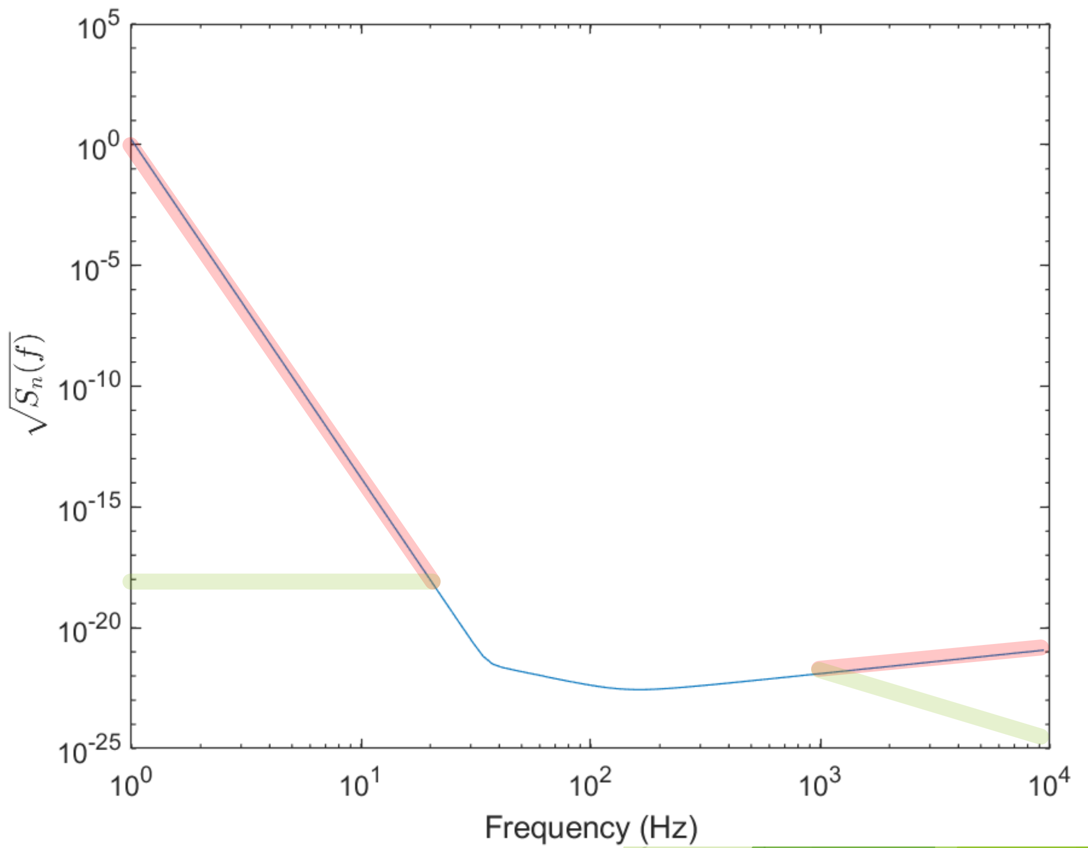
- The PSD is provided in the file **NOISE/iLIGOSensitivity.txt**
- It is a plain text file which can be read into Matlab
- First column is Frequency f (Hz) and second column is $\sqrt{S_n(f)}$

```
>> y = load('iLIGOSensitivity.txt','-ascii');
>> whos y
  Name  Size    Bytes   Class
  y     97x2    1552   double
>> loglog(y(:,1),y(:,2))
```



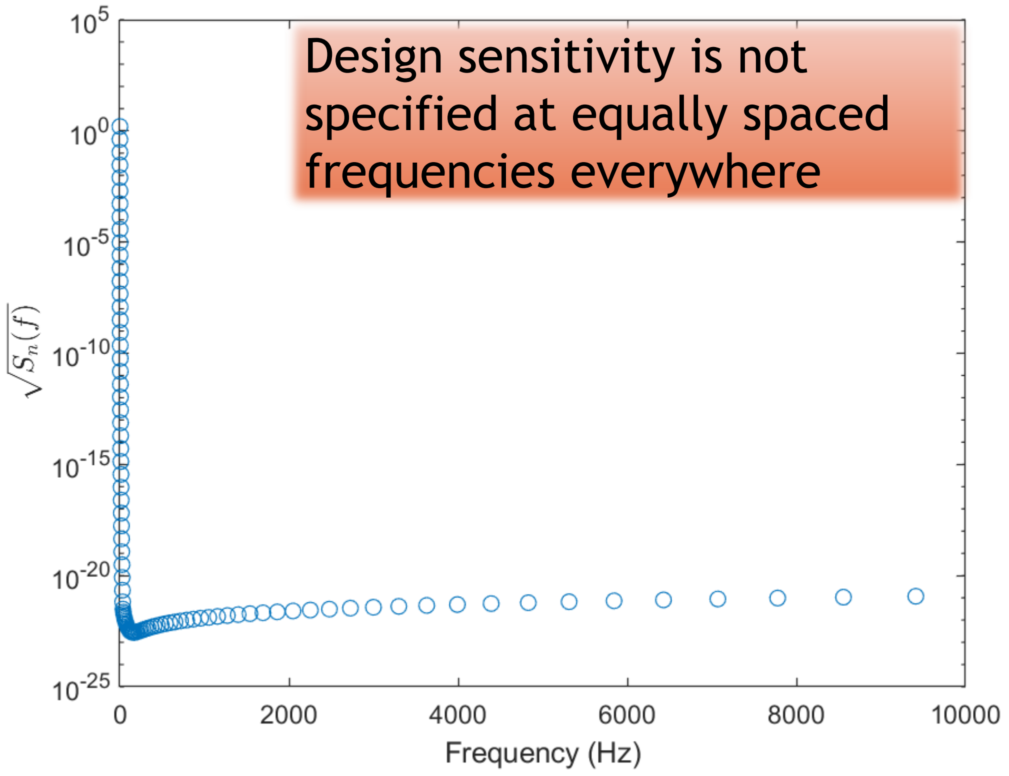
Modifications

- In any data analysis method, the low and high frequency parts will be filtered out $\Rightarrow$ the PSD of the simulated noise need not match the design PSD in those parts
- The order of an FIR filter that can reproduce the steeply rising seismic part in its transfer function will be very high $\Rightarrow$ Making the PSD goes smoothly to zero or just be a constant in these parts will help Matlab in designing better filters



Modifications

- For $f \leq 50$ Hz: $S_n(f) \rightarrow S_n(f = 50)$
- For $f \geq 700$ Hz: $S_n(f) \rightarrow S_n(f = 700)$
  - 700 Hz is where the inspiral phase of a binary of double neutron star will terminate
  - No point in keeping noise above this frequency in the data
- Remember that you need normalized frequencies of 0 and 1 for input to FIR1
  - Add $f = 0, S_n(f = 0)$ and $\frac{f_s}{2}, S_n\left(f = \frac{f_s}{2}\right)$ to the list if these are missing
- While not necessary for FIR1, interpolating the modified PSD to a regular grid of frequency values is helpful: Use the `interp1` function
- Use previously written code to produced noise realizations and estimate the PSD



Example of simulated LIGO noise PSD

- Example of an acceptable fit.
- Note that the LIGO plots are logarithmic while the plot here is on a linear scale for frequency
  - Also, different truncation and sampling frequencies here
- The “bumpiness” in the PSD near the minimum is an artifact of the approximation inherent in filter design
  - You should try to minimize such artifacts by choosing design parameters appropriately.