

# *Characterization of Noise in Gravitational Waves using Power Spectral Densities*

Lucas Romero Fernández

30<sup>th</sup> of March 2025

## 1 Introduction and Motivation

Gravitational Waves (GWs) and their detection have been one of the main focuses of the General Relativity (GR) scientific community in recent years. Notably, the observation of continuous GWs generated by strongly rotating neutron stars using Earth-based interferometric detectors such as LIGO, Advanced LIGO and VIRGO, among others, has gained particular attention worldwide. However, this task has proven to be both challenging and laborious, particularly regarding the technical aspect of instrumental sensitivity, which is fundamentally important due to the extremely weak nature of GW emissions far away from gravitational sources and gravity itself. For a more detailed analysis of these concepts, their implications and the fundamentals of their detection, see my previous work in [1]. Given these circumstances, the observation sensitivity of these detectors must be of the highest order possible and be a top priority, achieved through larger detectors and, most importantly, by isolating them from external interactions (for example, suspending them in vacuum and in extremely cold environments located deep underground). Unfortunately, complete isolation is not possible on Earth due to its seismic, thermal, quantum and other intrinsic signal fluctuations; therefore, all the collected data are contaminated by undesired noise, which must be suppressed in some form to obtain a clean GW signal. This noise and its properties, even if of random nature, can be statistically characterized using different methods. The process employing a specific procedure involving one-sided Power Spectral Densities (PSDs), discrete Fourier Transforms (DFTs), and the Welch method to compare it to a theoretical noise curve is the main objective of the present study.

## 2 Methodology

Before delving into the process in full detail, the nature and presentation of this work are a combination of theoretical concepts of signal processing and noise characterization, all complemented by numerical computing shown in the form of code snippets in the Julia Programming Language [2].

### 2.1 Initial Data Parameters

Beginning with the input data that the detector receives, the initial data used can be characterized as uninterrupted (no gaps in data values) and obtained within an observation time  $T_{obs}$ , which can be further divided for easier handling into  $M$  intervals of “coherence” time  $T_{coh} = \frac{T_{obs}}{M}$ . For simplicity, only one segment will initially be considered ( $M = 1$ ), resulting in  $T_{coh} = T_{obs}$ ; this will change further down the line for different values of  $M$  when the Welch method is implemented numerically. Furthermore, this data is initially sampled in the frequency domain from an initial frequency  $f_i$  to a final frequency  $f_f$  equal to the Nyquist frequency  $f_{Nyq}$ , the frequency to which, beyond and equivalent to its value, it is assumed that the signal does not have frequency content:  $f_f = f_{Nyq}$  of sampling frequency  $f_s = 2(f_f - f_i) = 2(f_{Nyq} - f_i)$ , which influences its discretization in the time domain with its corresponding time step size  $\Delta t = f_s^{-1}$  for each data segment/interval with identical duration and quantity of data

points.

Generally, for each data segment, the detector data output  $x(t)$  at a given time  $t$  can be expressed as follows:

$$x(t) = h(t) + n(t). \quad (1)$$

where  $h(t)$  is the GW signal and  $n(t)$  is the noise fluctuation. It is sampled in a sequence of points  $\{x_j \equiv x(t_j)\}$  at the start times of each segment with an array  $t_j = t + j\Delta t$  where  $j = 0, 1, \dots, (N - 1)$  ( $N = T_{\text{coh}}f_s$  is the number of sample points) [3]. Given the focus of this study and the method to be utilized, it is assumed that the output will consist solely of stationary, colored, and Gaussian noise with mean equal to zero or, in other words,  $x(t) = n(t)$ . The specific values and expressions of these parameters have been coded as follows:

```

1  # Definition of general constants, variables, arrays and functions
2
3  T_coh = 300 # Coherent time (in this case, equivalent to the observation time of 5 minutes) (Units: s)
4  f_Nyq = 2000 # Nyquist frequency (Units: Hz)
5  f_i = 0 # Initial frequency (Units: Hz)
6  f_f = f_Nyq # End frequency (Units: Hz)
7  f_s = 2*(f_f - f_i) # Sampling frequency (Units: Hz)
8  Deltat = 1/f_s # Time step size (Units: s)
9  N = T_coh*f_s # Number of sample points
10 t_array = collect(0:N-1)*Deltat

```

Listing 1: Parameters settings.

## 2.2 Noise Generation

From these parameters, the noise signal can begin to be properly constructed from the data sequence points  $\{x_j\}$  using the following convention of the Discrete Fourier Transform (DFT):

$$\tilde{x}_k = \Delta t \sum_{j=0}^{N-1} x_j e^{\frac{-2\pi i j k}{M}}, \quad (2)$$

and the respective inverse DFT:

$$x_j(t) = \frac{1}{\Delta t N} \sum_{k=0}^{N-1} \tilde{x}_k e^{\frac{2\pi i j k}{M}}, \quad (3)$$

where  $k = 0, \dots, (N - 1)$  is the frequency index corresponding to the positive/physical frequency  $f_k = \frac{k}{T_{\text{coh}}}$  if  $0 \leq f_k \leq \text{Int}(\frac{N}{2})$  (“Int(...)” indicating the integer part) or the negative frequencies  $f_k = \frac{k-N}{T_{\text{coh}}}$  if  $\text{Int}(\frac{N}{2}) < f_k \leq N - 1$ . Afterward, a dimensionless quantity known as the normalized power  $\rho_k$  can be constructed as:

$$\rho_k = \frac{|\tilde{x}_k|^2}{\langle |\tilde{n}_k|^2 \rangle}, \quad (4)$$

where  $\langle (\dots) \rangle$  denotes the ensemble average and, utilizing that  $T_{\text{coh}} = N\Delta t$ ,  $\langle |\tilde{n}_k|^2 \rangle \approx \frac{N\Delta t}{2} S_n(f_k) = \frac{T_{\text{coh}}}{2} S_n(f_k)$  (where  $S_n(f_k)$  corresponds to the one-sided Power Spectral Density (PSD)), which consequently yields from (4):

$$\rho_k = \frac{2|\tilde{x}_k|^2}{T_{\text{coh}} S_n(f_k)}. \quad (5)$$

The PSD, in the ideal case of a continuous range of values of  $t$  and  $T_{obs} \rightarrow \infty$ , could be defined for  $f \geq 0$  as the Fourier transform of the following auto-correlation function [3]:

$$S_n(f) = 2 \int_{-\infty}^{\infty} \langle n(t)n(0) \rangle e^{-2\pi i f t} dt, \quad (6)$$

where, understandably, it must not be influenced by any possibly present signal power in any case or manner. However, for the focused scope of this study, a theoretical noise curve constructed for the Advanced LIGO detectors [4] will be used as the reference PSD, expressed as follows [5]:

$$S_n(f) \equiv S_{n,theo}(f) = \begin{cases} \infty & \text{if } f < f_c, \\ S_0 \left[ x^{-4.14} - 5x^{-2} + \frac{111(1-x^2+\frac{x^4}{2})}{1+\frac{x^2}{2}} \right] & \text{if } f \geq f_c, \end{cases} \quad (7)$$

where  $x = \frac{f}{f_0}$  with the convenient scaling factor  $f_0 = 215$  Hz,  $f_c = 10$  Hz is the lower cut-off frequency and  $S_0 = 10^{-49} \text{ Hz}^{-1}$  is the noise curve “amplitude”. (7) is represented in Figure 1, keeping in mind that, computationally,  $\infty \equiv 0$  is assumed.

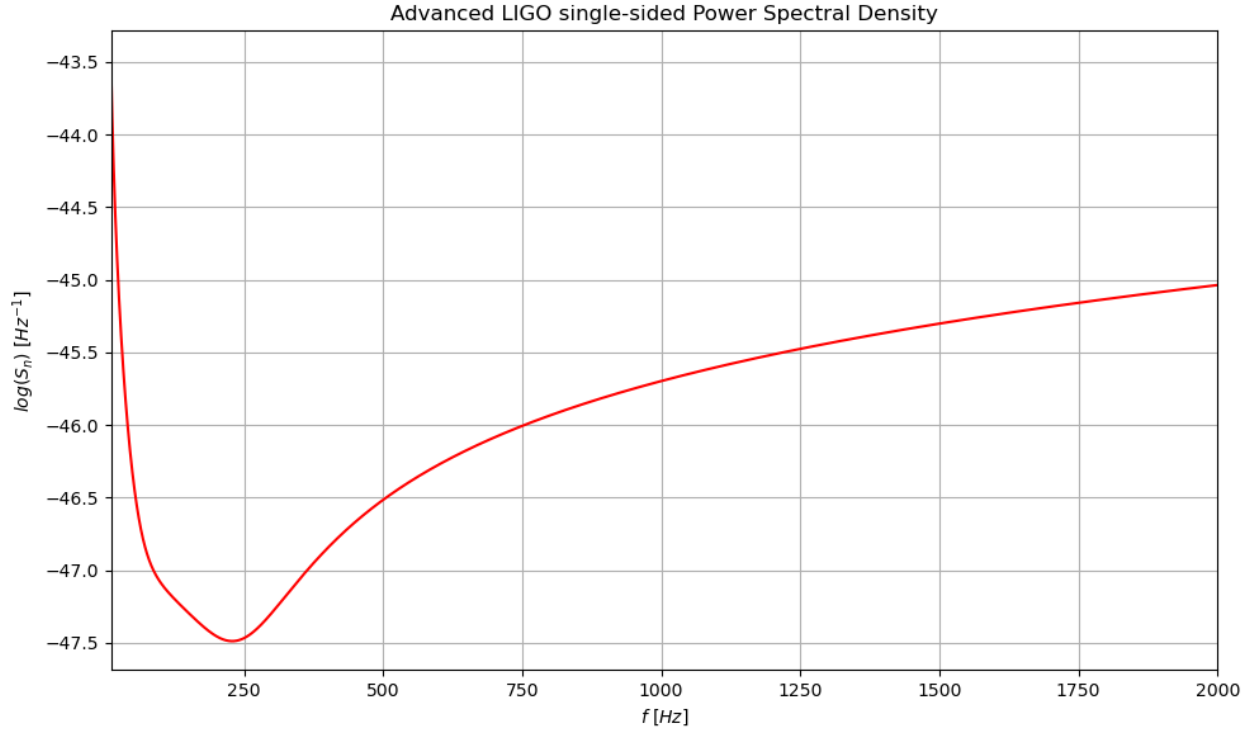


Figure 1: Advanced LIGO PSD from (7), represented in a range of frequencies  $[f_c, f_{Nyq}]$ , and in common logarithmic scale for easier visualization.

Expressing (4) differently, it results in:

$$2\rho_k = z_1^2 + z_2^2 \Rightarrow \frac{z_1^2 + z_2^2}{2} = \frac{|\tilde{x}_k|^2}{\langle |\tilde{n}_k|^2 \rangle}, \quad (8)$$

where  $z_1 = \frac{\sqrt{2}\text{Re}[\tilde{x}_k]}{\sqrt{\langle |\tilde{n}_k|^2 \rangle}}$  and  $z_2 = \frac{\sqrt{2}\text{Im}[\tilde{x}_k]}{\sqrt{\langle |\tilde{n}_k|^2 \rangle}}$  with  $\text{Re}[\dots]$  and  $\text{Im}[\dots]$  representing the real and imaginary parts, respectively. From there, the noise detector’s output in the frequency domain for only positive frequencies (to avoid complications

with the square root)  $\tilde{x}_k$  can be obtained using (8) and the definitions of  $z_1$ ,  $z_2$  and  $\langle |\tilde{n}_k|^2 \rangle$ :

$$\tilde{x}_k = \sqrt{\frac{\langle |\tilde{n}_k|^2 \rangle}{2}}(z_1 + iz_2) \approx \frac{1}{2}\sqrt{T_{coh}S_n(f_k)}(z_1 + iz_2), \quad (9)$$

where, if it is considered that  $\text{Re}[\tilde{n}_k]$  and  $\text{Im}[\tilde{n}_k]$  are independent random variables with identical variance and the Gaussian nature of the noise, then  $\text{Re}[\tilde{n}_k]$  and  $\text{Im}[\tilde{n}_k]$  have variances equal to  $\frac{\langle |\tilde{n}_k|^2 \rangle}{2}$  and  $z_1$  and  $z_2$  are normally/-Gaussian distributed with unit variance and nonzero mean [3]. To complete  $\tilde{x}_k$  with the values corresponding to negative frequencies, the complex conjugate of the values with positive frequencies is computed and then shifted to negative values to sort them on the frequency axis. Finally, to obtain the respective noise signal in the time domain, the inverse Fourier transformation is used, as expressed in (3).

The following code snippet summarizes the previously explained steps in a computational manner:

```

1 Random.seed!(1418) # Random noise seed setting (to always have the same random result)
2 f_c = 10 # Lower cut-off frequency (Units: Hz)
3 function Sn(f) # Theoretical noise/sensitivity curve (PSD) for Advanced LIGO (Units: Hz-1)
4     f_0 = 215 # Scaling factor (Units: Hz)
5     S_0 = 10-49 # "Amplitude" (Units: Hz-1)
6     x = f./f_0
7     return ifelse.(f .>= f_c, S_0*(x.-4.14) .- 5*x.-2) .+ 111*(1 .- x.2 .+ x.4/2)./(1 .+ x.2/2), 0)
8     end
9
10 # main_program
11
12 f_array = fftshift(fftfreq(N, f_s)) # Frequency array computation
13 f_pos_array = f_array[findall(f_array .>= 0)] # Only positive frequencies array
14
15 # Computing of the positive noise values in the frequency domain
16 x_k_pos_array = 0.5 .* (randn(length(f_pos_array)) .+ 1im .* randn(length(f_pos_array)))
17 ) .* sqrt.(T_coh .* Sn(f_pos_array))
18
19 # Filling the noise array with the remaining values corresponding to negative frequencies
20 x_k_array = [conj.(reverse(x_k_pos_array)); 0; x_k_pos_array[1:end-1]]
21
22 x_k_sort_array = ifftshift(x_k_array) # Sort the noise array from lowest to highest frequencies
23 x_j_array = ifft(x_k_sort_array)/Deltat # Transform the noise array to the time domain

```

Listing 2: Noise creation.

## 2.3 Computation of Power Spectral Densities (PSDs)

For the construction of the PSDs with the generated noise, the real part of the noise signal will be used exclusively (discarding the imaginary part in this context), and it will be constructed entirely numerically using unbiased periodograms for simplicity and convenience. Specifically, in the Julia Programming Language, the DSP package [6] will be used with two distinct functions/periodograms: `DSP.Periodograms.periodogram`, which computes the PSD purely by using the Fast Fourier Transform (FFT) algorithm with no data windowing method applied, and `DSP.Periodograms.welch_pgram`, which utilizes the Welch method to ensemble an average periodogram of a divided signal  $s$  with its respective periodograms of the  $M$  intervals (each having  $n$  samples), windowing the

data (in this case, using the typical Hann's/Hanning's window with 50% of data overlap between sections) in each segment to reduce spectral leakage. In the following code listing, the computational implementation of the FFT and the Welch periodograms (with different  $M$ s selected for comparison purposes) can be observed:

```

1  # Computation of the periodograms
2
3  # Periodogram using only the FFT estimation
4  FFT_array = periodogram(
5  real(x_j_array), # Detector/noise output
6  fs = f_s, # Frequency sampling
7  window = nothing # No window method applied
8  )
9
10 # Welch method for different amounts of intervals
11 # 2 intervals (M = 2)
12 Welch2_array = welch_pgram(
13 real(x_j_array), # Detector/noise output
14 n = div(length(real(x_j_array)),2), # Samples in each interval
15 fs = f_s, # Frequency sampling
16 window = hanning , # Hann window method
17 )
18 # 4 intervals (M = 4)
19 Welch4_array = welch_pgram(
20 real(x_j_array), # Detector/noise output
21 n = div(length(real(x_j_array)),4), # Samples in each interval
22 fs = f_s, # Frequency sampling
23 window = hanning , # Hann window method
24 )
25 # 10 intervals (M = 10)
26 Welch10_array = welch_pgram(
27 real(x_j_array), # Detector/noise output
28 n = div(length(real(x_j_array)),10), # Samples in each interval
29 fs = f_s, # Frequency sampling
30 window = hanning , # Hann window method
31 )
32 # Frequency axis for the different periodograms
33 f_FFT_array = range(f_i,f_f,length(FFT_array.power))
34 f_Welch2_array = range(f_i,f_f,length(Welch2_array.power))
35 f_Welch4_array = range(f_i,f_f,length(Welch4_array.power))
36 f_Welch10_array = range(f_i,f_f,length(Welch10_array.power))

```

Listing 3: PSDs estimations.

### 3 Results and Discussion

Following the procedure previously explained, in Listing 2, the noise signal can be obtained in both the frequency domain and the time domain, as represented in Figure 2 and Figure 3, respectively.

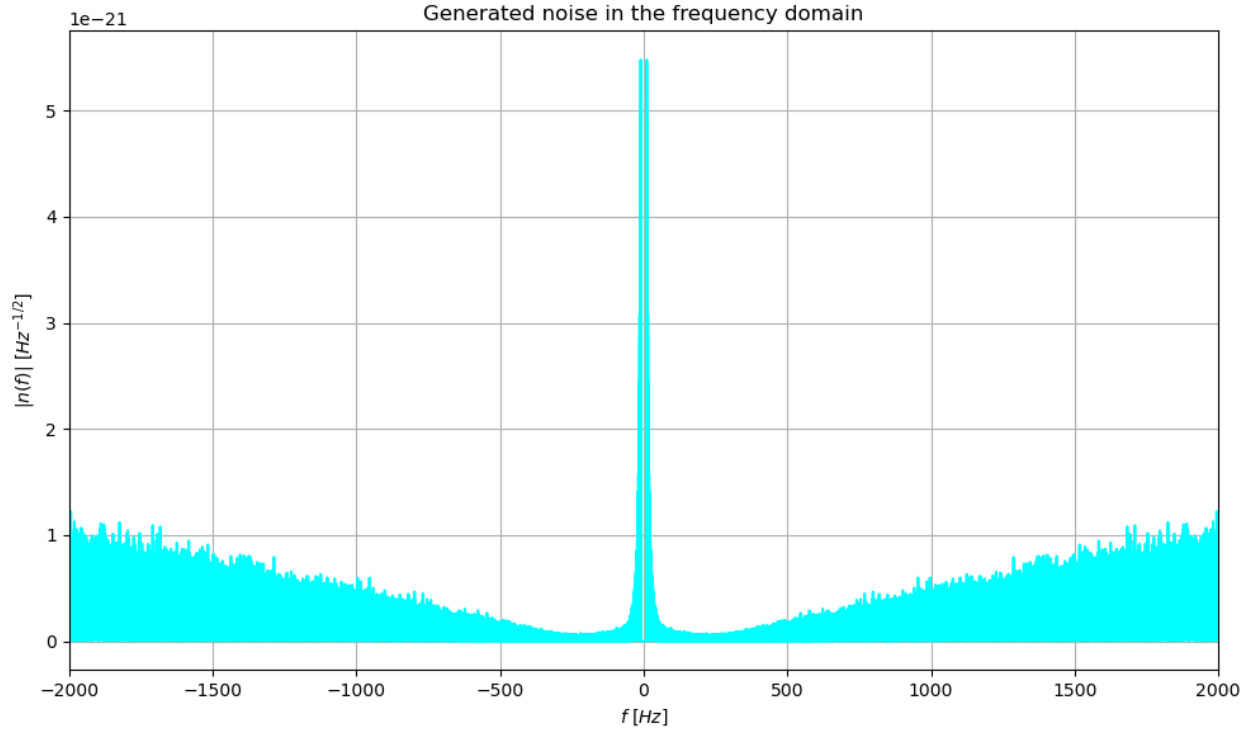


Figure 2: Complete noise signal  $\tilde{x}_k \equiv n(f)$  in the frequency domain and in absolute value, represented in a frequency range  $[-f_{Nyq}, f_{Nyq}]$ .

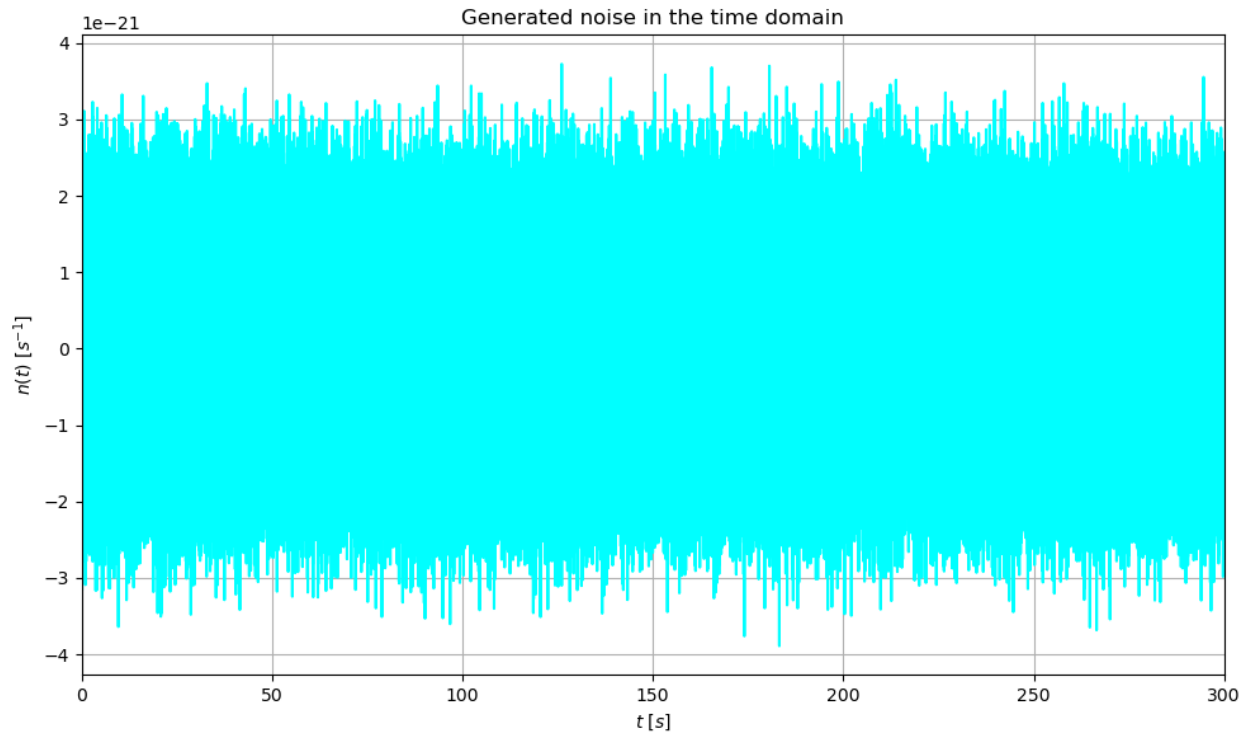


Figure 3: Noise signal  $x_j \equiv n(t)$  in the time domain, represented in a time range  $[0, T_{obs}]$ .

As can be seen in Figure 2 and Figure 3, the real/time component of the noise is of order  $\mathcal{O} \sim 10^{-21}$  and the imaginary/frequency component is of order  $\mathcal{O} \sim 10^{-21} - 10^{-25}$ , which orders of magnitude similar to GWs signals  $h(t)$  in this frequency range.

Finally, from Listing 2 and Listing 3, the different PSDs and periodograms are computed, including the reference noise curve  $S_{n,theo}$ , the FFT periodogram and the Welch periodograms with different numbers of segments  $M$  (specifically,  $M = 2, 4$  and  $10$ ). A comparison between them can be seen in Figure 4.

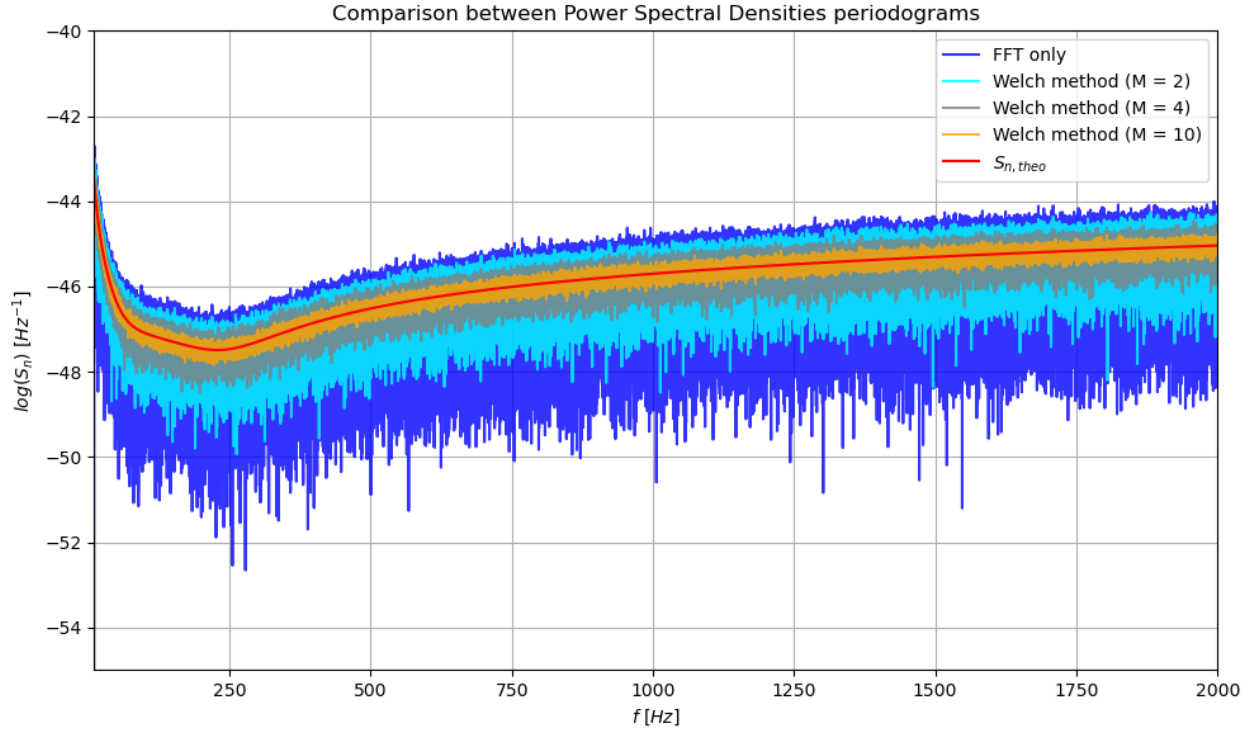


Figure 4: Different single-sided PSDs recovered using the DSP.jl package, all represented in common logarithmic scale for easier visualization. The reference noise curve  $S_{n,theo}$  is marked in red. The same sampling frequency used to construct the frequency sequences is utilized for the periodograms.

As can be seen in Figure 4, the type of periodogram/method applied and the function parameters (specifically,  $M$  and, thus  $n$ ) significantly affect the coarseness of the results. For instance, the FFT method results in a more imprecise outcome compared to the Welch method, whereas increasing  $M$  (and as  $n$  approaches  $f_s$ ) in the Welch method yields cleaner results that are closer to the reference noise curve.

## References

- [1] Lucas Romero Fernández. Linearized theory of gravity: Description and detection of its effects, 2025.
- [2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. The julia programming language. <https://julialang.org/>. Accessed the 29/03/25.
- [3] Badri Krishnan, Alicia M. Sintes, Maria Alessandra Papa, Bernard F. Schutz, Sergio Frasca, and Cristiano Palomba. Hough transform search for continuous gravitational waves. *Physical Review D*, 70(8), October 2004.

- [4] The LIGO Scientific Collaboration: J. Aasi et al. Advanced ligo. *Classical and Quantum Gravity*, 32(7):074001, mar 2015.
- [5] Manuel Luna and Alicia M Sintes. Parameter estimation of compact binaries using the inspiral and ringdown waveforms. *Classical and Quantum Gravity*, 23(11):3763, may 2006.
- [6] The Julia Programming Language. Dsp.jl: Periodograms - periodogram estimation. <https://docs.juliadsp.org/stable/periodograms/>. Accessed the 30/03/25.

## Full code listing

```

1  # Characterization of Noise in Gravitational Waves (GWs) using Power Spectral Densities (PSDs)
2  # Computed in Julia 1.11.3
3  # author: Lucas Romero Fernández 2025
4  # Start with "julia Characterization_of_Noise_in_Gravitational_Waves_using_Power_Spectral_Densities.jl"
5
6  using DSP # For the periodograms
7  using FFTW # For the Discrete Fourier Transforms (DFTs)
8  using LaTeXStrings # For the plots of the results
9  using PyPlot # For the plots of the results
10 using Random # For the generation of random noise.
11
12 # Definition of general constants, variables, arrays and functions
13
14 T_coh = 300 # Coherent time (in this case, equivalent to the observation time of 5 minutes) (Units: s)
15 f_Nyq = 2000 # Nyquist frequency (Units: Hz)
16 f_i = 0 # Initial frequency (Units: Hz)
17 f_f = f_Nyq # End frequency (Units: Hz)
18 f_s = 2*(f_f - f_i) # Sampling frequency (Units: Hz)
19 Deltat = 1/f_s # Time step size (Units: s)
20 N = T_coh*f_s # Number of sample points
21 t_array = collect(0:N-1)*Deltat
22 Random.seed!(1418) # Random noise seed setting (to always have the same random result)
23 f_c = 10 # Lower cut-off frequency (Units: Hz)
24 function Sn(f) # Theoretical noise/sensitivity curve (PSD) for Advanced LIGO (Units: Hz-1)
25     f_0 = 215 # Scaling factor (Units: Hz)
26     S_0 = 10-49 # "Amplitude" (Units: Hz-1)
27     x = f./f_0
28     return ifelse.(f .>= f_c, S_0*(x.-4.14 .- 5*x.-2 .+ 111*(1 .- x.2 .+ x.4/2)./(1 .+ x.2/2)), 0)
29 end
30
31 # main_program
32
33 f_array = fftshift(fftfreq(N,f_s)) # Frequency array computation
34 f_pos_array = f_array[findall(f_array .>= 0)] # Only positive frequencies array
35
36 # Computing of the positive noise values in the frequency domain
37 x_k_pos_array = 0.5 .* (randn(length(f_pos_array)) .+ 1im .* randn(length(f_pos_array)))
38 ) .* sqrt.(T_coh .* Sn(f_pos_array))

```



```

39
40 # Filling the noise array with the remaining values corresponding to negative frequencies
41 x_k_array = [conj.(reverse(x_k_pos_array));0;x_k_pos_array[1:end-1]]
42
43 x_k_sort_array = ifftshift(x_k_array) # Sort the noise array from lowest to highest frequencies
44 x_j_array = ifft(x_k_sort_array)/Deltat # Transform the noise array to the time domain
45
46 # Computation of the periodograms
47
48 # Periodogram using only the FFT estimation
49 FFT_array = periodogram(
50     real(x_j_array), # Detector/noise output
51     fs = f_s, # Frequency sampling
52     window = nothing # No window method applied
53 )
54 # Welch method for different amount of intervals
55 # 2 intervals (M = 2)
56 Welch2_array = welch_pgram(
57     real(x_j_array), # Detector/noise output
58     n = div(length(real(x_j_array)),2), # Samples in each interval
59     fs = f_s, # Frequency sampling
60     window = hanning, # Hann window method
61 )
62 # 4 intervals (M = 4)
63 Welch4_array = welch_pgram(
64     real(x_j_array), # Detector/noise output
65     n = div(length(real(x_j_array)),4), # Samples in each interval
66     fs = f_s, # Frequency sampling
67     window = hanning, # Hann window method
68 )
69 # 10 intervals (M = 10)
70 Welch10_array = welch_pgram(
71     real(x_j_array), # Detector/noise output
72     n = div(length(real(x_j_array)),10), # Samples in each interval
73     fs = f_s, # Frequency sampling
74     window = hanning, # Hann window method
75 )
76 # Frequency axis for the different periodograms
77 f_FFT_array = range(f_i,f_f,length(FFT_array.power))
78 f_Welch2_array = range(f_i,f_f,length(Welch2_array.power))
79 f_Welch4_array = range(f_i,f_f,length(Welch4_array.power))
80 f_Welch10_array = range(f_i,f_f,length(Welch10_array.power))
81
82 # Plots
83
84 # Theoretical noise/sensitivity curve (PSD)
85 figure(figsize = (10,6))
86 plot(f_pos_array,log10.(Sn(f_pos_array)),c = "red")
87 xlim(f_c,f_f)
88 xlabel(L"f\ [Hz]")

```

```

89 ylabel(L"log(S_{n})\ [Hz^{-1}]")
90 grid()
91 title("Advanced LIGO single-sided Power Spectral Density")
92 tight_layout()
93 savefig("Theoretical noise curve")
94 show()
95
96 # Generated noise in the frequency domain
97 plt.figure(figsize = (10,6))
98 xlim(-f_f,f_f)
99 plt.plot(f_array,abs.(x_k_array),c = "cyan")
100 plt.xlabel(L"f\ [Hz]")
101 plt.ylabel(L"|n(f)|\ [Hz^{-1/2}]")
102 plt.grid()
103 plt.title("Generated noise in the frequency domain")
104 plt.tight_layout()
105 plt.savefig("Generated noise (Frequency domain)")
106 plt.show()
107
108 # Generated noise in the time domain
109 plt.figure(figsize = (10,6))
110 plt.xlim(0,T_coh)
111 plt.plot(t_array,real(x_j_array),c = "cyan")
112 plt.xlabel(L"t\ [s]")
113 plt.ylabel(L"n(t)\ [s^{-1}]")
114 plt.grid()
115 plt.title("Generated noise in the time domain")
116 plt.tight_layout()
117 plt.savefig("Generated noise (Time domain)")
118 plt.show()
119
120 # Comparison between PSDs
121 plt.figure(figsize = (10,6))
122 plt.xlim(f_c,f_f)
123 plt.ylim(-55,-40)
124 plt.plot(f_FFT_array,log10.(FFT_array.power),c = "blue",label = "FFT only",alpha = 0.8)
125 plt.plot(f_Welch2_array,log10.(Welch2_array.power),c = "cyan",label = "Welch method (M = 2)",alpha = 0.8)
126 plt.plot(f_Welch4_array,log10.(Welch4_array.power),c = "grey",label = "Welch method (M = 4)",alpha = 0.8)
127 plt.plot(f_Welch10_array,log10.(Welch10_array.power),c = "orange",label = "Welch method (M = 10)",alpha = 0.8)
128 plt.plot(f_pos_array,log10.(Sn(f_pos_array)),c = "red",label = L"S_{n,theo}")
129 plt.xlabel(L"f\ [Hz]")
130 plt.ylabel(L"log(S_{n})\ [Hz^{-1}]")
131 plt.grid()
132 plt.title("Comparison between Power Spectral Densities periodograms")
133 plt.legend()
134 plt.tight_layout()
135 plt.savefig("Comparison between PSDs")
136 plt.show()

```