Semi-coherent searches for continuous gravitational waves, and the $\mathbf{N^{1/4}}$ law

Graham Woan

7 June 2021
LIGO-T2100266

Abstract

We revisit the relationship between sensitivity and the number of coherent segments in a semi-coherent search for continuous gravitational wave signals. Using the simplest model, we give a short tutorial on how to set the sensitivity level for such a search, how the search sensitivity is degraded as the number of coherent segments (N) increases and how to compute this factor using standard SciPy functions. Finally we show the origins of the well-known $N^{1/4}$ law from both a frequentist and Bayesian perspective.

1 Introduction

Searches for continuous gravitational waves are generally limited by computing power, and over the past 20 years a number of strategies and analysis pipelines have been developed to maintaining reasonable sensitivity whilst significantly reducing the processing burden. Most of these are “semi-coherent” analyses: rather than track a possible signal phase for an entire observing run (which may last for several years), the data is divided into N contiguous segments of shorter duration. These segments are searched in a phase-sensitive (coherent) manner but the results of these searches are combined incoherently. From earliest times we have known that this process needs massively less processing power than a fully-coherent search over the same data duration, but that the overall sensitivity is reduced by a factor of about $N^{1/4}$. Though well-founded and widely accepted, this result appears in the literature within specific and complex search pipelines and its origin can be difficult to pin down. As a result, the “$N^{1/4}$ law” risks becoming folklore.

In this technical note we use a toy model to demonstrate the relative sensitivity of a semi-coherent search as a function of its segmentation, and provide a simple Python function to compute it. In addition, we give an more intuitive explanation of its origin from both a frequentist and a Bayesian perspective.

This generic analysis has another application: recent discussions concerning data release policy in LIGO have highlighted the potential issue of “Swiss cheese” analysis, in which a sensitive continuous-wave search is performed outside the collaboration using only the data released around CBC events and before the official data release of the full observing run. Clearly this issue becomes more relevant as the number of CBC events increases. This note will help in computing its impact.

2 Semi-coherent statistics

We begin with a short tutorial, and consider a simple model of a sinusoidal signal in a real time-series of M samples (M even), containing Gaussian noise of variance $2\sigma^2$ (the factor of two here simplifies later expressions and has no other significance). There are $M/2 + 1$ frequency bins in this (real-to-complex) transform, and for clarity we will consider the situation in which the signal frequency falls exactly at the centre frequency of a bin.

We chop the full time-series of length T into N contiguous segments, each of length M, determine a power spectrum for each segment and then add these power spectra to form a single overall semi-coherent power spectrum for further analysis. This is a highly-simplified version of real continuous-wave semi-coherent search pipelines, but it contains the essential ingredients. Our goal is to determine the relationship between the smallest detectable signal amplitude and N, for a given false alarm and false dismissal probability. Generally we do not know the frequency bin that contains the signal, so we need to set the false alarm threshold high enough for noise to not exceed a certain level in *any* bin. There is therefore a significant additional "trials factor" to consider.

2.1 Setting the false alarm threshold

In the signal-free case, each single spectral channel in the Fourier transform of a segment contains complex Gaussian noise of variance σ^2 in each of the real and imaginary parts (by equipartition and Parseval's theorem). The square-modulus (power) spectrum of this has a χ^2 distribution with two degrees of freedom,

$$p(y_1) = \frac{1}{2\sigma^2} \exp\left(-\frac{y_1}{2\sigma^2}\right), \tag{1}$$

where y_1 is spectral power, here with a mean of $2\sigma^2$ and a variance of $4\sigma^4$.

The sum of N similar spectra gives a total noise power measurement that follows a χ^2 distribution but now with $2N$ degrees of freedom,

$$p_N(y) = \frac{(2\sigma^2)^{-N}}{(N-1)!} y^{N-1} \exp\left[-\frac{y}{2\sigma^2}\right], \tag{2}$$

and the cumulative probability distribution to a threshold y_t is

$$C_N(y_\mathrm{t}) = \int_0^{y_\mathrm{t}} p_N(y)\,\mathrm{d}y. \tag{3}$$

This function is available in the Python `scipy.stats` module as `chi2.cdf`. Using our measures

$$C_N(y_\mathrm{t}) \equiv \texttt{chi2.cdf(y\_t/sigma} * *\texttt{2}, \texttt{2} * \texttt{N)} \tag{4}$$

is therefore the probability that $y < y_\mathrm{t}$ in a single frequency bin of the summed spectrum.

The joint probability that all of the $M/2+1 \simeq M/2$ independent frequency bins contain noise power less than y_t is $[C_N(y_\mathrm{t})]^{M/2}$. If we set an overall false alarm probability of p_fa, our detection threshold is the value of y_t satisfying $p_\mathrm{fa} = 1 - [C_N(y_\mathrm{t})]^{M/2}$. Given p_fa is small, this approximates to solving

$$[1 - C_N(N, y_\mathrm{t})] - 2p_\mathrm{fa}/M = 0 \tag{5}$$

for y_t.

2.2 Sensitivity

We can now introduce a signal and determine how strong it has to be to pass this false alarm threshold with a high probability. We model the signal as a simple sinusoid of fixed (but possibly unknown) frequency ν and amplitude h in a real time-series, i.e.

$$v_i = h\cos(2\pi\nu t_i) + n_i, \tag{6}$$

where n_i is the noise value (of variance $2\sigma^2$) at time t_i and the index i runs over all samples.

If this time-series has M samples, and for simplicity if ν falls at the centre of a bin, the signal in the frequency domain is purely real with an amplitude of $Mh/2$, scaled by $1/\sqrt{M}$ by the conventional symmetric normalisation[1]. The bin in the power spectrum of a single segment that contains the signal has a non-central χ^2 distribution, with two degrees of freedom and a non-centrality parameter of $Mh^2/4$. Again, this is available in `scipy` as `ncx2.pdf(y/(sigma**2), 2, M*h*h/4/(sigma**2))`.

If we sum N of these spectra, the bin containing the signal now has a non-central χ^2 distribution $f(y)$ with $2N$ degrees of freedom, and a non-centrality parameter of $NMh^2/4$. The probability that this value is below the detection threshold (defined above) as a function of h is the false dismissal probability at that signal level p_{fd}, i.e.

$$\begin{aligned} p_{\mathrm{fd}} &= \int_0^{y_t(p_{\mathrm{fa}})} f(y : 2N,\, NMh^2/4)\,\mathrm{d}y \\ &\equiv \mathtt{ncx2.cdf(y/(sigma**2), 2*N, N*M*h*h/4/(sigma**2))}. \end{aligned} \tag{7}$$

Solving this equation for h gives us our signal amplitude sensitivity for a given false alarm and false dismissal probability.

3 Example

We can use typical numbers for a semi-coherent search over one year of data to demonstrate how sensitivity is affected by the data segmentation. The total number of samples over the year for a search up to 1 kHz will be approximately $T = 2000 \cdot 3600 \cdot 24 \cdot 365.25$. We use the Python function in Appendix A, which implements the analysis above, to compute the sensitivity of a semi-coherent search comprising N segments relative to a fully coherent search ($N = 1$). In addition to the case considered above where the frequency of the signal is not known, we also include the relative sensitivity if the frequency is known (i.e., a targeted search). We should note that the trials factor here is simply dependent on the number of frequency bins in the coherent segments and does not take into account the significantly larger trials factor from (e.g.) frequency derivative and sky position that a real continuous-wave search includes.

Table 1 shows the sensitivity reduction factor for specific segments lengths, compared to a fully coherent search at a known frequency. It is relevant to note that when we include a trials factor an increase in coherence time from 30 minutes to one day brings the semi-coherent search sensitivity to within nearly a factor of two of the fully coherent search sensitivity.

Fig. 1 shows this relative sensitivity as a function of N from $N = 1$ (fully coherent) to $N = 48 \cdot 365.25$ (coherence time of 30 min). The log-log plot also shows the $N^{1/4}$ line for

[1]This is 'ortho' normalisation in `NumPy`.

coherence time	N	F_{trials}	F
30 min	17532	9.92	7.07
1 day	365	4.22	2.80
1 week	52	2.97	1.85
1 month	12	2.44	1.41
6 months	2	2.14	1.09
1 year	1	2.08	1.00

Table 1: The relative sensitivity of a semi-coherent search over one year as a function of the number of coherent segments N. All values are relative to a fully coherent search at a known frequency. Column F shows the sensitivity reduction factor when the frequency is known, and column F_{trials} when the frequency is unknown. reduction.

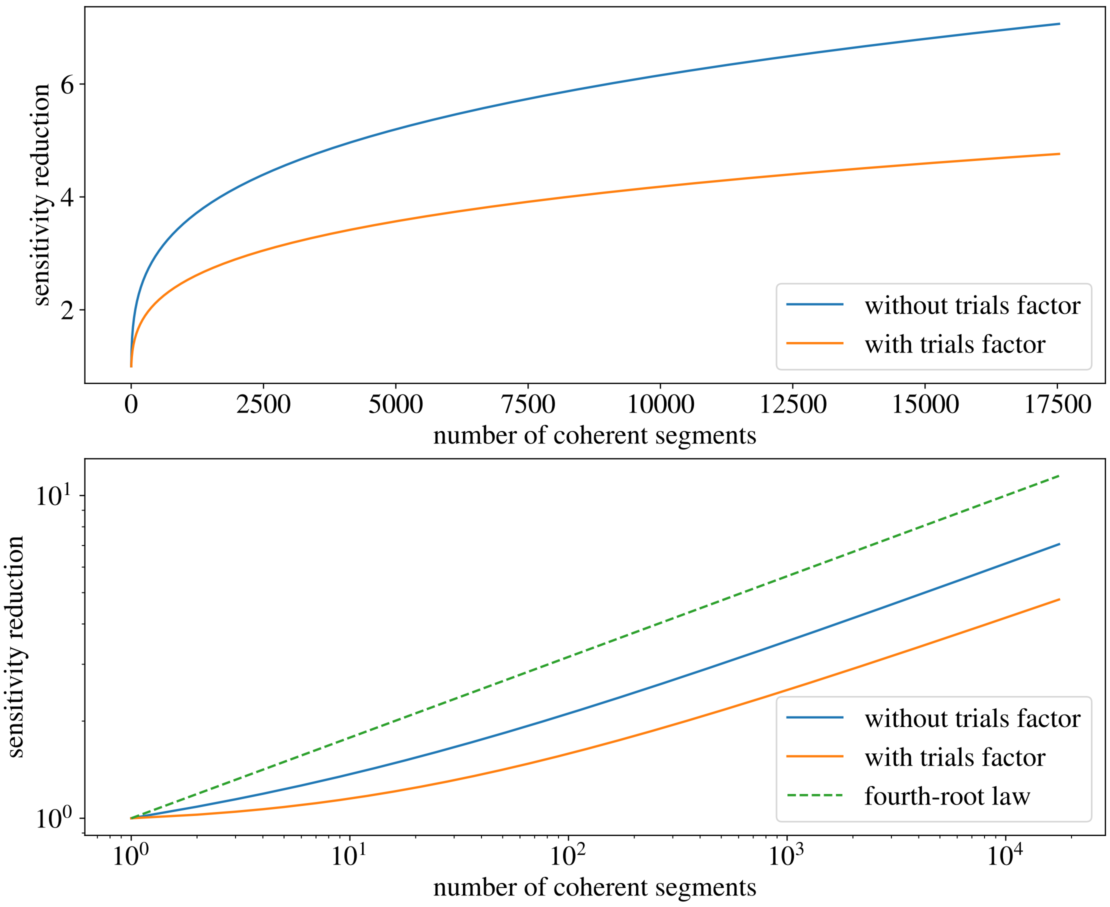

Figure 1: The relative sensitivities of semi-coherent searches for a sinusoidal signal in a one year time-series divided into N segments. The maximum value of N corresponds to a coherence time of 30 minutes. The two curves show the effect of including and excluding a search over frequency. In all cases the false alarm probability is 0.05 and the false dismissal probability 0.01. The log-log plot includes a slope corresponding to a $N^{1/4}$ law.

comparison. It is clear that for large N the search sensitivity degrades as approximately $N^{1/4}$ compared with the coherent search, and this slope is not affected by the trials factor. This dependence on N can be understood by considering the relationship between the signal-to-noise ratio of the signal in the time-series, and its signal-to-noise ratio in a power spectrum, which we will now investigate.

4 Signal-to-noise ratios, and the origins of the $N^{1/4}$ law

We model our signal in the frequency domain as complex, with amplitude A_0 and unknown phase, embedded in symmetric Gaussian noise n of variance σ^2 in each of the real and imaginary (x, y) components. We define the components as

$$A_0^2 = A_{0x}^2 + A_{0y}^2 \tag{8}$$
$$Z_x = A_{0x} + n_x \tag{9}$$
$$Z_y = A_{0x} + n_y \tag{10}$$

where $\text{var}[n_x] = \text{var}[n_y] = \sigma^2$. We can now determine the signal-to-noise ratio present before and after computing the square-modulus of the data.

The signal-to-noise ratio (snr) of a parameter is the ratio of the parameter's mean unbiased estimator $\langle E\rangle$ to the standard deviation of that estimator:

$$\text{snr} = \frac{\langle E\rangle}{(\text{var}[E])^{1/2}}. \tag{11}$$

If we know the phase of the signal we can rotate it to be purely real and use this real component as the unbiased estimator, resulting in an snr of

$$\gamma = \frac{A_0}{\sigma}. \tag{12}$$

Generally however the phase is not known. The Bayesian approach to this is presented in the next section, but for now we will simply assume that we use

$$E = Z_x^2 + Z_y^2 - 2\sigma^2 \tag{13}$$

as our estimator, which is unbiased in A_0^2 in the sense that

$$\langle E\rangle = A_{0x}^2 + A_{0y}^2 + \langle n_x^2\rangle + \langle n_y^2\rangle - 2\sigma^2 = A_0^2. \tag{14}$$

The snr for this estimator is

$$\Gamma = \frac{\langle E\rangle}{(\text{var}[E])^{1/2}} = \frac{A_0^2}{(\langle E^2\rangle - A_0^4)^{1/2}}. \tag{15}$$

Applying Isserlis' Theorem

$$\langle abcd\rangle = \langle ab\rangle\langle cd\rangle + \langle ac\rangle\langle bd\rangle + \langle ad\rangle\langle bc\rangle \tag{16}$$

for Gaussian zero-mean variables a, b, c, d we now expand the $\langle E^2\rangle$ term to give

$$\langle E^2\rangle = A_0^4 + 4\sigma^4 + 4\sigma^2 A_0^2,$$

so now

$$\mathit{\Gamma} = \frac{A_0^2}{2(\sigma^4 + A_0^2\sigma^2)^{1/2}}, \tag{17}$$

which we can rewrite in terms of $\gamma = A_0/\sigma$ as

$$\mathit{\Gamma} = \frac{\gamma^2}{2(1+\gamma^2)^{1/2}}. \tag{18}$$

Equation (18) is *fundamental* to understanding the $N^{1/4}$ law in semi-coherent searches. It quantifies the reduction in sensitivity that occurs when phase information is lost. The snr of the estimator generated from the power spectrum ($\mathit{\Gamma}$) is always less than the intrinsic snr of a frequency domain signal of known phase γ. Indeed, if $\gamma \ll 1$ we have

$$\mathit{\Gamma} \simeq \frac{\gamma^2}{2}, \tag{19}$$

and the snr in the power spectrum is very severely degraded.

When we perform a semi-coherent search we set a detection threshold on the final summed power spectrum, defined by a final snr $\mathit{\Gamma}_\mathrm{F}$. This spectrum is the result of summing N independent segment spectra, so for stationary noise in the Gaussian limit[2] the snr of each segment is simply

$$\mathit{\Gamma}_\mathrm{s} = \frac{\mathit{\Gamma}_\mathrm{F}}{N^{1/2}}. \tag{20}$$

If the total number of samples in the full data is T, the number of samples in a segment is $M = T/N$ and the frequency domain snr of our signal in a segment is

$$\gamma = \frac{M^{1/2}h}{2\sigma} = \gamma_\mathrm{i} M^{1/2}, \tag{21}$$

where γ_i is the initial snr, in a single time-series sample. If our signal is at threshold in the final summed spectrum, and the number of segments N is large, then γ and $\mathit{\Gamma}_\mathrm{s}$ will be necessarily small in each segment and

$$\mathit{\Gamma}_\mathrm{s} \simeq \frac{\gamma^2}{2} = \frac{\gamma_\mathrm{i}^2 M}{2} = \gamma_i^2 \frac{T}{2N}. \tag{22}$$

But the final snr on which we place the threshold is $\mathit{\Gamma}_\mathrm{F} = \mathit{\Gamma}_\mathrm{s} N^{1/2}$, so

$$\gamma_\mathrm{i}^2 = \frac{2\mathit{\Gamma}_\mathrm{F} N^{1/2}}{T}. \tag{23}$$

At the threshold strain sensitivity h_t, $\gamma_\mathrm{i} = h_\mathrm{t}/(2\sqrt{2}\sigma)$, so we can say that our semi-coherent search sensitivity varies as

$$h_\mathrm{t} \propto \frac{N^{1/4}}{T^{1/2}}, \tag{24}$$

as the final snr, $\mathit{\Gamma}_\mathrm{F}$, is largely fixed by the false alarm and false dismissal probabilities.

[2]This relation is slightly biased for small N.

If N is small, the snr in each of the segment power spectra Γ_s will necessarily not be small. In this case the approximation in Eqn. (19) is no longer valid and we need to use the full version of Eqn. (18) , giving

$$h_t = 4\sigma \frac{\Gamma_F}{T^{1/2}} \left[1 + \left(1 + \frac{N}{\Gamma_F^2} \right)^{1/2} \right]^{1/2} . \tag{25}$$

Although this sensitivity result is defined in terms of a final statistic signal-to noise-ratio (Γ_F) rather than a false alarm and false dismissal probability the agreement between the two is very good. Empirically, a choice of $\Gamma_F = 1.15$ matches the N-dependence of $p_{fap} = 0.05$ and $p_{fdp} = 0.01$ to within 5 percent for any value of N.

5 A Bayesian comment

Although there is no direct equivalent of signal-to-noise ratio in Bayesian inference there is a straightforward Bayesian interpretation of the impact that the loss of phase information has on estimating signal strength.

As before, the (complex) data in the frequency bin that contains the signal is

$$Z = A_0 \exp(i\phi) + n, \tag{26}$$

where ϕ is the signal phase, n is complex noise and $|Z|^2 \equiv y$. The joint likelihood for A_0 and ϕ is

$$p(Z \,|\, A_0, \phi) \propto \exp\left(-\frac{|Z - A_0 e^{i\phi}|^2}{2\sigma^2} \right), \tag{27}$$

and given ϕ we have a Gaussian likelihood for A_0. However, if ϕ is not known we need to construct a marginal likelihood for A_0 which, for a uniform prior on ϕ, has a Rice distribution

$$p(Z \,|\, A_0) \propto \int_0^{2\pi} \exp\left(-\frac{|Z - A_0 e^{i\phi}|^2}{2\sigma^2} \right) d\phi \tag{28}$$

$$\propto \frac{|Z|}{\sigma^2} \exp\left(-\frac{|Z|^2 + A_0^2}{2\sigma^2} \right) I_0\left(\frac{A_0 |Z|}{\sigma^2} \right), \tag{29}$$

where I_0 is the modified Bessel function of the first kind, order zero. We can express this in terms of the power in the bin, $y = |Z|^2$, to give

$$p(y \,|\, A_0) \propto \exp\left(-\frac{y + A_0^2}{2\sigma^2} \right) I_0\left(\frac{A_0 \sqrt{y}}{\sigma^2} \right). \tag{30}$$

For a uniform prior on A_0 this is the posterior pdf for A_0 when its phase information is lost, but is also a non-central χ^2 distribution with two degrees of freedom and a non-centrality parameter A_0^2, corresponding to the analysis in Sec. 2.2.

A Example Python function

```python
def sensitivity(T,N,fap,fdp,search):
    '''
Computes the relative sensitivity of a semi-coherent search for a sinusoidal
signal in Gaussian noise.  T samples are divided into N segments, and the
power spectra of these segments are summed to give a final semi-coherent
spectrum.  The false alarm probability can take account of, or ignore, the
trials factor that comes with searching over frequency in this final spectrum,
set by the "search" boolean variable.  The function returns the sensitivity
threshold in the final spectrum and amplitude of a signal at this detection
threshold for noise with a variance of 1 in the time domain.

T = total number of samples
N = number of subdivisions (segments)
fap = false alarm probability (0...0.1 note restricted range)
fdp = false dismissal probability (0...1)
search (boolean) whether to include the trials factor from a frequency search
over the spectrum
    '''
    import numpy as np
    from scipy.optimize import minimize_scalar
    from scipy.stats import chi2
    from scipy.stats import ncx2

    def funfap(y,N,M,fap, trials):
        '''
function to be minimised over y to determine the false alarm probability
threhold level, taking account of the trails factor over frequency channels.
        '''
        return (1-chi2.cdf(2*y, 2*N) - fap/trials)**2
    def funfdp(A,thresh,N,M,fdp):
        '''
function to be minimised over A to determine the sensitivity level for the
given false dismissal probability.
        '''
        return (ncx2.cdf(2*thresh, 2*N, 2*N*M*A*A/4) - fdp)**2

    M=T//N # length of a coherent segment
    if search:
        trials = M/2
    else:
        trials = 1

    result1 = minimize_scalar(funfap, bracket=[0,4*N], args=(N,M,fap,trials),
                              method='Golden', tol=1e-10)
    thresh = result1.x
    result2 = minimize_scalar(funfdp, bracket=[0,1],  args=(thresh,N,M,fdp),
                              method = 'Golden', tol=1e-10)

    if not result1.success and result2.success:
        print("did not converge")

    return thresh, result2.x
```