LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note **LIGO-T2200025-v2-SEI** 2022/08/05

Update to BLRMS Code: Custom BLRMS, Transposed Direct Form 2, and Reduced DC Order

Nathan A. Holland, Jim Warner, Brian Lantz

We describe an update to the Band-Limited Root-Mean-Square (BLRMS) code to implement 65 Hz to 100 Hz and 130.5 Hz to 200 Hz BLRMS, and resolve some issues with the DC BLRMS. Users may find this technical note helpful if they wish to implement additional custom BLRMS in the future.

California Institute of Technology
LIGO Project, MS 100-36
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project, Room NW22-295
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
P.O. Box 159
Richland, WA 99352
Phone (509) 372-8106
Fax (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (225) 686-3100
Fax (225) 686-7189
E-mail: info@ligo.caltech.edu

http://www.ligo.caltech.edu/

Contents

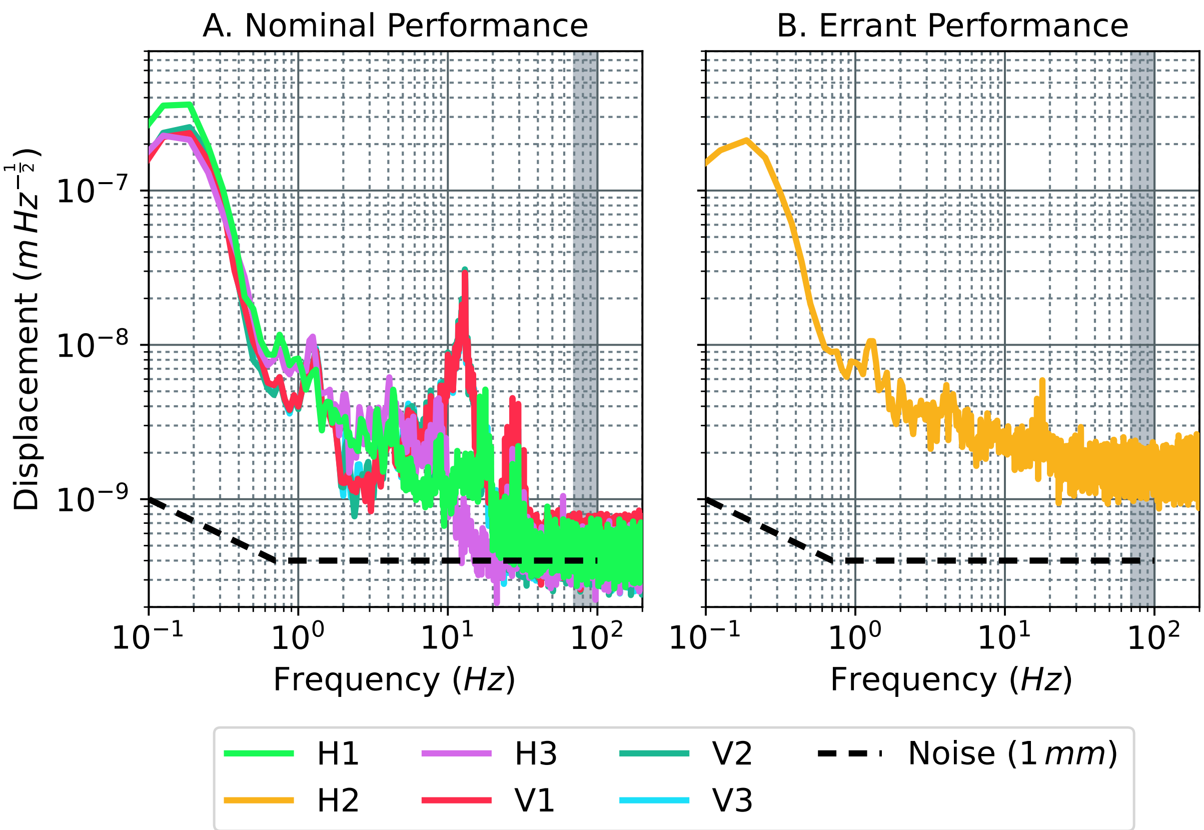

Figure 1: HAM3 CPS displacement spectra from a weekly performance check [7]. At high frequencies the performance should match the sensor noise. This is true for part **A**; 2 horizontal and all 3 vertical sensors. This is not true for **B**, the remaining horizontal sensor, which indicates a fault or issue. We highlight the frequency range 70 Hz to 100 Hz, demonstrating the lack of features in the spectra.

1 Motivation

The HAM and BSC, ISI have commercial Capacitive Position Sensor (CPS) for low frequency coupled sensing. Unfortunately these sometimes fail, or glitch - one such example at LHO is [9][1]. Jennifer Watchi lead an effort to improve the diagnosis of these errors [15]. The code for these diagnostics is available in the Seismic isolation (SEI) SVN [1][2][3]. We show an example of the CPS, nominal and errant behaviour, in figure 1.

There is a desire to automate these checks, which could see them performed periodically, or continuously. The standard tool for this is guardian [6]. Unfortunately guardian operates at a maximum sampling rate of 16 Hz. This is too slow to perform some of the tests necessary, which check between 70 Hz and 100 Hz. We conducted an investigation [3] which verified that all the data processing can be made compatible with guardian. To achieve this, changes to the BLRMS code are required, adding a 65 Hz to 100 Hz BLRMS.

[1] ITMY
[2] /Common/MatlabTools/check_cps_noise_hams_jw.m
[3] /Common/MatlabTools/check_cps_noise_bsc_jw.m

We pursue two opportunistic, additional changes to the BLRMS code; focussed on improving their usefulness. The first is a resolution to the DC (0 Hz) (DC) BLRMS *integrating forever* [14]. Presently this makes these BLRMS unusable. This is addressed in section 3.3.

The second is *offset hopping*, with a contemporaneous example in [11]. These hops introduce offsets into the Suspension (SUS), and by extension Interferometer Sensing and Control (ISC), systems. This glitchy behaviour is coincident with unusual high frequency, $> 100\ Hz$, noise on the CPS. None of the existing BLRMS options are suitable for monitoring this frequency range [12], so a new one is requested. This new, $100\ Hz - 200\ Hz$, BLRMS [13] is detailed in section 3.2.

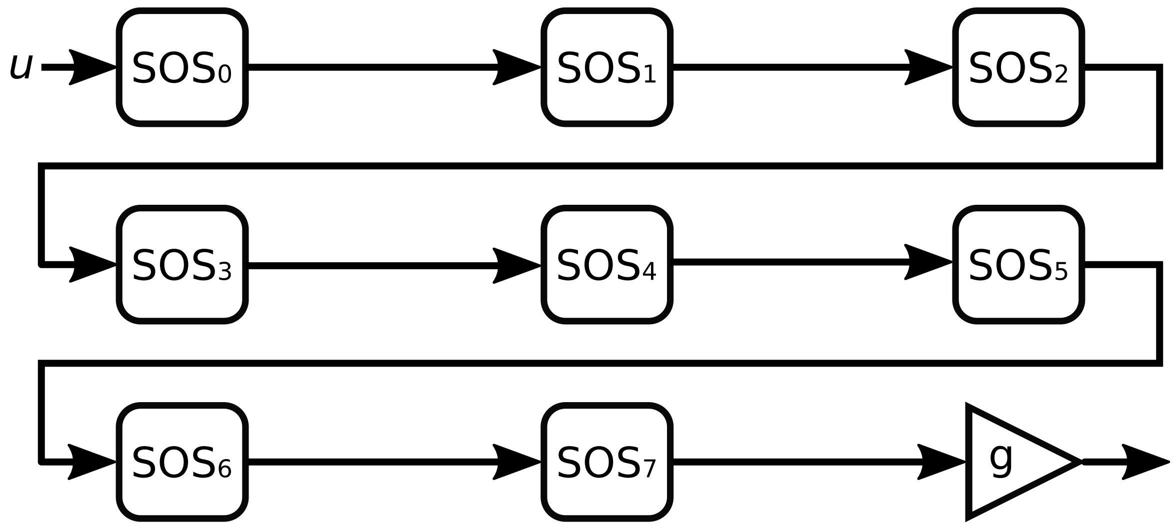

Figure 2: Overview of the operation of the BLRMS, Band Limit filter. Modified cascaded SOS are used to form the BL filter. Each SOS is expanded upon in figure 3. The RMS value of y is the output of the BLRMS. Equation 1 (line *2*) shows the transfer function implementation.

2 Band-Limited Root-Mean-Square Filters

The original work on the BLRMS is described in [4], with source available at [1][4]. Here we refresh how the BLRMS work, with particular attention to the Band Limit (BL) filtering component - which is updated by this work.

The BLRMS operate in groups of 8. Each cycle one BL filter and Root Mean Square (RMS) operation is performed. Consequentially the BLRMS operate at $\frac{1}{8}^{\text{th}}$ of the model's sample rate. No explicit decimation is performed, with the downsampling, then BL filter handling this operation.

$$\begin{aligned} H_{\mathrm{i}}(z) &= \frac{\beta_{\mathrm{i},2}\, z^{-2} \;+\; \beta_{\mathrm{i},1}\, z^{-1} \;+\; 1}{a_{\mathrm{i},2}\, z^{-2} \;+\; a_{\mathrm{i},1}\, z^{-1} \;+\; 1} \\ y(z) &= \left(g \prod_{\mathrm{i}\,=\,0}^{7} H_{7-\mathrm{i}}(z) \right) u(z) \end{aligned} \tag{1}$$

Figure 2 shows the layout, and equation 1 (line *2*) details the operation of the BLRMS, BL filter. The code implementing this is given in appendix A.2. Frequency shaping is achieved with 8, series cascaded, SOS followed by an overall gain, g.

[4] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/

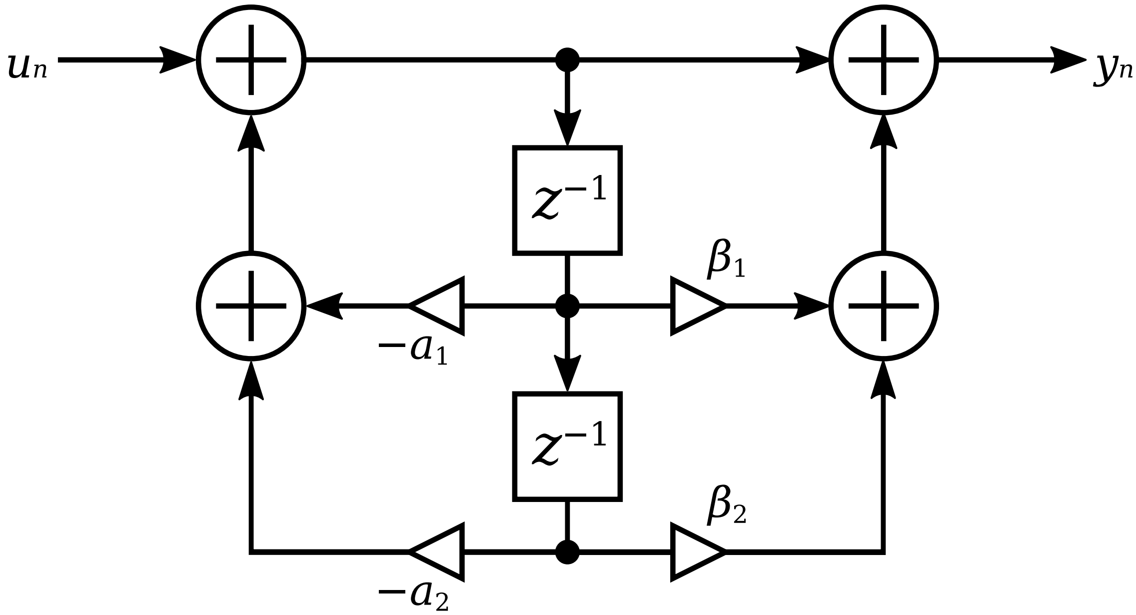

Figure 3: Implementation of each SOS within the BLRMS Band Limit filter. Each is implemented as a modified direct form 2. Equation 1 (line *1*) shows the Z domain transfer function. Code and difference equation form of this is found in appendix A.2.

$$H(z) \;=\; \frac{b_2\, z^{-2} \;+\; b_1\, z^{-1} \;+\; b_0}{a_2\, z^{-2} \;+\; a_1\, z^{-1} \;+\; 1} \tag{2}$$

Cascaded SOS implementation is a standard, efficient means of implementing an arbitrary filter [8][5]. Equation 2 shows the Z domain transfer function of a single, *standard form* SOS - signal flow is shown in figure 4. In *standard form* the gain is incorporated into the SOS structure. The BLRMS use an alternate SOS structure, see figure 3, with the overall filter gain factored out and applied separately. LIGO's implementation is less computationally intensive but many software tools won't return SOS in this format. Equations 3 yield the transformation between the LIGO form and *standard form*.

$$\beta_{\mathrm{i},1} \;=\; \frac{b_{\mathrm{i},1}}{b_{\mathrm{i},0}} \qquad \beta_{\mathrm{i},2} \;=\; \frac{b_{\mathrm{i},2}}{b_{\mathrm{i},0}} \qquad g \;=\; \prod_{\mathrm{i}\,=\,0}^{7} b_{\mathrm{i},0} \tag{3}$$

[5] page: *Direct Form II*

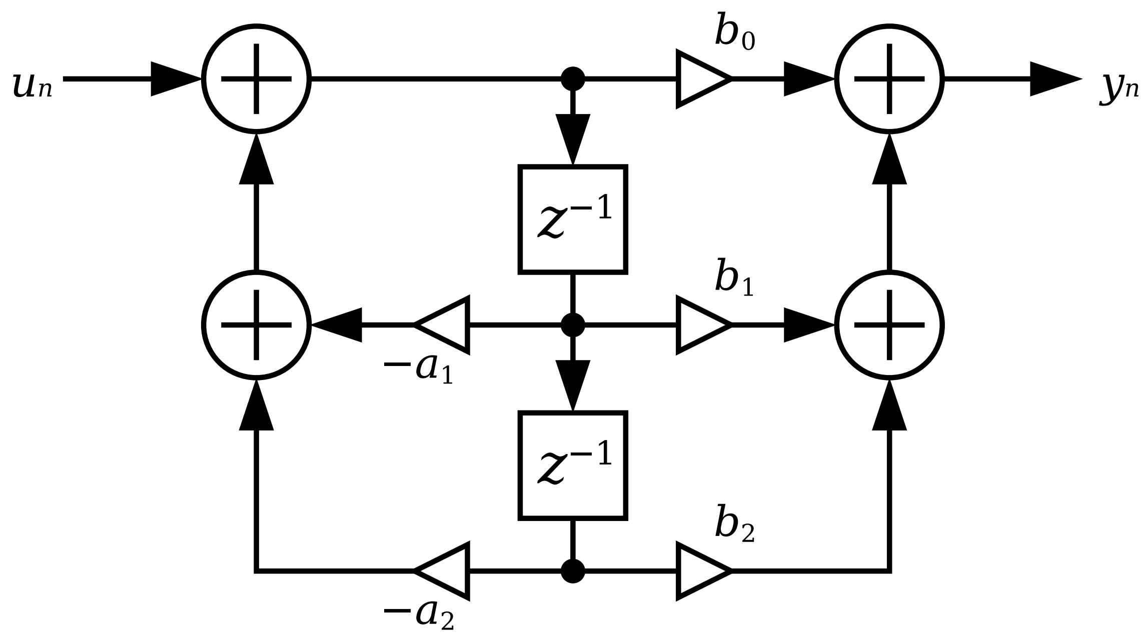

Figure 4: *Standard* implementation of a SOS. Note the introduction of a b_0 in comparison to figure 3, accounting for the SOS gain. Equation 2 gives the mathematical implementation. This representation, of a SOS filter, is common in the literature.

3 Updates to the Band-Limited Root-Mean-Square Code

Here we describe the updates to the BLRMS code to address the issues raised in section 1. This can be split into 4 parts; adding 2 new custom BLRMS for use with HAM and BSC CPS monitoring (sections 3.1 and 3.2), reducing the order of the DC BLRMS (section 3.3); and, changing the BL filter form to *transposed direct form* 2 (section 3.3).

3.1 New 65 Hz to 100 Hz Band-Limited Root-Mean-Square

As motivated in section 1, and preliminary testing, in [3], adding a new BLRMS filter will allow high frequency, CPS error sensing through guardian. The existing BLRMS *band options* are already full, necessitating adding a new *band option*. The code changes, achieving this are shown in listing 5.

The purpose of this *high frequency* BLRMS is to detect if the CPS signal matches known sensor noise - this sensitivity is reached above $\approx$ 30 Hz. There can be environmental noise up to $\approx$ 50 Hz, and the power lines sit at 60 Hz. For this reason we select 65 Hz as the lower cut off for the BL filter. To match with Jennifer's scripts we keep the upper cut off at 100 Hz.

To generate the SOS of the BL filter we have adapted the BLRMS design filter, MatLAB code [4][1][6]. The full, adapted code is shown in appendix B.2, listing 14. The magnitude response of this filter is shown in figure 5.

[6] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/sos_coeffs.m

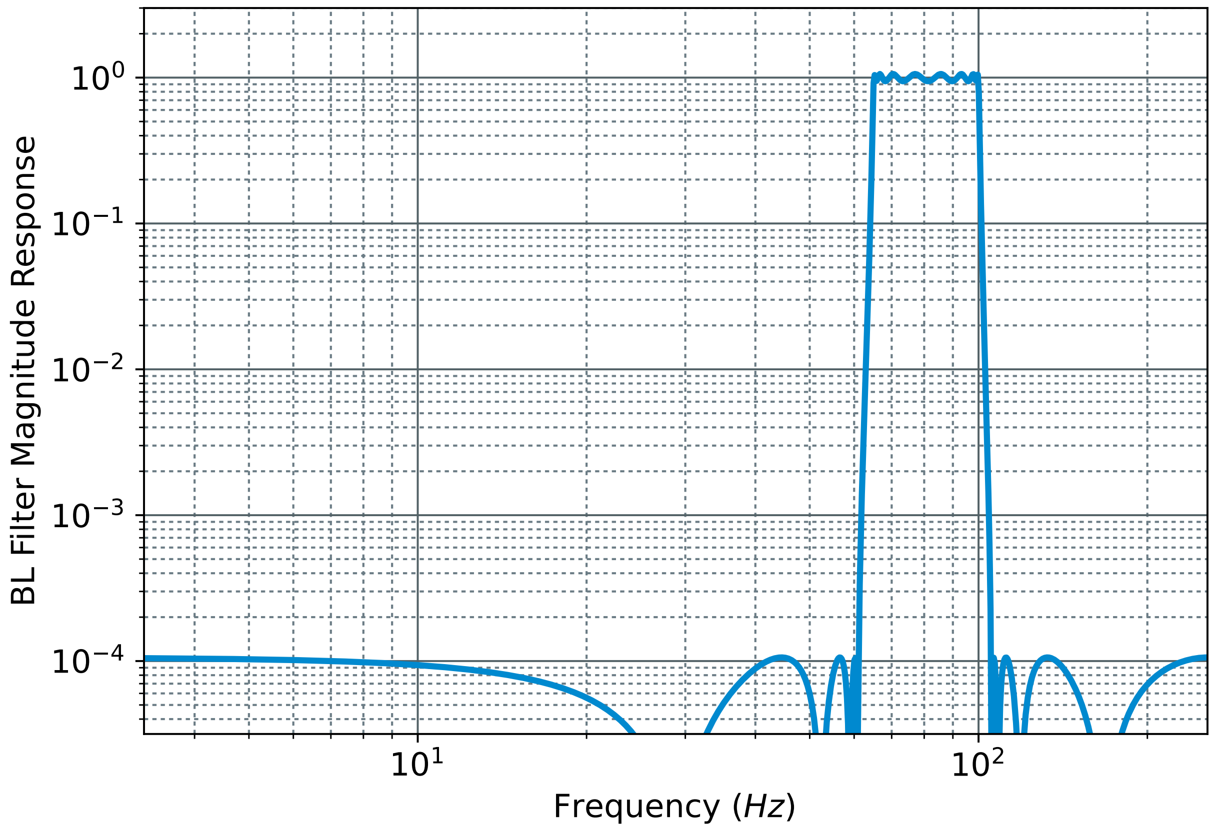

Figure 5: Bandpass filter of the new 65 Hz to 100 Hz BLRMS. Shown up to the Nyquist frequency of the BLRMS section, 256 Hz.

The additions, lines *35 - 42* and *45 - 55* in listing 14, provide error checking. Specifically ensuring that too many frequencies aren't requested ($\leq$ 9), and no frequencies are higher than the Nyquist frequency the BL filters operate at, respectively. Listing 14 is only setup to support BL filters, not Low Pass Filter (LPF)s and neither High Pass Filter (HPF)s.

Lines *58 - 62*, in listing 14, pre-warps the linear, *F domain* frequencies, while transforming to *S domain* angular frequencies. This implements equation 4, sourced from [5]. Such pre-warping is necessary when the Tustin transform is implemented, implicitly, at line *86* in listing 14. This ensures that the actual frequencies match the desired frequencies. In the previous BLRMS technical note [4] this pre-warping was manually performed, only for frequencies approaching 256 Hz.

$$\omega_{\text{warped}} \; = \; \frac{2}{T_{\text{sampling}}} \tan\left(\frac{T_{\text{sampling}}}{2} \, 2\,\pi\, f_{\text{F domain}}\right) \tag{4}$$

Zero padding is used; lines *67 - 70* and *99 - 101*, ensuring that unset frequencies have the BLRMS set to 0 - achieved by setting $g \; = \; 0$. The output printed to terminal can then, simply, be copied into the C code. This padding ensures the new BLRMS set has the appropriate size, 8.

Frequency Hz	**Pass/Stop Band**	**True RMS**	**BLRMS Result**	**Difference** dB
50	Stop	70.71	4.1×10^{-3}	-85
75	Pass	70.71	67.63	-0.387
115	Stop	70.71	4.5×10^{-3}	-84

Table 1: Sinusoidal tone test of the new 65 Hz to 100 Hz BLRMS. The steady state BLRMS result is compared with a true, time domain RMS. The passband difference between the two is given in dB, and is expected to be $\leq 0.5\ dB$. This difference originates from the ripple in the BL filter.

Simple tests have been performed to verify this new BLRMS is operational [10]. The results of these test are shown in table 1. A sinusoidal tone was input, with magnitude of 100, at 50 Hz, 75 Hz, 115 Hz, 160 Hz, and 215 Hz. These tone injections lasted for $\approx 40\ s$. To ensure that the steady state value is used $\approx 30\ s$ elapses before the BLRMS output it taken. The output value has been recorded and compared with the true RMS. The code in listing 17 accomplishes this task. A passband difference of $\pm\ 1\ dB$ is tolerated due to the ripple in the BL passband - only $\pm\ 0.5\ dB$ is expected. The suppression in the stop band is expected to be $\geq 79.5\ dB$.

Results in table 1 are as expected. The pass band difference is $< 1\ dB$, and the stop band rejection is $> 79.5\ dB$. The results at 160 Hz, and 215 Hz are consistent with this but aren't tabulated because they are far outside the passband.

3.2 New 130.5 Hz to 200 Hz Band-Limited Root-Mean-Square

To detect glitchy behaviour, coincident with high frequency noise on the CPS, a 100 Hz to 200 Hz BLRMS was requested; details in section 1. Unfortunately the 2$^{\text{nd}}$ harmonic of the power lines, seen in figure 1, occurs in this band at 120 Hz. To avoid placing this peak in band, 120 Hz needs to reside in the BL filter's stop band. As can be seen in figures 5 and 6 the stop band contains multiple notches. Additional suppression, beyond 80 dB, is achieved in these notches.

We take this opportunity to place 120 Hz in the notch immediately below the pass band. This permits the widest pass band, while maximising the power line rejection. Appendix B.3 contains the code necessary to achieve this.

The script, listing 15, fixes all parameters of the elliptic filter, except the lowest frequency of the pass band - see lines *73 - 79*. An optimiser adjusts this cut off to minimise the Band Pass (BP) filter magnitude at 120 Hz, lines *45 - 46*. All frequencies are prewarped, lines *81 - 86*, to ensure faithful translation from *S domain* to *Z domain*. The magnitude is evaluated in dB, line *95*, to establish a smooth cost space.

The resulting, optimised, BL filter is shown in figure 6. Our script determines that 130.4689 Hz is the optimal, lower cut off frequency. With this choice 177 dB of attenuation is achieved at 120 Hz; this is 97 dB of attenuation beyond the standard 80 dB.

We add 130.4689 Hz and 200 Hz to the custom BLRMS script; appendix B.2, listing 14, line

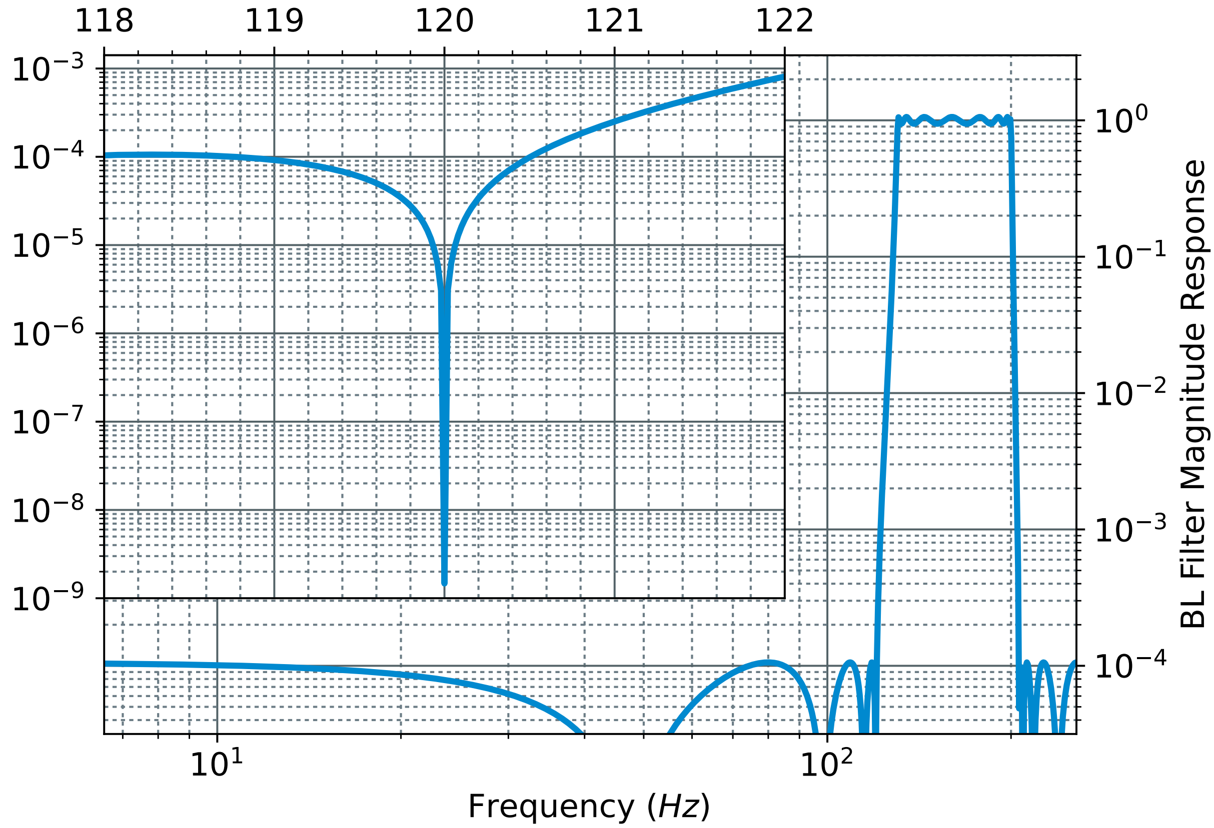

Figure 6: Bandpass filter of the new 130.5 Hz to 200 Hz BLRMS. Insert, upper left, zooms in around 120 Hz, revealing the notch targeted to 120 Hz. Shown up to the Nyquist frequency of the BLRMS section, 256 Hz.

30. The last, non-trivial BL filter, RMS, and coefficients when running listing 14 represent this new BLRMS. It has been added as the second, of 8, BLRMS in the new BLRMS set following the 65 Hz to 100 Hz from section 3.1.

We repeat the same simple, tests as used for the new 65 Hz to 100 Hz BLRMS, in section 3.1. Again a difference of 0.5 dB is expected in the pass band, with ± 1 dB tolerated. The stop band is expected to have a rejection of $\geq$ 79.5 dB. Results are detailed in table 2.

Table 2 shows the results from running listing 17. The frequencies in the stop band show the expected attenuation of $>$ 79.5 dB. The attenuation at 160 Hz is greater than the expected $\leq$ 0.5 dB, however within the tolerable limit of $<$ 1 dB. This minor discrepancy may warrant further investigation, which is beyond the scope of this technical note. As this is within the tolerable limit we certify that this new BLRMS is working. This discrepancy is potentially explained because only 1 data point has been inspected, see listing 17. Although no variation is expected if any is present it has been missed in this simple analysis.

The 3$^{\text{rd}}$ harmonic of the powerline, 180 Hz, is within this band. Spectra in the aLOGs,

Frequency Hz	**Pass/Stop/ Reject Band**	**True RMS**	**BLRMS Result**	**Difference** dB
115	Stop	70.71	2.0×10^{-4}	-111
160	Pass	70.71	63.60	-0.9212
215	Stop	70.71	3.5×10^{-3}	-86

Table 2: Sinusoidal tone test of the new 130.5 Hz to 200 Hz BLRMS. The steady state BLRMS result is compared with a true, time domain RMS. The passband difference between the two is given in dB, and is expected to be $\leq 0.5\ dB$. This difference originates from the ripple in the BL filter.

[11][7][8] and [12][9], do not show any statistically significant lines at 180 Hz. This is both with and without the presence of this additional noise, which we hope to detect. We ascertain this absence by visual inspection, and conclude that the powerline 3rd harmonic is not an issue for the new 130.5 Hz to 200 Hz BLRMS.

3.3 Fix the DC Band-Limited Root-Mean-Square

As outlined in section 1 the DC BLRMS *integrate forever*, making them unusable. We pursue 2 changes to resolve this signal processing issue. The first is reducing the LPF order, by half, reducing the number of non-trivial computations. The second is to alter the signal processing algorithm, to the best practise SOS implementation - increasing the numerical stability. Both changes were tested separately but we ultimately took the decision to merge them.

The BLRMS code has been designed to accept 8 SOS, all of which are required. As described in [4], and implemented in [1][10][11] all of these SOS are used. Compare line *18* in listing 3, with line *37* in listing 4. This shows that the roll off for the DC LPF is twice as steep as the roll-up/roll-down for the BL filters.

Aside from consuming all 8 SOS we see no reason for the DC LPF to have a steeper transition than the BL filters. Inspection of equation 1 shows that by choosing all of $a_1,\ a_2,\ \beta_1,\ \beta_2\ =\ 0$ then $H(z)$ becomes trivial, 1. Doing this for 4 SOS will reduce the DC LPF transition from 16th order to 8th order.

We implement the code to reduce the DC LPF order in Python, see listing 16. The principle advantage is the complicated conversion process; pre-warping, generation, and conversion to digital SOS is conducted in one step, lines *82 - 85*. We then convert this from *standard form* to the LIGO form in lines *87 - 93*. To obtain 8 SOS in the BL filter we pad with 4 trivial SOS, lines *95 - 101*.

Figure 7 compares the 8th and 16th order BL filters. The cut off at 30 mHz is altered

[7]attachment 1
[8]attachment 2
[9]attachment 2
[10]BSC-ISI/Stanford/s1isi_tools/design_BLRMS/documentation/sos_coeffs.m
[11]/BSC-ISI/Stanford/s1isi_tools/design_BLRMS/documentation/BLRMSFILTER.c

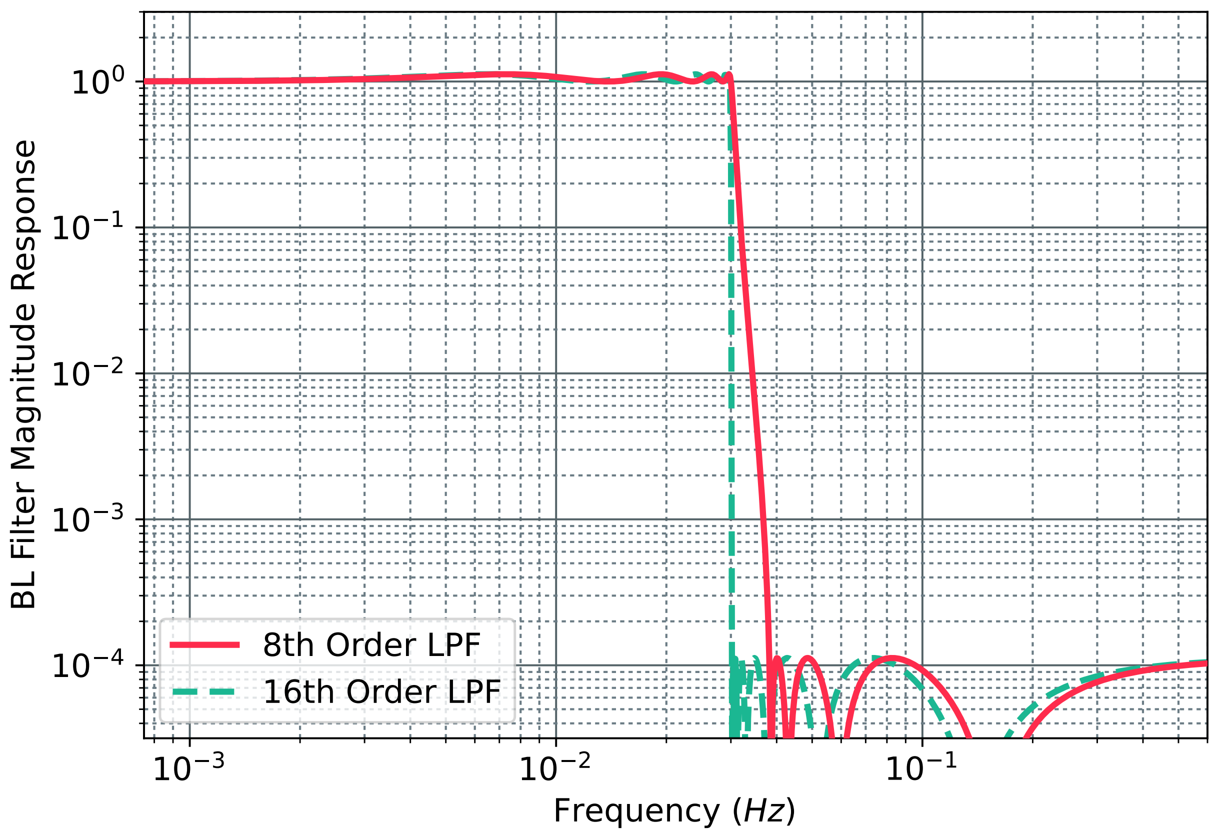

Figure 7: Comparison between 8$^{\text{th}}$ and 16$^{\text{th}}$ order LPFs. Both have 8 SOS, generated in Python, with the 8$^{\text{th}}$ order LPF having 4, trivial SOS.

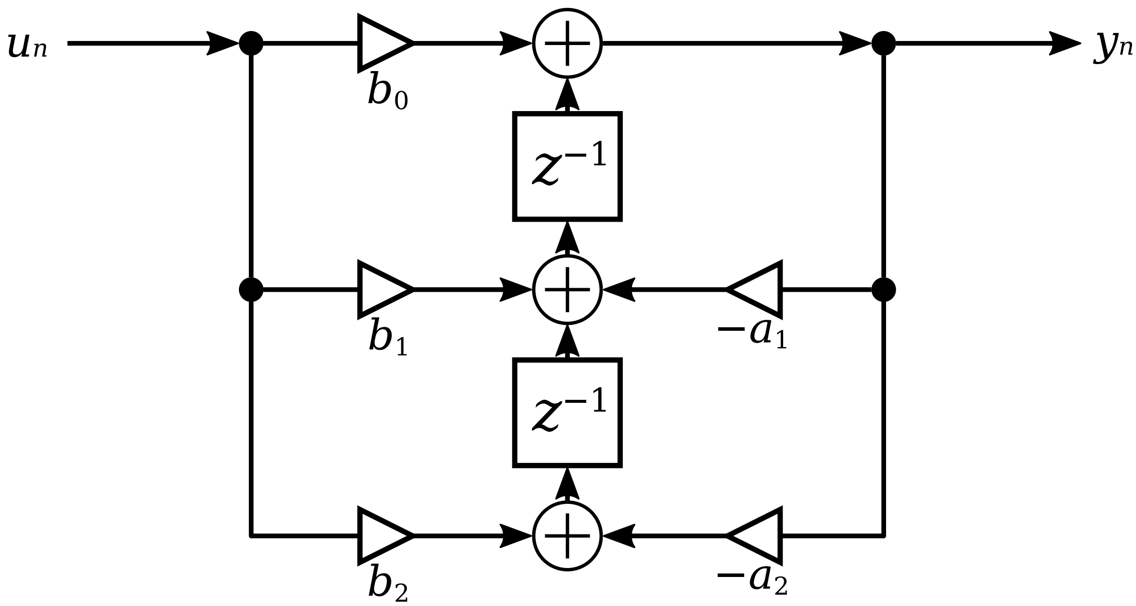

Figure 8: *Transposed direct form* 2 implementation of a SOS. Compared with figure 4 the zero and pole order is reversed. This is the recommended implementation of a SOS filter, commonly shown in the literature.

from nearly immediate to occurring between 30 mHz and 40 mHz. By 40 mHz maximum attenuation has been reached.

Lines *147 - 154* in listing 16 print the BL filter, SOS form to be added to the BLRMS code. This is also saved to a text file with lines *68*, and *157 - 171*. We discuss the performance of this order reduction after outlining the signal processing change.

Figure 3 is the current implementation of SOS filtering in the BLRMS. Here the poles, associated with a_1 and a_2, are implemented before the zeros, associated with β_1 and β_2. This is just one method of *direct form* 2 filtering.

Figure 8 shows the alternate method for *direct form* 2 filtering. This is called *transposed direct form* 2 [8][12]. In this transposed form the zeros are implemented before the poles. Factorisation of the *common gain* can, again, be done to get a format similar to the current BLRMS implementation. The signal flow for this *factorised* form is shown in figure 9.

The order of the pole/zero implementation is clearest when considering their effects in the frequency domain [8][13]. This especially true when the poles/zeros come in complex conjugate pairs, and have an associated Quality Factor (Q) - which is true for elliptic filters used in the BLRMS.

The frequency content, of the input signal, near high Q poles is amplified. This presents a problem when considering the finite precision, of summation, in the summing junctions: figure 3. This amplification can result in numerical instability, for example oscillations, when

[12]page: *Transposed Direct Forms*
[13]page: *More about Potential Internal Overflow of DF-II*

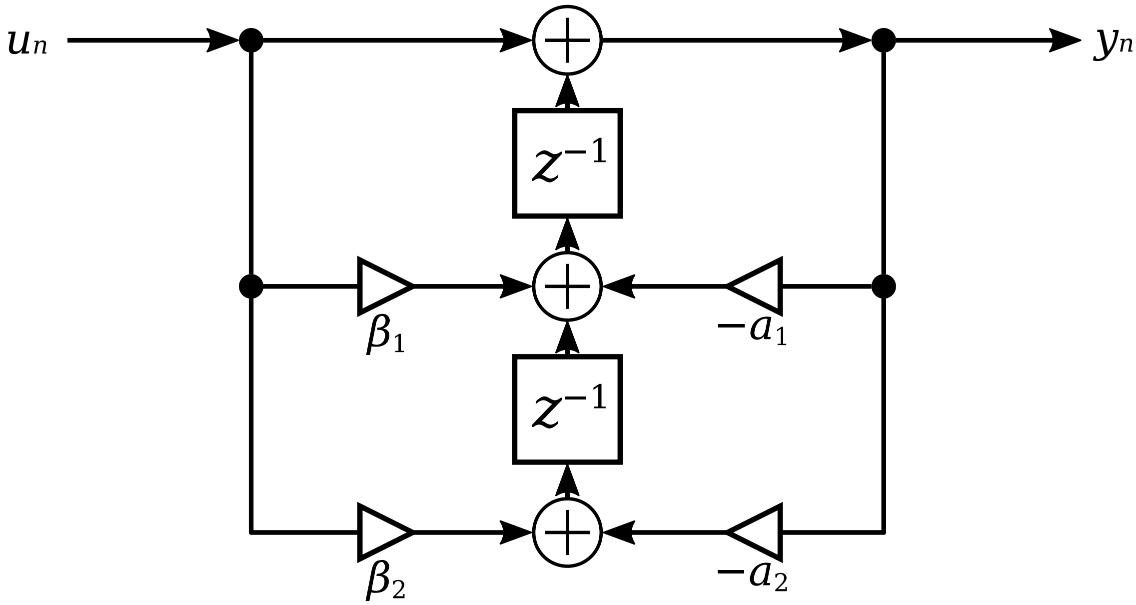

Figure 9: Transposed form of the SOS format used in the updated LIGO BLRMS. Compared to the previous implementation, figure 3, the zero/pole order is reversed.

the poles are implemented before the zeros.

Conversely the frequency content of an input signal is suppressed near high Q zeros. This still presents a problem for the finite precision summing junction: figure 9. As discussed in [2] this finite precision can result in limited suppression, when the zeros precede the poles. This limited suppression is not a problem for the BLRMS which are only rated for $\frac{1}{10^4}$, 80 dB, suppression.

Changing from *direct form* 2, to *transposed direct form* 2 exchanges the, potential, numerical instability problem for a limited suppression problem. For the suppression required in the BLRMS this is not an issue, as previously mentioned. To make this change the BLRMS code needs alteration. Appendix B.1 shows the new implementation of the BLRMS BL filter, specifically listing 13. Of particular import are lines *505 - 509*. Here:

- The new output is formed from the input and the previous history.
- The 1st internal state is updated using the current input, current output, and previous 2nd internal state.
- The 2nd internal state is updated using the current input, and current output.

Compared with lines *430 - 432* in appendix A.2, listing 2 which does the following:

- Form the current 1st internal state using the current input, previous 1st internal state, and previous 2nd internal state.

- Form the output from the current and previous 1st internal state, and previous 2nd internal state.

Figure 10 shows the response of the reduced (8^{th}) order, *direct form* 2; and, full (16^{th}) order, *transposed direct form* 2 BLRMS: parts **I** and **II**. Part **III** shows the current, production BLRMS. The output from the modified BLRMS is in agreement and convergent. The output from the production BLRMS is clearly beginning to diverge, visually at an exponential rate. This divergence began $>$ 1 week after a model restart, which resets the BLRMS.

A zoom in, near the start of figure 10, is shown in figure 11. Here the mean minute trends have been exchanged for the second trend data. Both modified BLRMS, parts **I** and **II**, are smooth and in agreement. While the trend in the production BLRMS, part **III**, agrees with the modified BLRMS there are 2 differences. The production BLRMS have a different offset, and more significantly have a clear oscillation. The spectra in figure 11, part **IV**, reveals that the oscillation is at 30 mHz and its harmonics. This frequency is exactly the DC BLRMS cut off, where poles of the elliptic LPF are located. Integrated, these oscillations; over a significant time period, can explain the divergent behaviour of the current, production BLRMS. The spectra of both modified BLRMS perfectly overlay.

Both modifications, reducing the LPF to 8^{th} order and changing to *transposed direct form* 2, individually resolve the lack of usability for the DC BLRMS. We decide to merge these 2 solutions by copying the reduced order SOS into the transposed form algorithm.

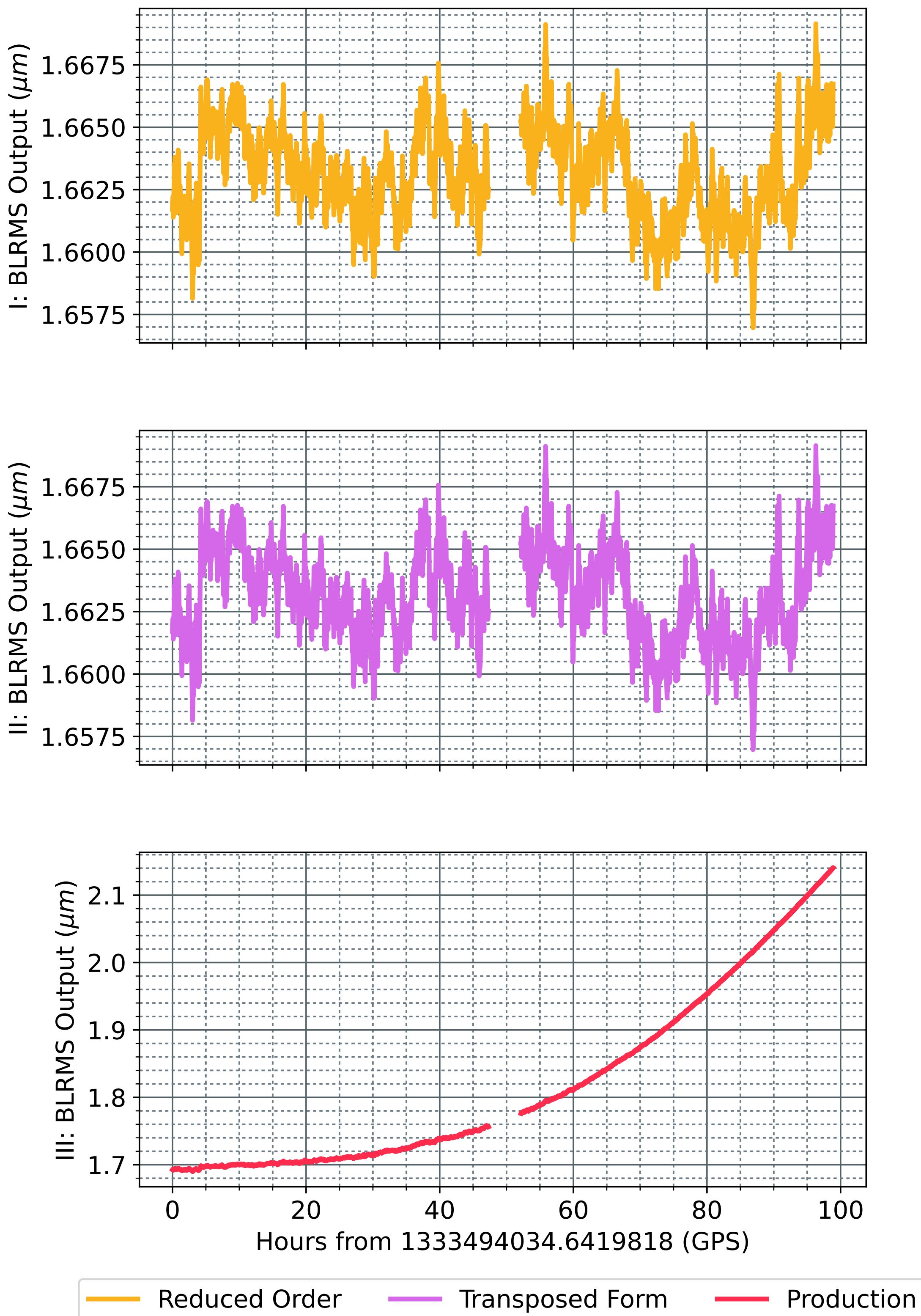

Figure 10: Mean minute trend of DC BLRMS output. Figure 11 is a zoom near the start of this data. Parts **I** and **II** shows the trial reduced order and transposed BLRMS outputs, and are well behaved. The current production BLRMS output is in **III**; and is demonstrating divergent, *integrate forever* behaviour. It took several weeks, after a model restart, for the outputs to diverge.

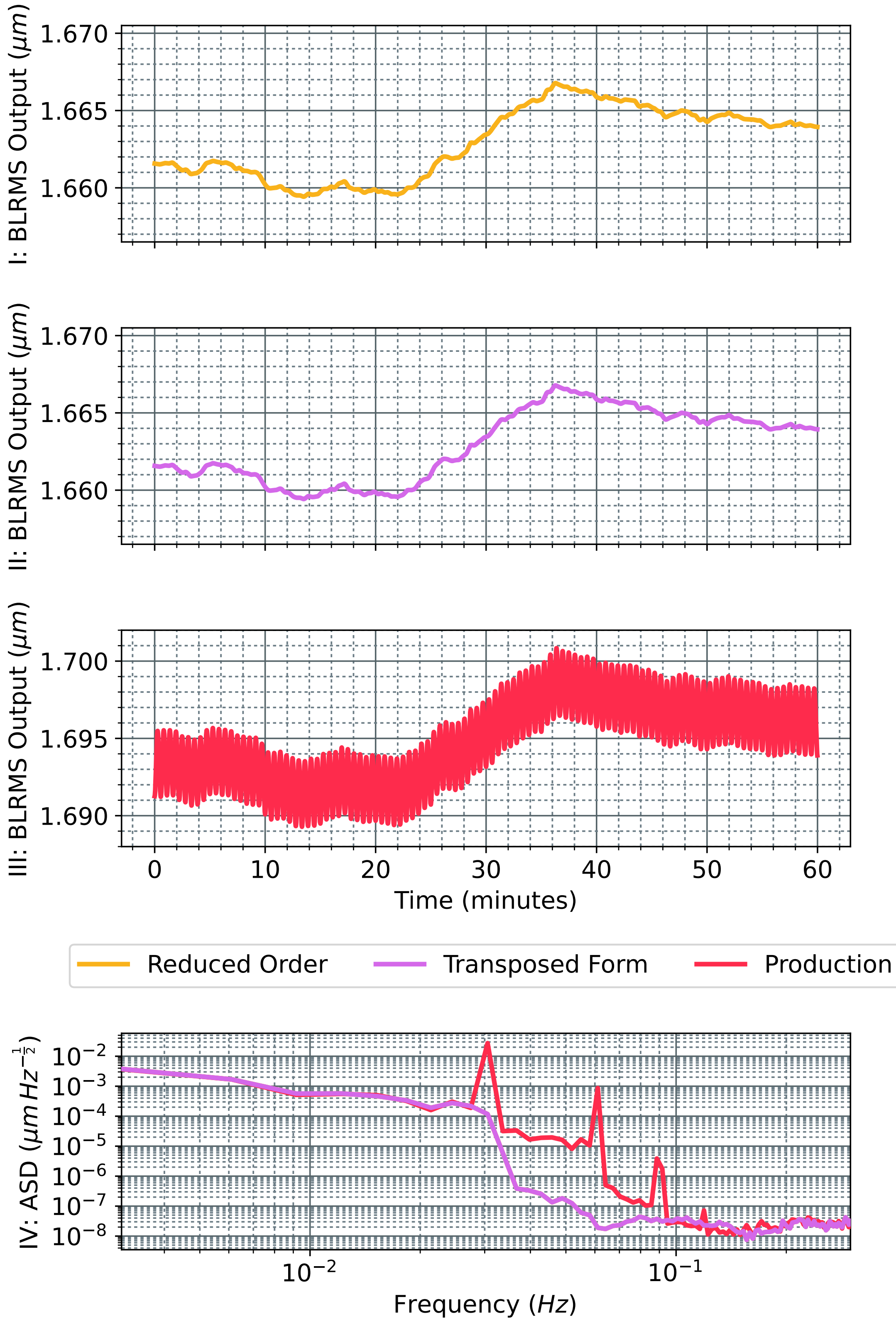

Figure 11: Second trend of DC BLRMS output. Parts **I** and **II** shows the trial reduced order and transposed BLRMS outputs. The current production BLRMS output is in **III**, which has a different value and developed an oscillation. The frequency of this oscillation is revealed as 30 mHz (and harmonics) in part **IV**, exactly the DC BLRMS cut off frequency.

4 New Production Band-Limited Root-Mean-Square

Appendix B.1 contains the code changes to the BLRMS C code. We have added the 60 Hz to 100 Hz (section 3.1) and 130.5 Hz to 200 Hz (section 3.2) to set 5, as BLRMS 0 and 1 respectively. To resolve the DC BLRMS issue we have implemented both the reduced order, and changed the SOS filter form to direct form 2 transposed (section 3.3).

The LPF order reduction has been implemented for all sets; see appendix B.1 listings 6, 7, 8, 9, and 11. We anticipate the changes to have a more significant effect for models sampled at a higher rate (16 384 Hz), sets 2 and 3.

One consideration, whose resolution is beyond the scope of this study, relevant to the new *high frequency* BLRMS is aliasing. Above 60 Hz the input signal is small, $\approx 5 \times 10^{-10}\ m\,Hz^{-\frac{1}{2}}$. The simple downsampling of the BLRMS, select every 8$^{\text{th}}$ point, means that higher frequency signals can be aliased down. This is an issue when there are any features, e.g. narrow resonances, in the top 7 frequency octiles of the input signal. Both new BLRMS; 65 Hz to 100 Hz (section 3.1), and 130 Hz to 200 Hz (section 3.2) are susceptible to this complication.

We suggest solving the aliasing issue by implementing a universal LPF; applied to the input signal before downsampling. Such a LPF could be applied at the full model rate, which varies. Precisely determining a cut off frequency is a trade off between the noise profile of the input signal, and the effect on the BLRMS result. For sufficiently featureless input signals the cut off frequency could be made high enough to have little effect on the BLRMS result. Implementation and precise placement of this LPF is beyond the scope of this document.

Another consideration, again beyond the scope of this study, is addressing the limited suppression of *transposed direct form* 2, see section 3.3. This may limit the power line, harmonic suppression, 177 dB from section 3.2, that can be achieved - no tests have been made. The solution proposed [2], and implemented for the Control and Data acquisition System (CDS) filters is changing to a state space filter form. However a careful study of [5] reveals that state space filter forms also suffer from numerical issues as high Q zeros/poles approach 0 Hz, which occurs in the BLRMS. The open nature of this problem leaves it beyond the scope of this document.

The updated BLRMS code is available on the SEI SVN [1][14]. It is ready to be implemented into the production CDS code.

[14] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/new_production/BLRMSFILTER.c

References

[1] LIGO Scientific Collaboration. Ligo seismic svn. svn checkout -r9645 https://svn.ligo.caltech.edu/svn/seismic/.

[2] Matthew Evans. Digital filter noise. Technical Report G09000928-v1, LIGO Scientific Collaboration, 2009.

[3] Nathan A. Holland. Cps noise detection - guardian data processing. SEI aLOG 1830.

[4] Chris Kucharczyk and Brian T. Lantz. Real-time band-limited rms monitor. Technical Report T1100613-v3, LIGO Scientific Collaboration, 2012.

[5] Diego Passuello. Digital filters. Technical Report VIR-0639A-20, VIRGO Scientific Collaboration, 2020.

[6] Jameson Graef Rollins. Distributed state machine supervision for long-baseline gravitational-wave detectors. *Review of Scientific Instruments*, 87(9):094502, September 2016.

[7] Anthony Sanchez. H1 isi cps sensor noise spectra check - weekly. LHO aLOG 63461.

[8] Julius O. Smith. *Introduction to Digital Filters*. `https://ccrma.stanford.edu/~jos/filters/`, 2007.

[9] Patrick Thomas. H1 isi cps noise spectra check - weekly. LHO aLOG 60867.

[10] Jim Warner. Comment related to report: Both versions of 0-30mhz blrms seem better than production. SEI aLOG 1918.

[11] Jim Warner. Comment related to report: Sr3 hopping associated with ham5 isi glitches. LHO aLOG 62142.

[12] Jim Warner. Ham5 cps hops maybe caused by bad sync/power board. LHO aLOG 62164.

[13] Jim Warner. "ham8" testing cps blrms code. SEI aLOG 1867.

[14] Jim Warner and Brian T. Lantz. weekly SEI collaboration meeting on 2021-12-17T19:30+01. discussing next steps to progress CPS error detetction.

[15] Jennifer Watchi. Cps noise detection script. SEI aLOG 1823.

Acronyms

aLIGO The Advanced LIGO Detector, housed in the LIGO Observatories. 29

BL Band Limit. 1, 4–10, 12, 13, 20–26, 29

BLRMS Band-Limited Root-Mean-Square. 1–10, 12–17, 20, 23, 26, 27, 29, 31, 35, 38

BP Band Pass. 8, 29

BSC Basic Symmetric Chamber: large vacuum chamber enlosing the suspended core optics. 2, 6

CDS Control and Data acquisition System. 17

CPS Capacitive Position Sensor. 2, 3, 6, 8

DC DC (0 Hz). 1, 3, 6, 10, 14–17, 23–26, 31, 35

HAM Horizontal Access Module: small vacuum chamber encasing auxiliary optics mounted on tables. 2, 6

HPF High Pass Filter. 7

ISC Interferometer Sensing and Control. 3

ISI Internal Seismic Isolation: in-vacuum isolation. 2

ITM Input Test Mass. 2

LHO LIGO Hanford Observatory. 2, 18

LIGO Laser Interferometer Gravitational-wave Observatory. 5, 10, 13

LPF Low Pass Filter. 7, 10, 11, 14, 17, 21, 23–25

Q Quality Factor. 12, 13, 17

RMS Root Mean Square. 4, 8–10, 26

SEI Seismic isolation. 2, 17, 18, 23, 35

SOS Second Order Sections. 1, 4–6, 10–14, 17, 20–26, 31, 35

SUS Suspension. 3

SVN Apache Subversion Control Software. 2, 17, 23

U.S. United States of America. 29

A Key Snippets of Existing BLRMS Code

Here we exhibit key snippets of the existing/old BLRMS code to illustrate its operation, or limitations.

A.1 Number of Bands and Sections

The number of SOS, and BL filters are hard coded into the BLRMS C code - both are set to 8. This choice imposes limitations on how the BLRMS code can be modified, without major rewrites. Interpretation of listing 1 shows; there must be 8 SOS per BL filter, and there must be 8 BL filters per BLRMS option.

```
    #define NUM_BANDS 8
    #define NUM_SOS 8 // number of SOS (defined by order * 2)
    #define NUM_COEFFS 4 // number of coefficients (since b0 = a0 = 1, can just use [b1 b2 a1 a2]
    #define NUM_BAND_OPTIONS 5 // number of different sets of bands, including default zeroes bands
```

Listing 1: Hard coded BLRMS values.

A.2 Implementation of Band Limit Filter

The BL filter is implemented in direct form 2, cascaded SOS, as illustrated in figure 3. Reading the comments on lines 420, 424, 430, 434, and 438 in listing 2 illuminates the purpose of the code, which immediately follows.

```
415 // Declare the variables we'll need below to speed up allocation
416 double new_input = cur_avg, new_output = 0;
417 double hist1, hist2, new_w, a1, a2, b1, b2;
418 double *hist1_ptr = &w_hist[band][0][0], *hist2_ptr = &w_hist[band][0][1], *coeff_ptr = &sos_coeffs[set][band][0][0];
419 for(ii = 0; ii < NUM_SOS; ii++){
420 // Get the previous history values
421 hist1 = *hist1_ptr;
422 hist2 = *hist2_ptr;
423
424 // Get the coefficients from the coefficient matrix
425 b1 = *coeff_ptr++;
426 b2 = *coeff_ptr++;
427 a1 = *coeff_ptr++;
428 a2 = *coeff_ptr++;
429
430 // Calculate the new w, and then the new output
431 new_w = new_input - a1 * hist1 - a2 * hist2;
432 new_output = new_w + b1 * hist1 + b2 * hist2;
433
434 // Shift the histories and increment the history pointers
435 *hist2_ptr++ = hist1; ++hist2_ptr;
436 *hist1_ptr++ = new_w; ++hist1_ptr;
437
438 // Push the output down the line of sections
```

```
new_input = new_output;
}
```

Listing 2: Loop implementing BL filter, as a direct form 2, cascaded SOS.

Listing 2 can be compared with the difference equations 5 for direct form 2, derived from figure 3. Inspection reveals that they are direct translations of each other.

$$\begin{aligned} w_n &= u_n - a_1 w_{n-1} - a_2 w_{n-2} \\ y_n &= w_n + \beta_1 w_{n-1} + \beta_2 w_{n-2} \end{aligned} \tag{5}$$

A.3 Assignment of SOS Coefficients

The following code snippets are used in the script which defines the existing SOS coefficients. The first, listing 3, is used for the LPF, and the second, listing 4, is used for the BL filters. Both are taken from [1][15]

```
%% Lowpass filter coefficients

[num den] = ellip(order*2, ripple, atten, 2*pi*corner,'low','s');
ellip_tf_c = 10^(1/20) * tf(num,den); % normalize to 1
ellip_zpk_d = zpk(c2d(ss(ellip_tf_c),Ts, 'tustin'));

%figure
%bode(ellip_tf_c, ellip_zpk_d);
%step(ellip_tf_c, ellip_zpk_d);

[zz, pp, kk] = zpkdata(ellip_zpk_d);

[low_sos, low_g] = zp2sos(zz{:}, pp{:}, kk);
full_sos{1}.sos = low_sos;
full_sos{1}.gain = low_g;
```

Listing 3: Extract showing the LPF SOS generation.

```
%% Bandpass filter coefficients

for x = 1:length(frequencies) - 1
lower = frequencies(x);
upper = frequencies(x+1);
[num den] = ellip(order, ripple, atten, [2*pi*lower,
    2*pi*upper],'bandpass','s');
ellip_tf_c = tf(num,den);
ellip_zpk_d = zpk(c2d(ss(ellip_tf_c),Ts, 'tustin'));

%figure
%bode(ellip_tf_c, ellip_zpk_d);
%step(ellip_tf_c, ellip_zpk_d);

[zz, pp, kk] = zpkdata(ellip_zpk_d);
```

[15] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/documentation/sos_coeffs.m

```

[sos, g] = zp2sos(zz{:}, pp{:}, kk);
full_sos{x+1}.sos = sos;
full_sos{x+1}.gain = 1.0591 * g;
end
```

Listing 4: Extract showing the BL SOS generation.

B Source Code for Updated Production BLRMS

This appendix contains code from the new BLRMS, code used to investigate different options for the updated BLRMS, and code used to generate the new BLRMS code.

B.1 Key Excerpts from New BLRMS Code

Key excerpts from the new BLRMS C code. This code is available at [1][16]. Only changes and additions are shown, otherwise the BLRMS code remains unaltered.

```
13 #define NUM_BANDS 8
14 #define NUM_SOS 8 // number of SOS (defined by order * 2)
15 #define NUM_COEFFS 4 // number of coefficients (since b0 = a0 = 1, can just use [b1 b2 a1 a2]
16 #define NUM_BAND_OPTIONS 6 // number of different sets of bands, including default zeroes bands
```

Listing 5: Portion of the new BLRMS C code, adding a new BLRMS bank, line *16*.

```
//SET 1
{
{{-1.999996253790484, +1.000000000000000, -1.999841253785180, +0.999841268671519},
{-1.999999472654055, +1.000000000000000, -1.999891439401114, +0.999891501986762},
{-1.999999720370403, +1.000000000000000, -1.999945718565804, +0.999945829225718},
{-1.999999775142958, +1.000000000000000, -1.999984289359235, +0.999984424520308},
{+0.000000000000000, +0.000000000000000, +0.000000000000000, +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000, +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000, +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000, +0.000000000000000}},
```

Listing 6: SOS of the 1st BL filter in set 1, the DC LPF. Only 4 of the SOS are non-trivial, see section 3.3 for details. These have the ordering $\beta_1,\ \beta_2,\ a_1,\ a_2$.

The reduction in the DC LPF order for sets 2 through 4, inclusive, is performed in an identical fashion to the reduction for set 1, see section 3.3. Appendix B.4, listing 16 gives the prototype for the code to achieve this, and because of this similarity we do not expand upon the method to update the LPF SOS.

Changes to the BLRMS code, reducing the DC BLRMS order, are shown in listings 7, 8, 9, and 11. For interested readers the code which generates these SOS and g is available on the SEI SVN [1][17].

[16] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/new_production/BLRMSFILTER.c

[17] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/DC_reduce_order/lower_dc_order__sets_2to4.py

```
//SET 2
{
{{-1.999999765861705, +1.000000000000000, -1.999960313875479,
    +0.999960314805930},
{-1.999999967040875, +1.000000000000000, -1.999972870481165,
    +0.999972874392927},
{-1.999999982523149, +1.000000000000000, -1.999986450114725,
    +0.999986457031110},
{-1.999999985946435, +1.000000000000000, -1.999996097659635,
    +0.999996106107251},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000}},
```

Listing 7: SOS of the 1st BL filter in set 2, the DC LPF.

```
//SET 3
{
{{-1.999999765861705, +1.000000000000000, -1.999960313875479,
    +0.999960314805930},
{-1.999999967040875, +1.000000000000000, -1.999972870481165,
    +0.999972874392927},
{-1.999999982523149, +1.000000000000000, -1.999986450114725,
    +0.999986457031110},
{-1.999999985946435, +1.000000000000000, -1.999996097659635,
    +0.999996106107251},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000}},
```

Listing 8: SOS of the 1st BL filter in set 3, the DC LPF.

```
// SET 4
{
{{-1.999985015203025, +1.000000000000000, -1.999682502988829,
    +0.999682562529462},
{-1.999997890616909, +1.000000000000000, -1.999782765418519,
    +0.999783015747538},
{-1.999998881481772, +1.000000000000000, -1.999891218763489,
    +0.999891661391151},
{-1.999999100571924, +1.000000000000000, -1.999968308645248,
    +0.999968849285312},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
```

```
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000},
{+0.000000000000000, +0.000000000000000, +0.000000000000000,
    +0.000000000000000}},
```

Listing 9: SOS of the 1st BL filter in set 4, the DC LPF.

```
374 // SET 5.
375 {
376 {{+0.939904055858490, +1.000000000000003, -0.943142921463701,
        +0.907406742982836},
377 {-1.876855971549091, +0.999999999999999, -1.122798637250702,
        +0.913466548484862},
378 {-0.229729632244250, +0.999999999999838, -0.793139968232068,
        +0.933521844315428},
379 {-1.600516301950299, +0.999999999999938, -1.267848900042316,
        +0.944422158492960},
380 {-0.475923704748913, +1.000000000000643, -0.705508111128152,
        +0.965874326607312},
381 {-1.499243708783478, +1.000000000000305, -1.352355813288312,
        +0.973274900023501},
382 {-0.543802035583123, +0.999999999999518, -0.671133348662952,
        +0.990040940574222},
383 {-1.466562439300325, +0.999999999999758, -1.391046678718050,
        +0.992417256904423}},
384 {{-1.667974710038036, +1.000000000000000, +0.685608657586911,
        +0.813508707073553},
385 {+1.956841363253865, +1.000000000000006, +1.060971054689685,
        +0.833851286285895},
386 {-0.707868928375943, +1.000000000000000, +0.350445889919364,
        +0.867254696207462},
387 {+1.782512855641971, +0.999999999999983, +1.332042138935376,
        +0.901759881881546},
388 {-0.319729758187417, +0.999999999999996, +0.147351929111666,
        +0.933256342152839},
389 {+1.678904306649074, +1.000000000000028, +1.474063748250583,
        +0.955508315359327},
390 {-0.196034513566946, +1.000000000000002, +0.061429785294229,
        +0.980682675155925},
391 {+1.639726694624601, +0.999999999999989, +1.535926533237836,
        +0.987689082261325}},
```

Listing 10: SOS of the new BL filters. See section 3.1 for details of lines *376 - 383*. See section section 3.2 for details on lines *384 - 391*.

```
// The overall gain for each band
static double g[NUM_BAND_OPTIONS][NUM_BANDS] =
{
{0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
{0.00011218306675448346, 0.000105869013273, 0.000105796356279,
    0.000105558685102, 0.000105248093676, 0.000108075052843,
    0.000145444643508, 0.00111268350408}, // Set 1
{0.0001121971269324529, 0.000105899630900, 0.000105880590539,
    0.000105809979638, 0.000105645813872, 0.000105273600293,
    0.000106242311402, 0.00014984827880}, // Set 2
```

```
{0.000112197126932452 9, 0.000105899630900, 0.000105880590539,
    0.000105809979638, 0.000105645813872, 0.000105273600293,
    0.000112773780492, 0.00296961827569}, // Set 3
{0.000112164542178747 2, 0.000105829331565, 0.000105693343841,
    0.000105336599667, 0.000105622097453, 0.000122409832774,
    0.000304523250745, 0.10925941222306}, // Set 4
{0.000254775749195, 0.001082577254430, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0} // Set 5
};
```

Listing 11: Altered gains, g, for the changed DC and new BL filters. Unused BLRMS have $g = 0$.

```
{0.001949317738791, 0.001949317738791, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0} // Set 5
```

Listing 12: Additional RMS coefficients for the new BLRMS.

```
// Declare the variables we'll need below to speed up allocation
double new_input = cur_avg, new_output = 0;
double hist1, hist2, a1, a2, b1, b2;
double *hist1_ptr = &w_hist[band][0][0], *hist2_ptr = &w_hist[band][0][1],
    *coeff_ptr = &sos_coeffs[set][band][0][0];
for(ii = 0; ii < NUM_SOS; ii++){
// Get the previous history values
hist1 = *hist1_ptr;
hist2 = *hist2_ptr;

// Get the coefficients from the coefficient matrix
b1 = *coeff_ptr++;
b2 = *coeff_ptr++;
a1 = *coeff_ptr++;
a2 = *coeff_ptr++;

// Calculate the new output, b0 = 1
new_output = new_input + hist1;

//Calculate the histories
hist1 = b1 * new_input - a1 * new_output + hist2;
hist2 = b2 * new_input - a2 * new_output;

// Shift the histories and increment the history pointers
*hist2_ptr++ = hist2; ++hist2_ptr;
*hist1_ptr++ = hist1; ++hist1_ptr;

// Push the output down the line of sections
new_input = new_output;
}
```

Listing 13: For loop implementing the BL filter as a direct form 2, transposed SOS filter. See section 3.3 for details.

B.2 Generation of Custom BLRMS Band Options

The following MatLAB code is used to generate a full, custom *band option* for the BLRMS C code. The code is also available at [1][18].

```
%% SOS Coefficients for real-time BLRMS monitor
% updated by BTL on Nov 9, 2011 to simplify the conversion to discrete time
% Available from:
% https://svn.ligo.caltech.edu/svn/seismic/BSC-ISI/Stanford/s1isi_tools/design_BLRMS/documentation/sos_coeffs.m
% clear all, close all;


%{
Modified by N.A. Holland on 2022-01-07.
Contact: nholland@nikhef.nl

Modifications: Added the source url, for my own reference, and added frequency
prewarping, which is important for the new high frequency BLRMS.


Modified by N.A. Holland on 2022-01-21
Contact: nholland@nikhef.nl

Modifications: Cleaned the code for committing to the SEI SVN.
%}


%% General settings
format long;
mdl_rate = 4096; % The ISI model rate, or another if required.
Ts = 8/mdl_rate; % sample time. 1/8 th of the models running rate.
order = 8; % elliptical filter order
ripple = 1; % 1db of ripple
atten = 80; % 80db of attenuation
frequencies = [65, 100, 130.4688823820248, 200]; % frequency band limits


%% Warnings

% Too many frequencies.
if length(frequencies) > 9
err_msg = ['The BLRMS C code is setup to accept 8 BLRMS. ' ...
'You have requested ' num2str(length(frequencies) - 1) ...
' BLRMS - which is too many. Instead split these over ' ...
'several BLRMS, integer options.'];
error(err_msg)
end


% Frequencies are too large.
f_nq = 0.5/Ts;
mask = frequencies >= (f_nq);

```

[18] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/CPS_65Hz_to_100Hz/sos_coeffs_cust.m

```
if any(mask)
err_msg = ['Digital filters can only operate up to (<) the Nyquist' ...
' frequency of ' num2str(f_nq) ' Hz. You have requested' ...
' the following invalid frequencies: ' ...
sprintf('%.1f Hz, ', frequencies(mask))];
error(err_msg(1:end-2))
end


%% Prewarp the frequencies.
% This can handle anything which f -> 256 Hz (from below)

warped_w = (2/Ts) .* tan(2*pi*Ts/2 .* frequencies);
warped_f = warped_w ./ (2*pi);


%% Bandpass filter coefficients

% Preallocate space.
% Ensures that there is are full SOS elements for the C code.
full_sos = cell(1,8);
full_sos(1:8) = {struct('sos', zeros(8,6), 'gain', 0)};


% Create each SOS
for x = 1:length(warped_w) - 1
% Frequencies of the bandpass.
lower = warped_w(x);
upper = warped_w(x + 1);

% b, a filter form
[num, den] = ellip(order, ripple, atten, [lower, upper], ...
'bandpass', 's');
ellip_tf_c = tf(num,den);

% Convert to digital domain via state space form, to preserve numerical
% accuracy.
ellip_zpk_d = zpk(c2d(ss(ellip_tf_c),Ts, 'tustin'));

%figure
%bode(ellip_tf_c, ellip_zpk_d);
%step(ellip_tf_c, ellip_zpk_d);

[zz, pp, kk] = zpkdata(ellip_zpk_d);

[sos, g] = zp2sos(zz{:}, pp{:}, kk);
full_sos{x}.sos = sos;
full_sos{x}.gain = 1.0591 * g; % Gain is adjusted slightly.
end

%% RMS coefficients
alpha = zeros(1,8); % actual coefficients
tau = zeros(1,8); % decay time constants

% Fill values
for x = 1:length(warped_f)-1
```

```
tau(x) = max(1, 8 / sqrt( warped_f(x) * warped_f(x + 1)));
alpha(x) = Ts / (Ts + tau(x));
end

%% write to a text file that we can just copy into C
coefficients = '';
gains  = '{';
alphas = '{';
taus   = '{';

for k=1:8
for j=1:order
coefficients = [coefficients; ...
sprintf(' {%+.15f, %+.15f, %+.15f, %+.15f},  ', ...
full_sos{k}.sos(j,2:3), full_sos{k}.sos(j,5:6))];
end
coefficients(order*(k-1) + 1, 1) = '{';
coefficients(order*k, end-2:end-1) = '},';
gains = strcat(gains, sprintf(' %.15f,', full_sos{k}.gain));
alphas = strcat(alphas, sprintf(' %.15f,', alpha(k)));
taus   = strcat(taus, sprintf(' %.15f,', tau(k)));
end
coefficients(end,end-1) = ' ';
gains(end, end-1:end) = '};';
alphas(end, end-1:end) = '};';


% Muted because this isn't needed in the C code.
%taus

% Printed to terminal because these are used.
coefficients
gains
alphas
```

Listing 14: MatLAB script to generate a custom, elliptical BP filter *band option*. Any unset BL filters become 1.

B.3 Placing Power Line Harmonics in Band Limit Filter Elliptic Notch

aLIGO is based in the United States of America (U.S.) where the mains power frequency is 60 Hz. This leads to large, environmental noise peaks at 60 Hz, and its natural number multiples, coupling in through electronics. It is preferable to avoid noise peaks, such as these, in the BLRMS.

It is relatively straight forward to manually tune elliptical filters to avoid a specific frequency. However it is tedious, bordering on impractical, to place a specific frequency in a bandstop notch of an elliptic filter manually. We wrote the script in listing 15 to place the 120 Hz powerline 2nd harmonic in the first, lower bandstop notch of the 130 Hz to 200 Hz BLRMS - see section 3.2.

```
%% A Script to Generate Jim Warner's High Frequency BLRMS
%{
```

```
Jim has requested a ~ 100 Hz to ~ 200 Hz BLRMS, see SEI aLOG 1867
(https://alog.ligo-la.caltech.edu/SEI/index.php?callRep=1867). This is
motivated by LHO aLOG 62142, when the CPS was playing up some more
(https://alog.ligo-wa.caltech.edu/aLOG/index.php?callRep=62142).At the SEI
call on 2022-03-11 it was decided to place this in excess of 120 Hz, to
avoid the power line 2nd harmonic.

This script optimises the location of the 1st elliptic notch to 'dump' the
120 Hz AC powerline harmonic. This is an idea Brian T. Lantz had, with
respect to, the 65 Hz filter, in late 2021. I'm actioning it now, and maybe
I will retrospectively apply it to the 65 Hz to 100 Hz BLRMS filter.


Author: Nathan A. Holland
Contact: nholland@nikhef.nl
Date: 2022-03-18

Version: 0.80

Changelog:
2022-03-25 Fixed bug in code, evalfr needs 's' not 'w' and got
130.4689 Hz as the lower frequency.
2022-03-18 Created, based upon existing work, and command line
experimentation.
%}

clear all


%% Main

% Optimisation options.
min_opts = struct();
min_opts.Display = 'notify';


% Bounds - empiricially determined.
f_min = 125;
f_max = 135;


% Minimisation.
[f_opt, min_mag, xit_flag, opt_out] = fminbnd(@filter_mag_120, ...
f_min, f_max, min_opts);


% Adjust some parameters.
suppr_add = -1*(min_mag + 80);


% Printing.
['Lower BL frequency of ' num2str(f_opt, '%.4f') ' Hz ' ...
'achieves additional suppression of ' num2str(suppr_add, '%.1f') ' dB.']


%% Function to optimise the frequency.
```

```
function [mag] = filter_mag_120(f_low)
%{
A function returning the magnitude of the BL filter, as a function of
the lowest band pass frequency. It is targetted to minimising this at
120 Hz, the 2nd Harmonic of the US AC power line.

Usage:
[mag] = filter_mag_120(f_low)

mag   - The magnitude of the BL filter at 120 Hz.

f_low - The lower frequency of the band limit, around 130.5 Hz.
%}

% Constant values.
order = 8;
rpl_dB = 1;
Ts = 8/4096;
atten_dB = 80;
f_hi = 200;
f_out = 120;

% Frequency warping.
warp = @(f) (2/Ts) * tan(2*pi*Ts/2 * f);

w_lo = warp(f_low);
w_hi = warp(f_hi);
w_out = warp(f_out);

% Define the analogue filter.
[A, B, C, D] = ellip(order, rpl_dB, atten_dB, [w_lo w_hi], ...
'bandpass', 's');
filt = ss(A, B, C, D);

% Get the response.
H = evalfr(filt, j*w_out);
mag = 20*log10(abs(H));
end
```

Listing 15: MatLAB script to notch 120 Hz as part of an elliptic filter. Chosen to remove the 2nd harmonic of the mains powerline.

B.4 Reducing the DC BLRMS Order

Python code, available at [1][19], to lower the order of the DC BLRMS, for BLRMS set 1 (models sampled at 4096 Hz). The SOS this generates are given in listing 6.

```
"""
If at first you don't succeed try again. Why am I more hopefull in Python?
> Because Python has tools to DIRECTLY generate the SOS in the z domain (which
I know how to use).
I'm slightly surprised it works but it validates my suspicion that you can
```

[19] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/DC_reduce_order/investigate_lower_dc_order.py

```
reduce the BL order. This is CLOSE to the numerical precision limits, internal
to the filters.


Author: Nathan A. Holland
Contact: nholland@nikhef.nl
Date: 2022-01-28

Version: 1.01
Changelog:
2022-08-02 Altered the gain adjustment to 1 dB, from 0.5 dB.
2022-01-28 Created, copying across from MatLAB script.
"""


#------------------------------------------------------------------------------

# Imports.
import numpy as np

import scipy.signal as sig

import matplotlib.pyplot as plt


#------------------------------------------------------------------------------

# Script variables.

# Version.
_version_ = 1.02
_vers_str_ = f'{_version_:.2f}'


# Signal processing details.

# Sampling time
Ts = 1/512
# LPF filter order
order = 8
# Ripple in dB.
ripple = 1
# Attenuation in dB.
atten = 80
# Corner frequency - f domain.
corner = 0.03


# Frequency vector
f = np.geomspace(5e-4, 1, num = 1001)


# Plotting.
_figsize = (8+1/4, 5+7/8)
_dpi = 150
_fontsize = 14.0
```

```python


# File output.
_outplots = f'Lower_DC_BLRMS_BL_Order-BL_Filter-Python_{_vers_str_}.'

_outtxt = f'Lower_DC_BLRMS_BL_Order-BL_SOS_G-Python_{_vers_str_}.txt'


#------------------------------------------------------------------------------

def main():

    """
    Run the investigation in Python. Initial previews DON'T LOOK PROMISING but
    there's nothing to do but try.
    """

    # Generate the sos.

    # Default.
    sos_dflt = sig.iirfilter(order, corner, ftype = 'ellip', btype = 'lowpass',
                             rp = ripple, rs = atten, fs = 1 / Ts,
                             output = 'sos')

    # Convert it into SOS, G form.
    sos_wG = np.array(sos_dflt)
    # Gain IS adjusted for the DC filter - 1 dB but should be sqrt of
    # this.
    g = sos_wG[0, 0]
    sos_wG[0, 0:3] *= 1 / g
    g *= 10**(1/20)

    # Generate the padding.
    sos_padding = np.zeros((4,6), dtype = float)
    sos_padding[:, 0] = 1
    sos_padding[:, 3] = 1

    # Add the padding
    sos8_wG = np.concatenate((sos_wG, sos_padding), axis = 0)


    # Generate the frequency response - not hopeful.
    # It does work.
    _, H_dflt = sig.sosfreqz(sos_dflt, worN = f, fs = 1 / Ts)
    _, _H_soswG = sig.sosfreqz(sos_wG, worN = f, fs = 1 / Ts)
    _, _H_sos8 = sig.sosfreqz(sos8_wG, worN = f, fs = 1 / Ts)

    # Convert to magnitude.
    mag_dflt = np.abs(H_dflt)
    mag_soswG = g * np.abs(_H_soswG)
    mag_sos8 = g * np.abs(_H_sos8)


    # Plot.
    fig_sos, ax_sos = plt.subplots(ncols = 1, nrows = 1, figsize = _figsize,
```

```
                                  dpi = _dpi)

    ax_sos.set_xscale('log', subs = np.arange(2, 10))
    ax_sos.set_xlim(left = np.min(f), right = np.max(f))
    ax_sos.set_yscale('log', subs = np.arange(2, 10))
    ax_sos.set_ylim(bottom = 10**(-90/20), top = 2)

    ax_sos.plot(f, mag_dflt, label = '4 SOS, Default', linewidth = 5.0,
                color = '#000000', zorder = 2)
    ax_sos.plot(f, mag_soswG, label = '4 SOS, with Gain', linewidth = 3.0,
                color = '#FFC107', linestyle = (0, (5,3)), zorder = 3)
    ax_sos.plot(f, mag_sos8, label = '8 SOS (Padded), with Gain', zorder = 4,
                color = '#1E88E5', linestyle = (0, (3,3,1,3)), linewidth = 2.5)

    ax_sos.set_xlabel('Frequency ($Hz$)', fontsize = _fontsize)
    ax_sos.set_ylabel('Magnitude', fontsize = _fontsize)
    ax_sos.set_title('Python 8th Order SOS LPFs', fontsize = _fontsize)
    ax_sos.tick_params(labelsize = _fontsize)
    leg_sos = ax_sos.legend(fontsize = _fontsize, loc = 'best')
    ax_sos.grid(which = 'major', color = '#536267')
    ax_sos.grid(which = 'minor', color = '#6b7c85', linestyle = (0, (2,2)),
                linewidth = 0.85)

    fig_sos.tight_layout()
    fig_sos.savefig(_outplots + 'png')
    fig_sos.savefig(_outplots + 'pdf')
    plt.close(fig_sos)


    # Print full sos for inspection.
    print('Printing full SOS for visual inspection.')

    with np.printoptions(precision = 15, linewidth = 80, sign = '+'):
        print('SOS:')
        print(sos8_wG)
        print('Gain:')
        print(g)


    # Dump to a text file.
    with open(_outtxt, 'x') as sos_file:
        # Format the string.
        sos_str = np.array2string(sos8_wG[:,[1,2,4,5]], max_line_width = 116,
                                  precision = 15, separator = ', ', sign = '+',
                                  floatmode = 'fixed')

        # Write SOS.
        sos_file.write('SOS:\n')
        sos_file.write(sos_str.replace('[', '{').replace(']', '}'))

        # Write Gain, with dynamic precision.
        g_prec = max(15 - int(format(g, 'e').split('e')[-1]), 15)
        sos_file.write('\nGain:\n')
        sos_file.write(format(g, f'.{g_prec}f'))
```

```python


    print('Done.')


#------------------------------------------------------------------------------

if __name__ == '__main__':
    main()


# END.
```

Listing 16: Python code to directly generate *Z domain* SOS DC BLRMS with reduced order.

B.5 Validating the New BLRMS

Code used to validate that the new BLRMS, sections 3.1 and 3.2, are operating correctly. This code is available from [1][20]. The data this analyses was originally posted in the SEI aLOG [10] and has also been copied to the [1] too[21][22]

```python
"""
A script to compare the BLRMS result to an actual RMS measurement. To ensure
there is no cheating I calculate the comparison RMS from first principles -
which is no different from the divide by sqrt(2) trick.

Author: Nathan A. Holland
Contact: nholland@nikhef.nl
Date: 2022-05-24

Version: 1.10

Changelog:
    2022-05-25 Removed redundant and confusing additional analysis.
    2022-05-24 Created, using the data I have.
"""


#------------------------------------------------------------------------------

import numpy as np

import scipy.io as sio
from scipy.integrate import quad

import matplotlib.pyplot as plt

import h5py
```

[20] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/documentation/generate_rms_comparison.py

[21] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/documentation/High_Frequency_BLRMS_Outputs-100cnts_P2P__300s-SEI_aLOG1918-20220520.mat

[22] /BSC-ISI/Stanford/s1isi_tools/design_BLRMS/cust_BLRMS/documentation/BLRMS_Testing_EXC_Select-20220520.csv

```python


#-------------------------------------------------------------------------------

# Script variables.

# Input time series
_infile = 'High_Frequency_BLRMS_Outputs-100cnts_P2P__300s-SEI_aLOG1918-' + \
'20220520.mat'

_selfile = 'BLRMS_Testing_EXC_Select-20220520.csv'

# Actually twice what Jim quotes in his aLOG
_exc_amplitude_p2p = 200

# Seconds needed to synchronise with the NDScope plot.
_total_time = 300
_pic_offset = -44

# Sample rate of the data I have.
_sample_rate = 16


# Frequencies.
freqs = [50, 75, 115, 160, 215]


#-------------------------------------------------------------------------------

# Helper functions.

def true_rms(frequency : float, amplitude : float = _exc_amplitude_p2p):

    """
    A function to calculate the TRUE, time series, RMS of a signal, with a
    given frequency. Not really necessary because I cpould just divide the
    amplitude by np.sqrt(2).

    Usage:
     rms = true_rms(frq, amplitude = amp)

    Inputs:
     frq - The linear, signal frequency in Hz.
     amp - OPTIONAL The amplitude of the sinusoid, peak to peak.
           DEFAULT value is 100.

    Output:
     rms - The RMS, single cycle, of the time series.
    """

    # Generate the integration boundaries.
    t_left = -np.pi / (2*np.pi * frequency)
    t_right = np.pi / (2*np.pi * frequency)


    # Generate the integration function.
```

```
    int_func = lambda time : (amplitude/2 * np.sin(2*np.pi*frequency*time))**2


    # Integrate the function.
    integral, _ = quad(int_func, t_left, t_right)

    # Mean squared,
    ms = integral / (t_right - t_left)

    return np.sqrt(ms)


#------------------------------------------------------------------------------

def main():

    """
    Runs the main purpose of this script, comparing the BLRMS values to real
    RMS values.
    """

    # Open the data.
    dat_dict = sio.loadmat(_infile, squeeze_me = True)

    # Upack the data.
    channels = dat_dict['channels'].tolist()
    GPSstart_time = dat_dict['start_time']
    BLRMS_data = dat_dict['cps_blrms']

    blrms_065_100 = BLRMS_data[:,0]
    blrms_130_200 = BLRMS_data[:,1]


    # Generate a time vector, in minutes since 1337111434 GPS
    time = np.arange(blrms_065_100.size) / _sample_rate
    time -= _total_time
    time -= _pic_offset
    time /= 60

    # Get the time selection points.
    sel_mins = np.loadtxt(_selfile)


    # Cut the data.
    # Based upon the image I have been provided.
    sel_050 = np.abs(time - sel_mins[0]).argmin()
    sel_075 = np.abs(time - sel_mins[1]).argmin()
    sel_115 = np.abs(time - sel_mins[2]).argmin()
    sel_160 = np.abs(time - sel_mins[3]).argmin()
    sel_215 = np.abs(time - sel_mins[4]).argmin()


    # Compare the RMS.
    for frq in freqs:
        # Get the string, length 3.
        f = f'{frq:03}'
```

```

        # True RMS value.
        true_rms_val = true_rms(frq)

        # Estimated RMS, from BLRMS.
        blrms_065_100_sel = blrms_065_100[eval(f'sel_{f}')]
        blrms_130_200_sel = blrms_130_200[eval(f'sel_{f}')]

        # Get the dB difference.
        diff_065_100_sel = 20 * np.log10(blrms_065_100_sel / true_rms_val)
        diff_130_200_sel = 20 * np.log10(blrms_130_200_sel / true_rms_val)

        # Report to the user.
        print(f'\n{frq} Hz')
        print(f'RMS Value: {true_rms_val:.4g}')
        print(f'65 Hz to 100 Hz BLRMS: {blrms_065_100_sel:.4g}' + \
              f' ({diff_065_100_sel:.4g} dB)')
        print(f'130 Hz to 200 Hz BLRMS: {blrms_130_200_sel:.4g}' + \
              f' ({diff_130_200_sel:.4g} dB)')


#------------------------------------------------------------------------------

if __name__ == '__main__':
    main()


# END.
```

Listing 17: Python code to validate the performance of the new custom BLRMS from sections 3.1 and 3.2.