RICH ORMISTON

ANALYTIC APPROACH TO NONLINEAR AND NONSTATIONARY FILTERING

University of Melbourne Meeting: August 2020

LIGO

LINEAR FILTERS

STATIONARY LINEAR FILTERS

SISO WIENER FILTER

Estimate of the noise in the signal d[k] from witness x[k]

$$y[k] = \sum_{m=0}^{M-1} a_k[m]x[k-m]$$

Minimizing the MSE sets the filter coefficients

$$\frac{\partial}{\partial a_k[n]}\langle e[k]^2\rangle = \frac{\partial}{\partial a_k[n]}\langle \big(d[k]-y[k]\big)^2\rangle$$

$$= -2\langle d[k]x[k-n]\rangle + 2\sum_{m=0}^{M-1} a_k[m]\langle x[k-m]x[k-n]\rangle$$

$$= \mathbf{p} + \mathbf{a}\hat{\mathbf{R}}$$

$$\longrightarrow \quad \mathbf{a} = \hat{\mathbf{R}}^{-1}\mathbf{p}$$

STATIONARY LINEAR FILTERS

MISO WIENER FILTER

Estimate of the noise in the signal d[k] from witness x[k] for the bth channel

$$y[k] = \sum_{b=0}^{B-1} \sum_{m=0}^{M-1} a_k^b[m] x^b[k-m]$$

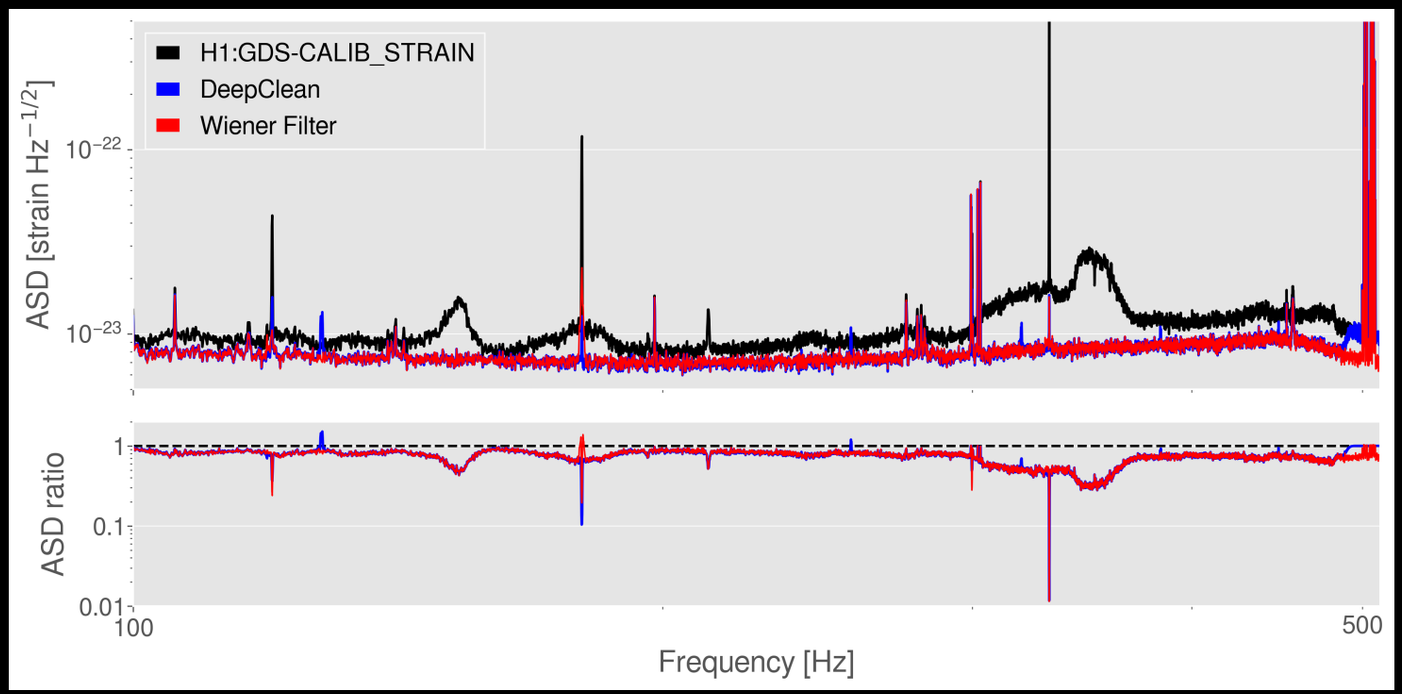
Minimizing the MSE sets the filter coefficients

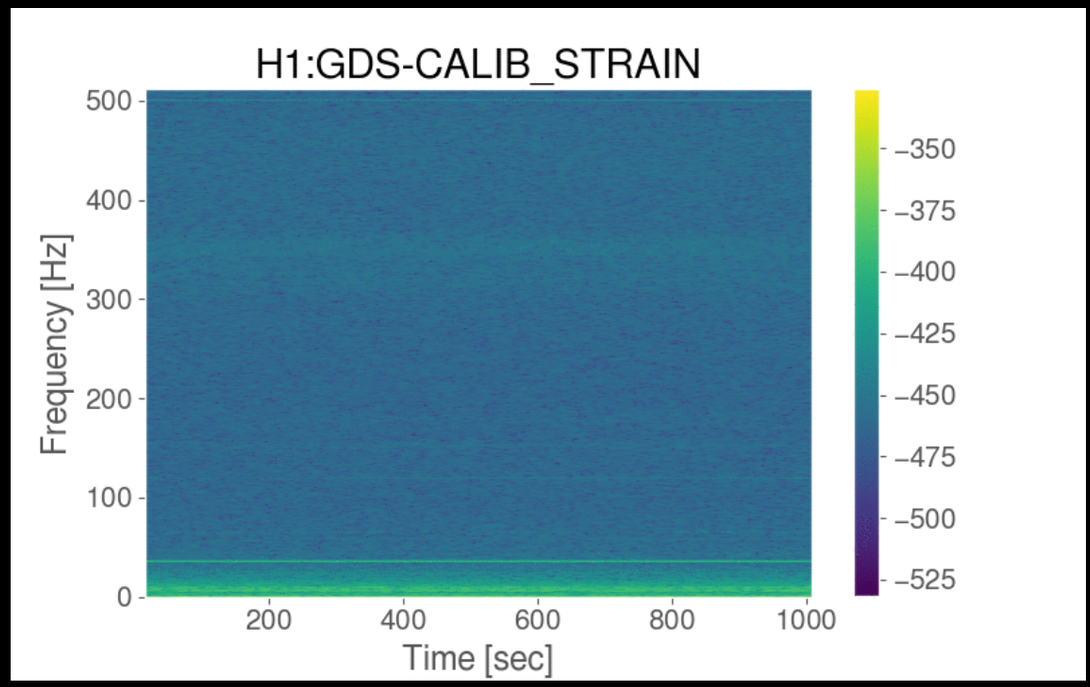
$$\frac{\partial \left\langle e^2[k] \right\rangle}{\partial a_k^{(b')}[j]} = -2 \left\langle d[k] \begin{pmatrix} x^0[k-j] \\ x^1[k-j] \\ \vdots \\ x^{B-1}[k-j] \end{pmatrix} \right\rangle + 2 \left\langle y[k-j] \begin{pmatrix} x^0[k-j] \\ x^1[k-j] \\ \vdots \\ x^{B-1}[k-j] \end{pmatrix} \right\rangle$$

$$\begin{pmatrix} \mathbf{p}^{(0)} \\ \mathbf{p}^{(0)} \\ \vdots \\ \mathbf{p}^{(B-1)} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{R}}^{(0,0)} & \hat{\mathbf{R}}^{(1,0)} & \cdots & \hat{\mathbf{R}}^{(B-1,0)} \\ \hat{\mathbf{R}}^{(0,1)} & \hat{\mathbf{R}}^{(1,1)} & \cdots & \hat{\mathbf{R}}^{(B-1,1)} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\mathbf{R}}^{(0,B-1)} & \hat{\mathbf{R}}^{(1,B-1)} & \cdots & \hat{\mathbf{R}}^{(B-1,B-1)} \end{pmatrix} \begin{pmatrix} \mathbf{a}^{(0)} \\ \mathbf{a}^{(1)} \\ \vdots \\ \mathbf{a}^{(B-1)} \end{pmatrix}$$

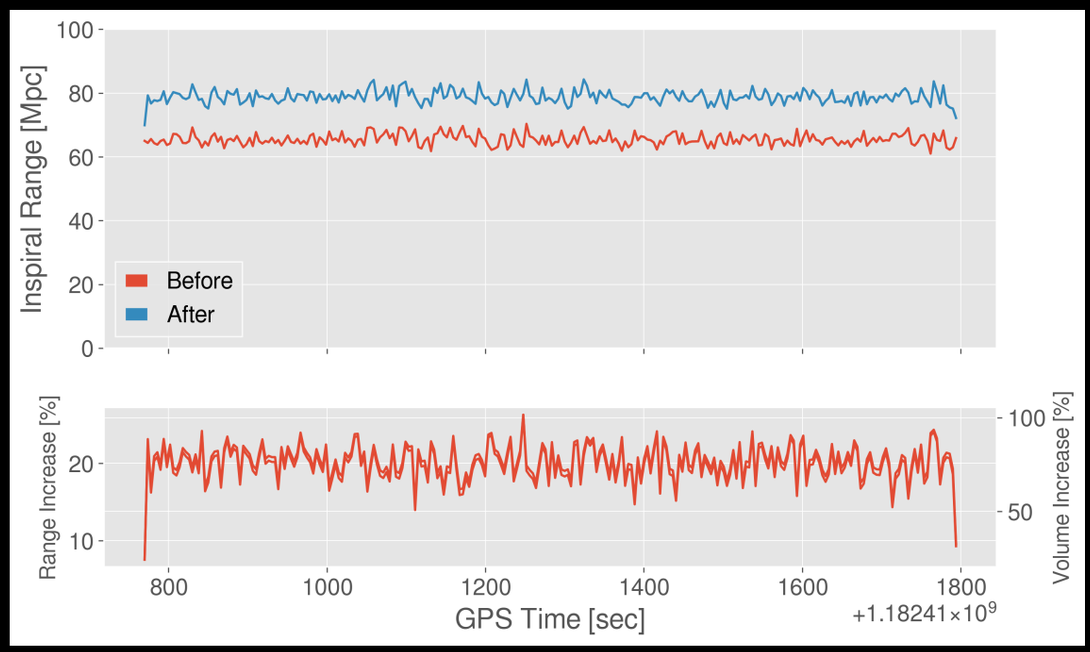
STATIONARY LINEAR FILTERS

MISO WIENER FILTER (W/ DEEPCLEAN)





Neural networks like DeepClean can do linear subtraction as well as the analytic Wiener filter



LIGO

NONLINEAR FILTERS

STATIONARY NONLINEAR FILTER

VOLTERRA FILTER

Second order term of the filter expansion

$$y[k] = a_0 + \sum_{m=0}^{M-1} a_k[m]x[k-m] + \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} a_k[m,m']x_1[k-m]x_2[k-m'] + \mathcal{O}(x^3)$$

$$\left\langle d[k]x_1[k-i]x_2[k-j] \right\rangle = \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} a_k[m,m']\left\langle x_1[k-m]x_2[k-m']x_1[k-i]x_2[k-j] \right\rangle$$

$$P^{(1,2)}[i-j] = \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} a_k[m,m']C^{(i,j)}[m,m']$$

STATIONARY NONLINEAR FILTER

VOLTERRA FILTER: MOCK DATA

Witnesses

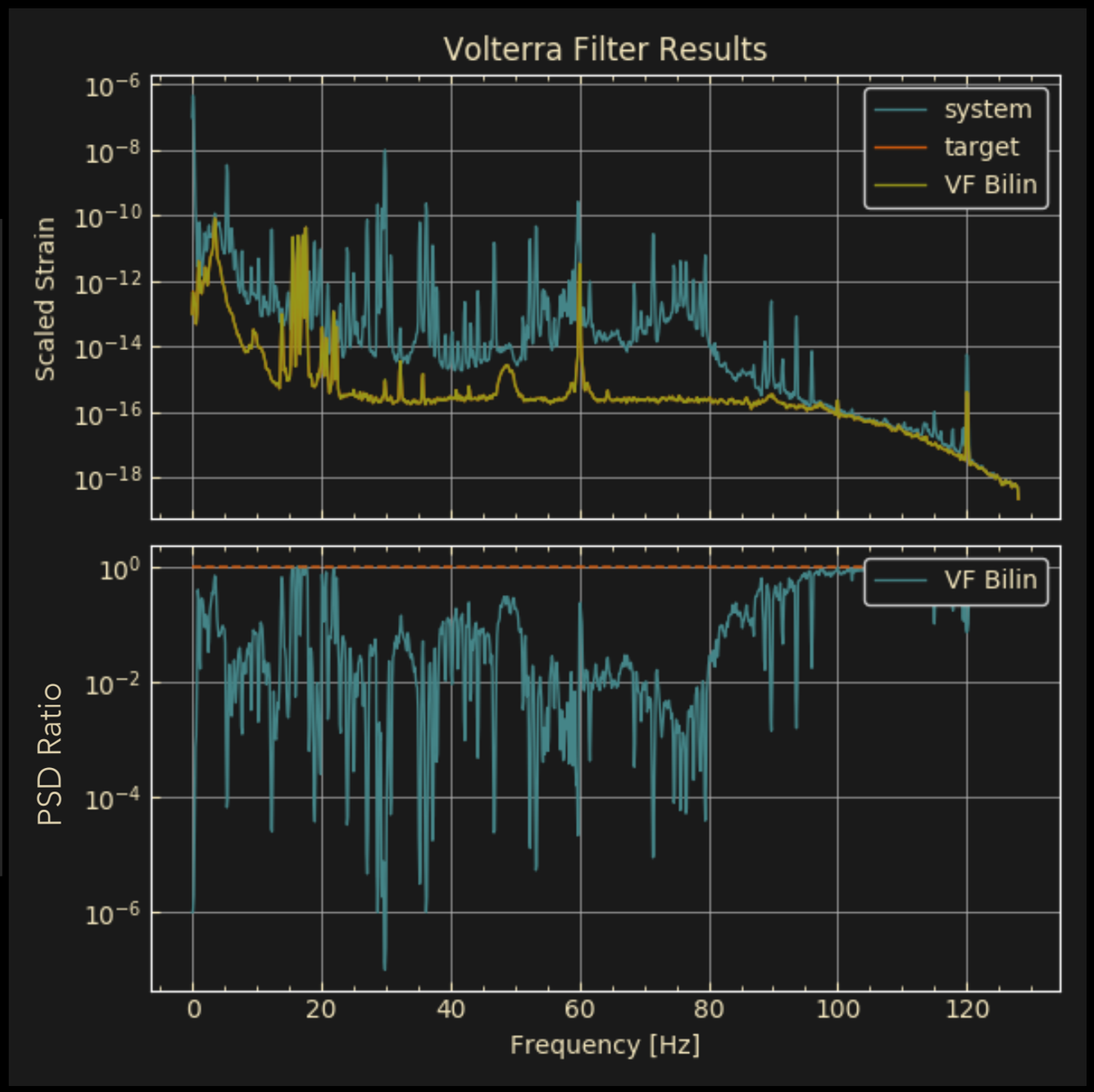
```
#------------------------------------------------------------------------------
#                        Load Data and Making Testing Set
#------------------------------------------------------------------------------
print('Loading data and creating testing sets')
fs, dur = 256, 512
channels = ['H1:PEM-CS_SEIS_LVEA_VERTEX_X_DQ', 'H1:PEM-CS_SEIS_LVEA_VERTEX_Y_DQ',
            'H1:CAL-DELTAL_EXTERNAL_DQ']

wits = lib.stream_data(1253370524, channels, dur=dur, fsup=fs, ifo='H1')
for chan in range(wits.shape[0]):
    wits[chan, :] -= np.mean(wits[chan, :])

# Generate Target (true) Signal
t = np.linspace(0, dur, dur * fs)
true = wits[2, :]

# Bilinear Channels
coupled = wits[0, :] * wits[1, :]
norm_coup = coupled * np.max(true)/np.max(coupled)*200
d_bilin_wit = true + norm_coup
```

Target



Perfect subtraction for simple 1-tap bilinear noise

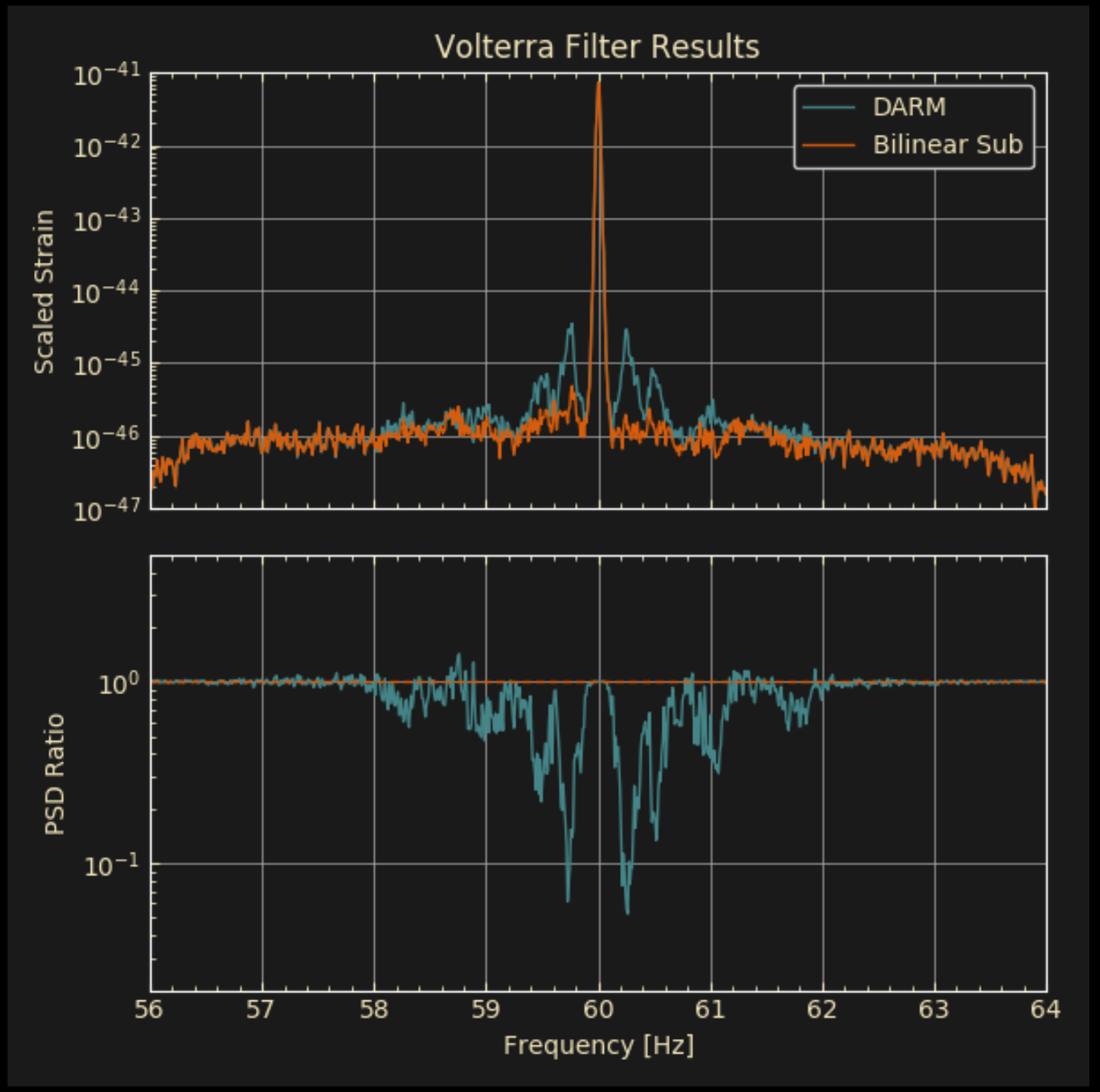
STATIONARY NONLINEAR FILTER

VOLTERRA FILTER: 60 HZ SIDEBANDS

The 60 Hz sidebands can be removed by using:

H1:PEM-CS_MAINSMON_EBAY_1_DQ
Witness to 60 Hz noise

H1:ASC-DHARD_Y_INMON
Low frequency modulation 60 Hz line



STATIONARY NONLINEAR FILTER

NONSENS VS VOLTERRA FILTER

Recall the 2nd order expansion

$$y[k] = \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} a_k[m, m'] x_1[k-m] x_2[k-m']$$

If x_2 moves slowly relative to x_1 across the M taps (the sum over m' is just M*number), then the sum over m' drops out and we have a modulated linear filter

$$y[k] \approx x_2[k] \sum_{m=0}^{M-1} a_k[m] x_1[k-m]$$

This is how Gabriele's NonSens subtraction works. The update equation for the filter coefficients becomes

$$\mathbf{a}[k+1] = \mathbf{a}[k] + 2\mu e[k] x_2[k] \mathbf{x_1}[k]$$

NONSTATIONARY FILTERS

LEAKY NORMALIZED LMS

Estimate of the noise in the signal d[k] from witness x[k]

$$y[k] = \sum_{m=0}^{M-1} a_k[m]x[k-m]$$

Instantaneous error update:

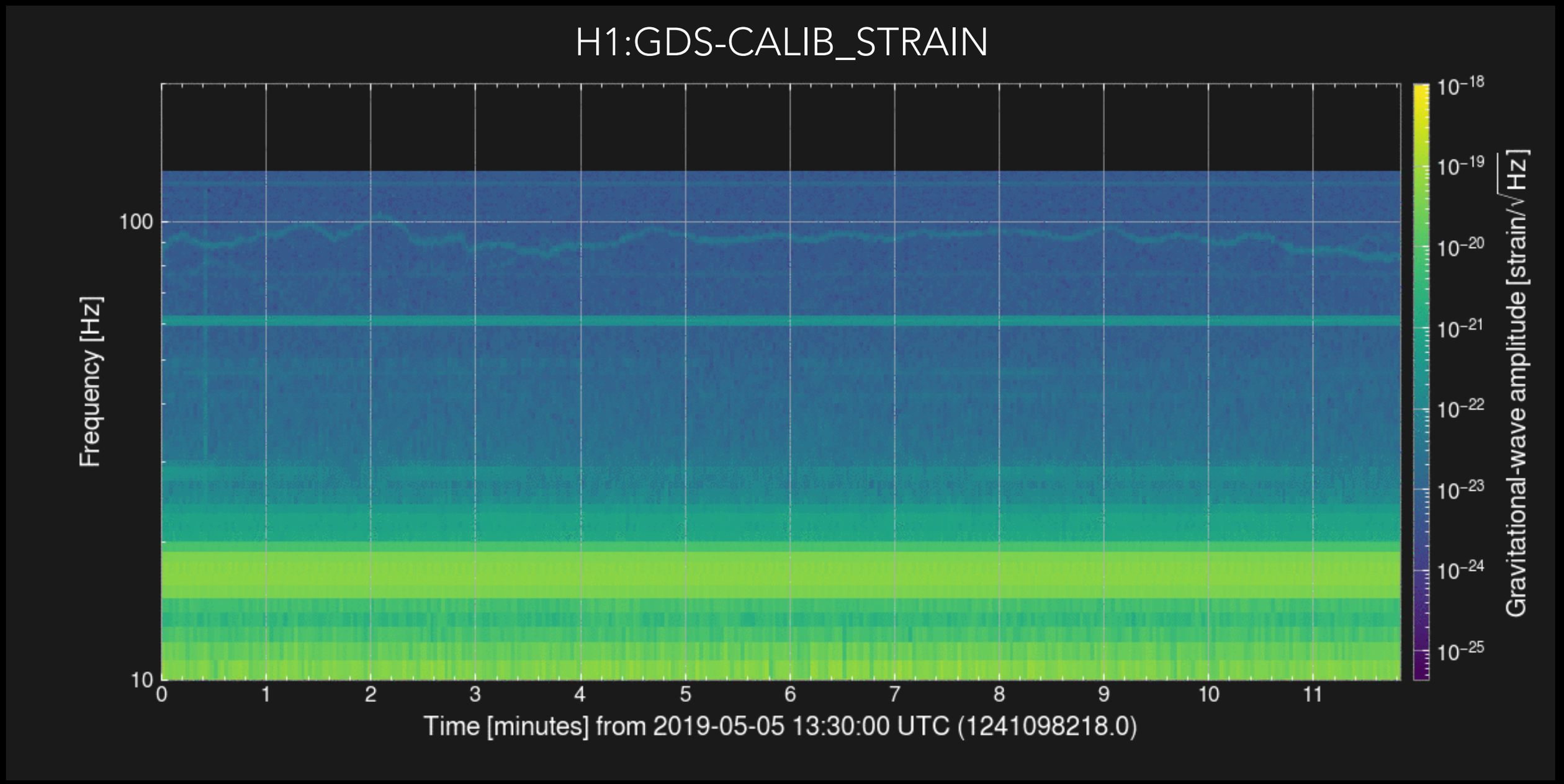
$$\begin{aligned}\mathbf{a}[k+1] &= \mathbf{a}[k] - \mu \overrightarrow{\nabla} e^2[k] \\ &= \mathbf{a}[k] + 2\mu e[k]\mathbf{x}[k]\end{aligned}$$

Normalizing the step size by the power and adding a ‘leak’ to the update gives

$$\mathbf{a}[k+1] = (1-\alpha)\,\mathbf{a}[k] - \frac{2\mu}{\mathbf{x}^T[k]\mathbf{x}[k] + \psi} e[k]\mathbf{x}[k]$$

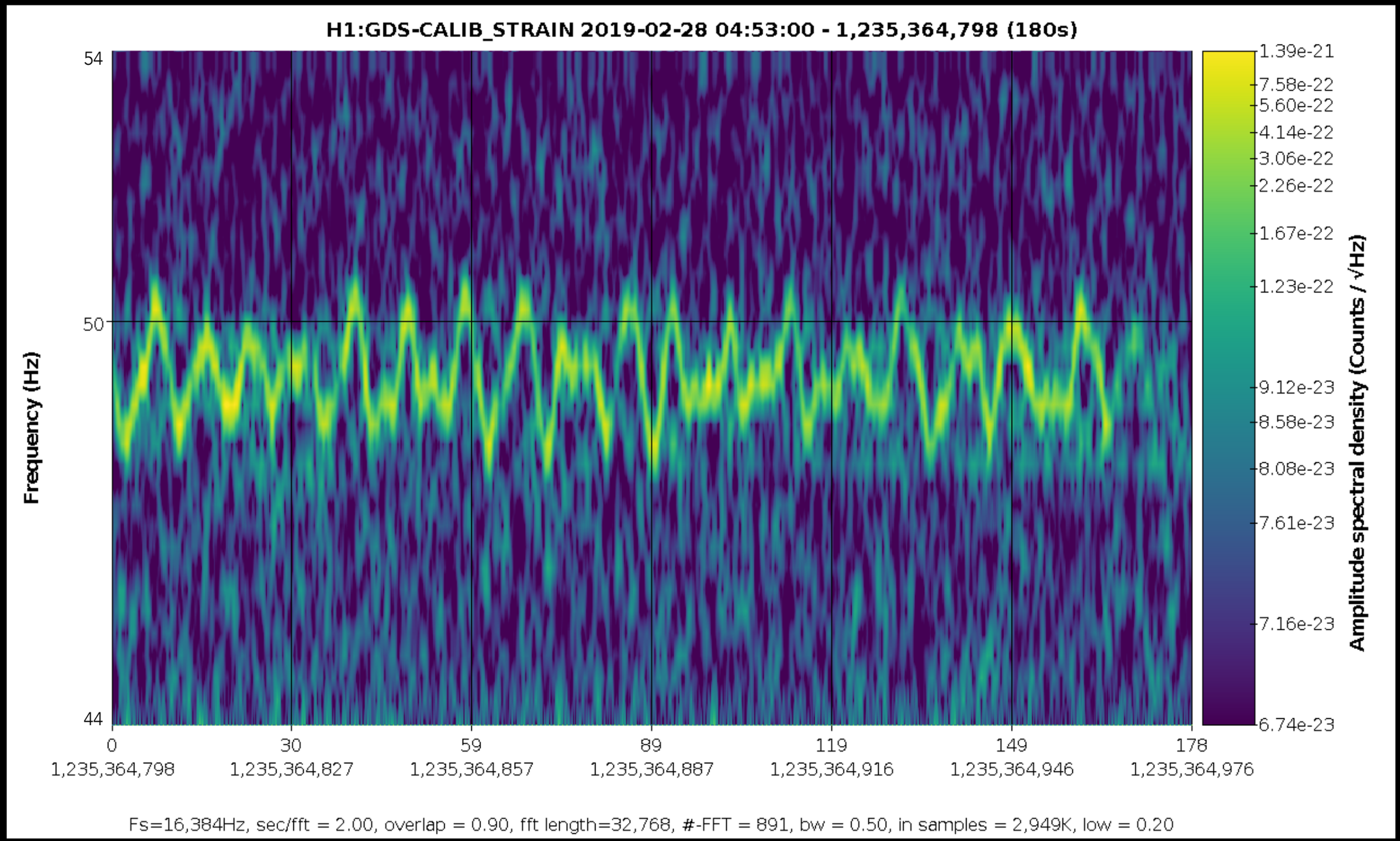
NONSTATIONARY LINEAR FILTERS

SQUEEZER WANDERING LINE



NONSTATIONARY LINEAR FILTERS

48HZ BUMP @ LHO: I



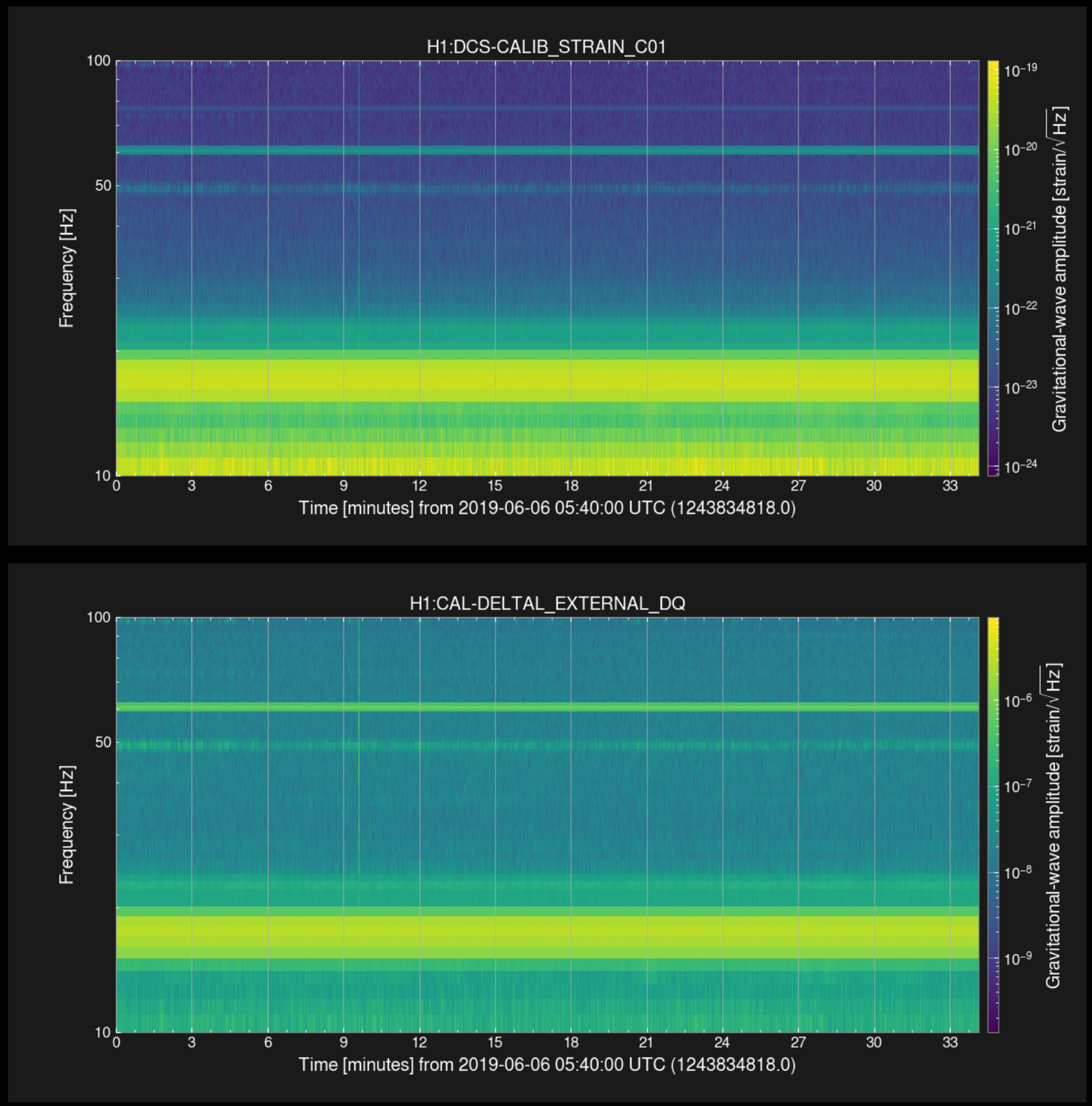
NONSTATIONARY LINEAR FILTERS

48HZ BUMP @ LHO: II

Noise apparently modulated by ~0.1Hz and ~0.2Hz motion (by eye).

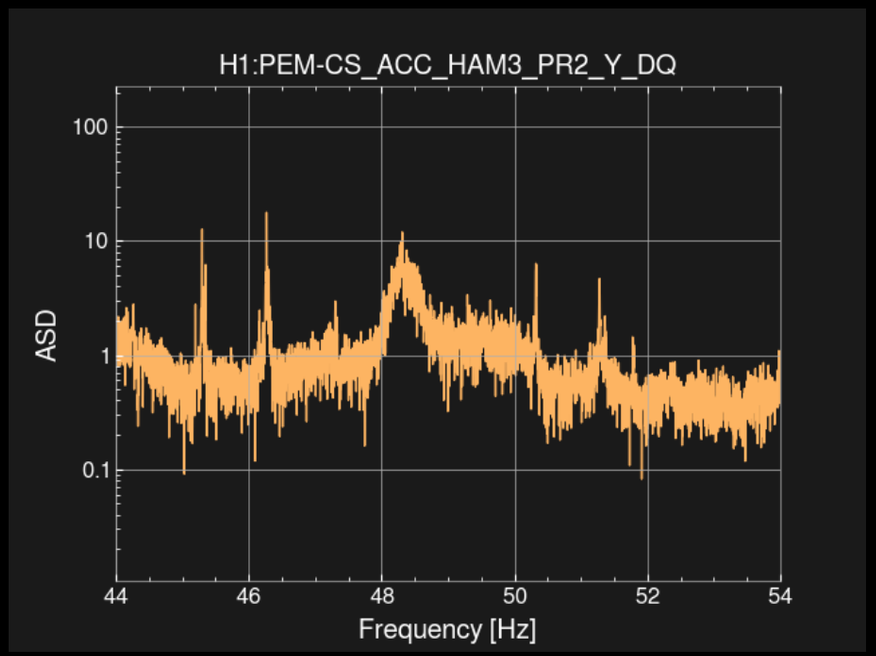
freq(t) would be very difficult to characterize.

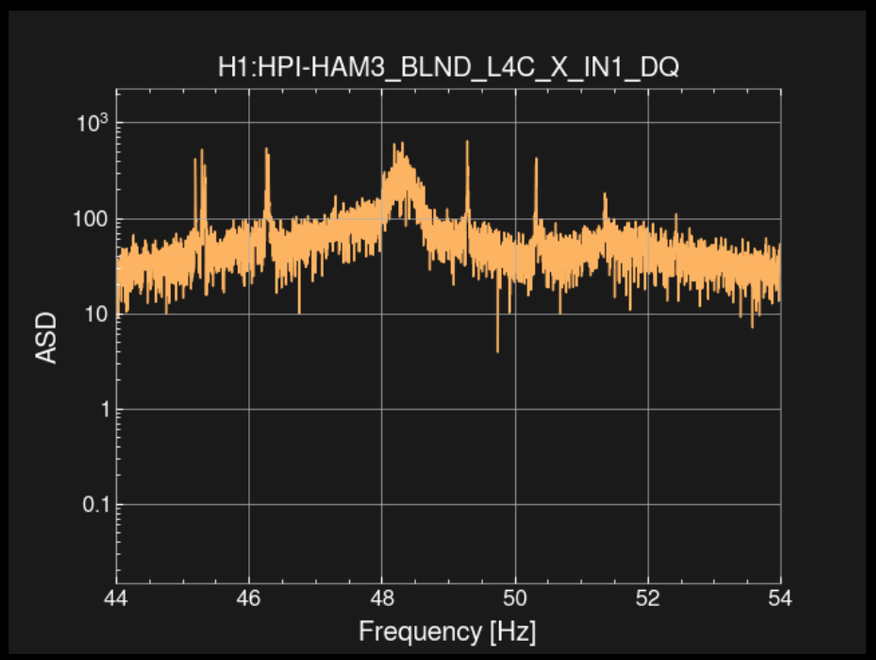
Possibly 3rd order filter or higher



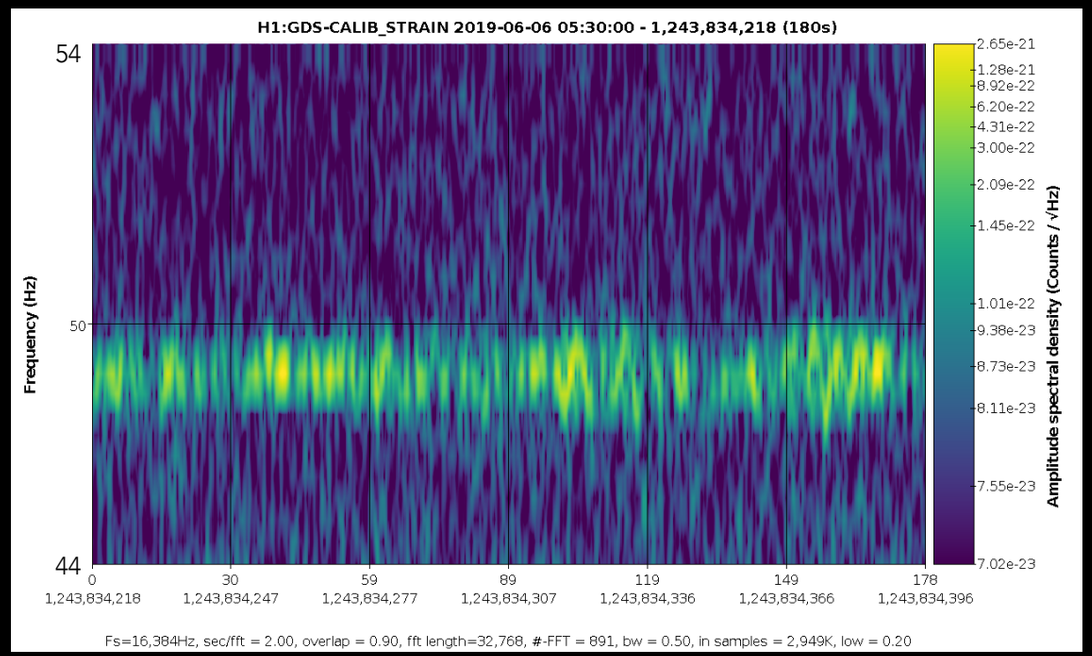
NONSTATIONARY LINEAR FILTERS

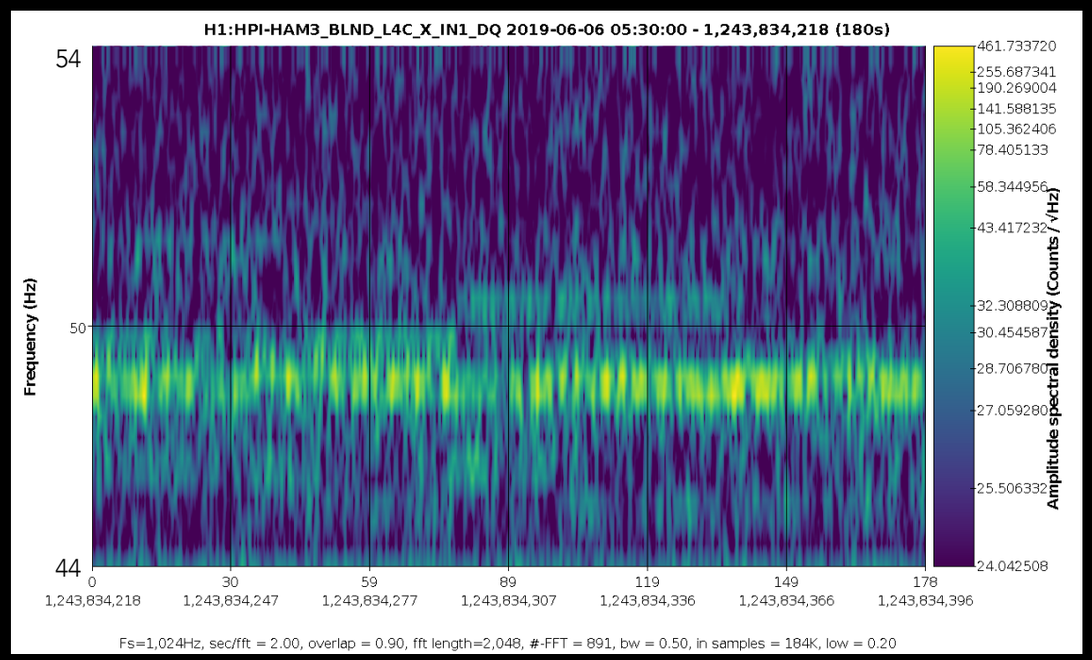
48HZ BUMP @ LHO: III





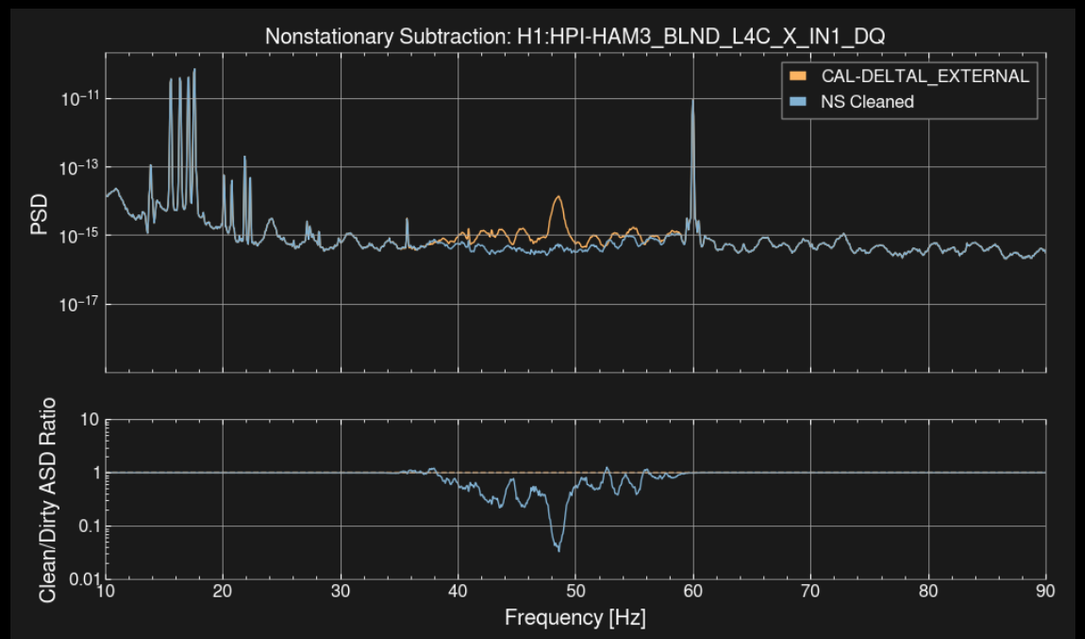
Maybe witnesses exist after all!

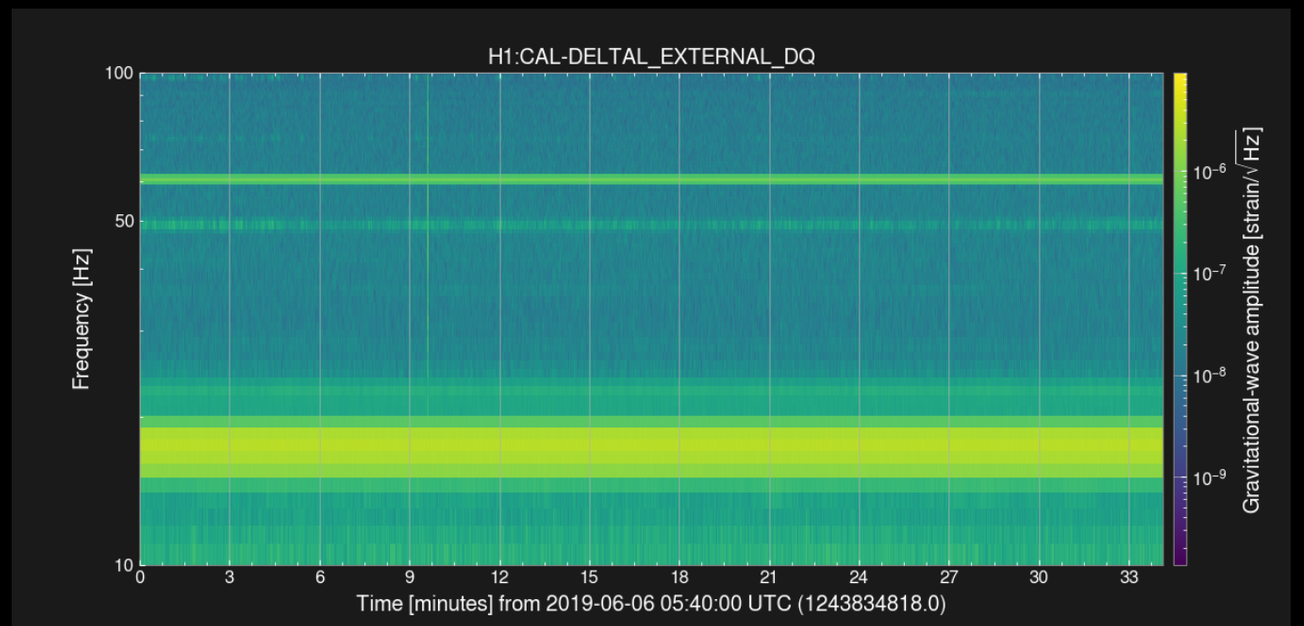




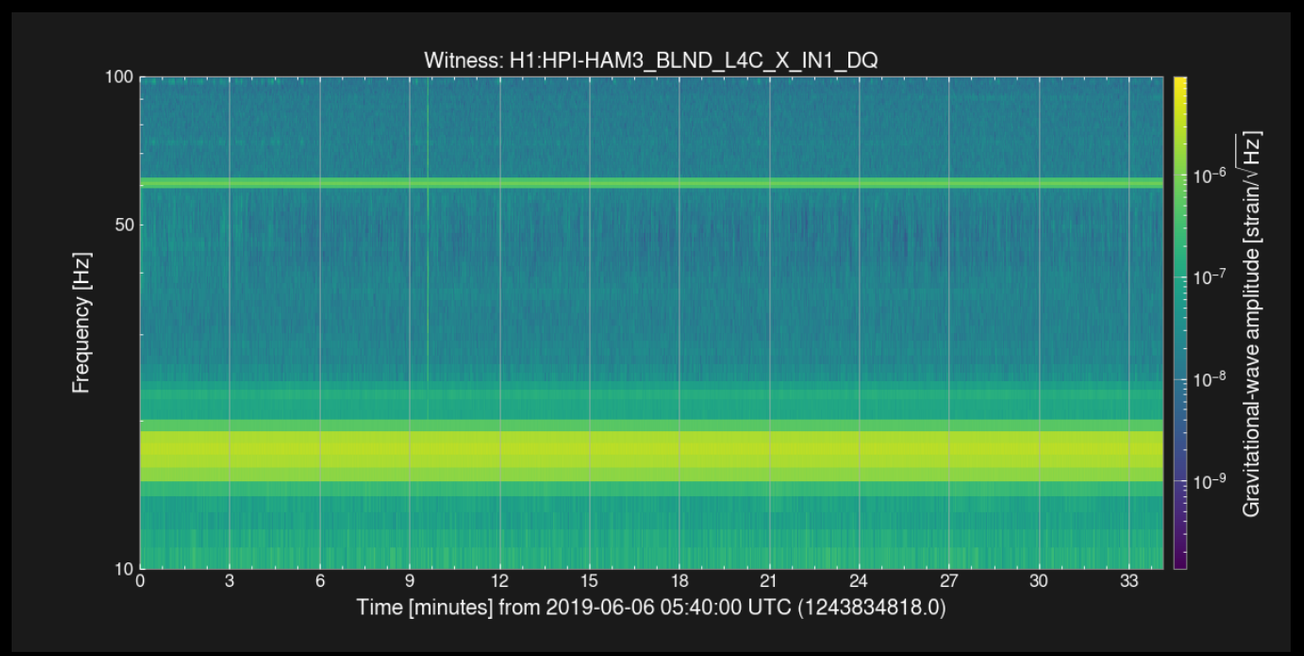
NONSTATIONARY LINEAR FILTERS

48HZ BUMP @ LHO: IV



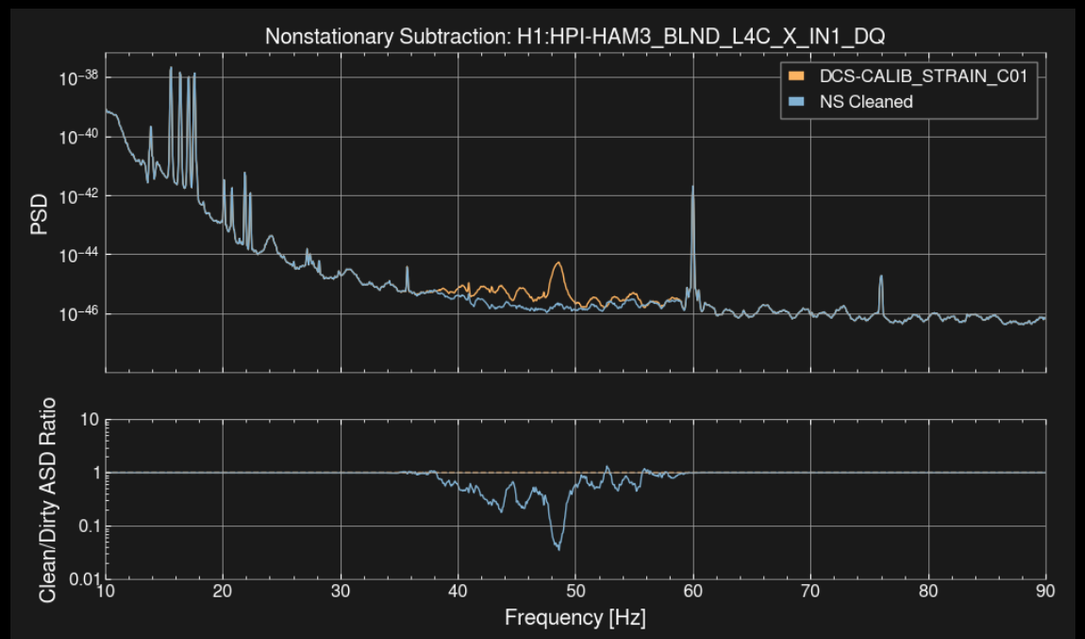


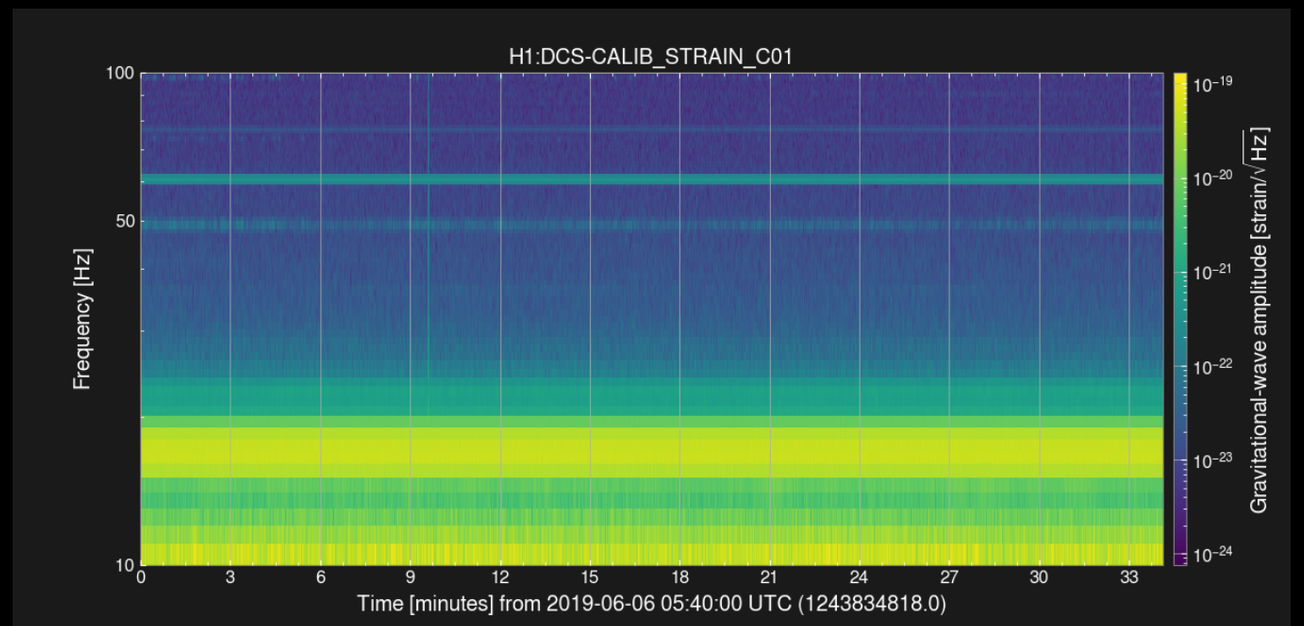
Seems to be removed from CAL-DELTAL_EXTERNAL_DQ



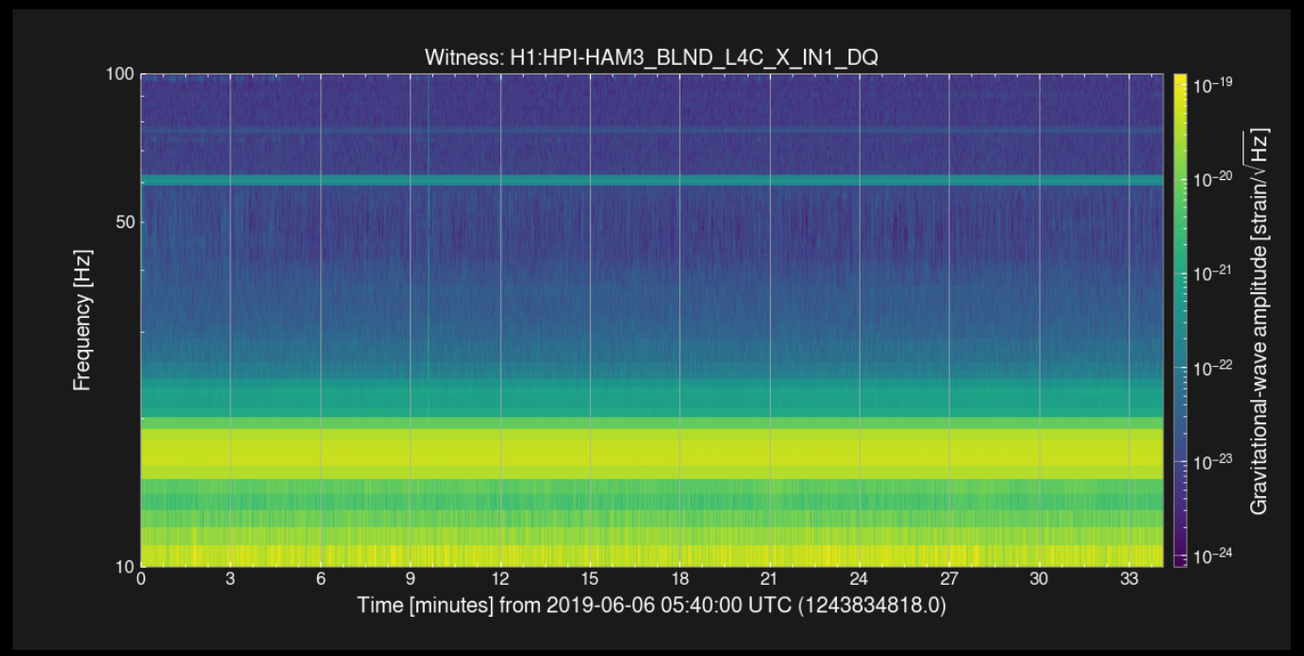
NONSTATIONARY LINEAR FILTERS

48HZ BUMP @ LHO: V





And from
DCS-CALIB_STRAIN_C01
(as it should)



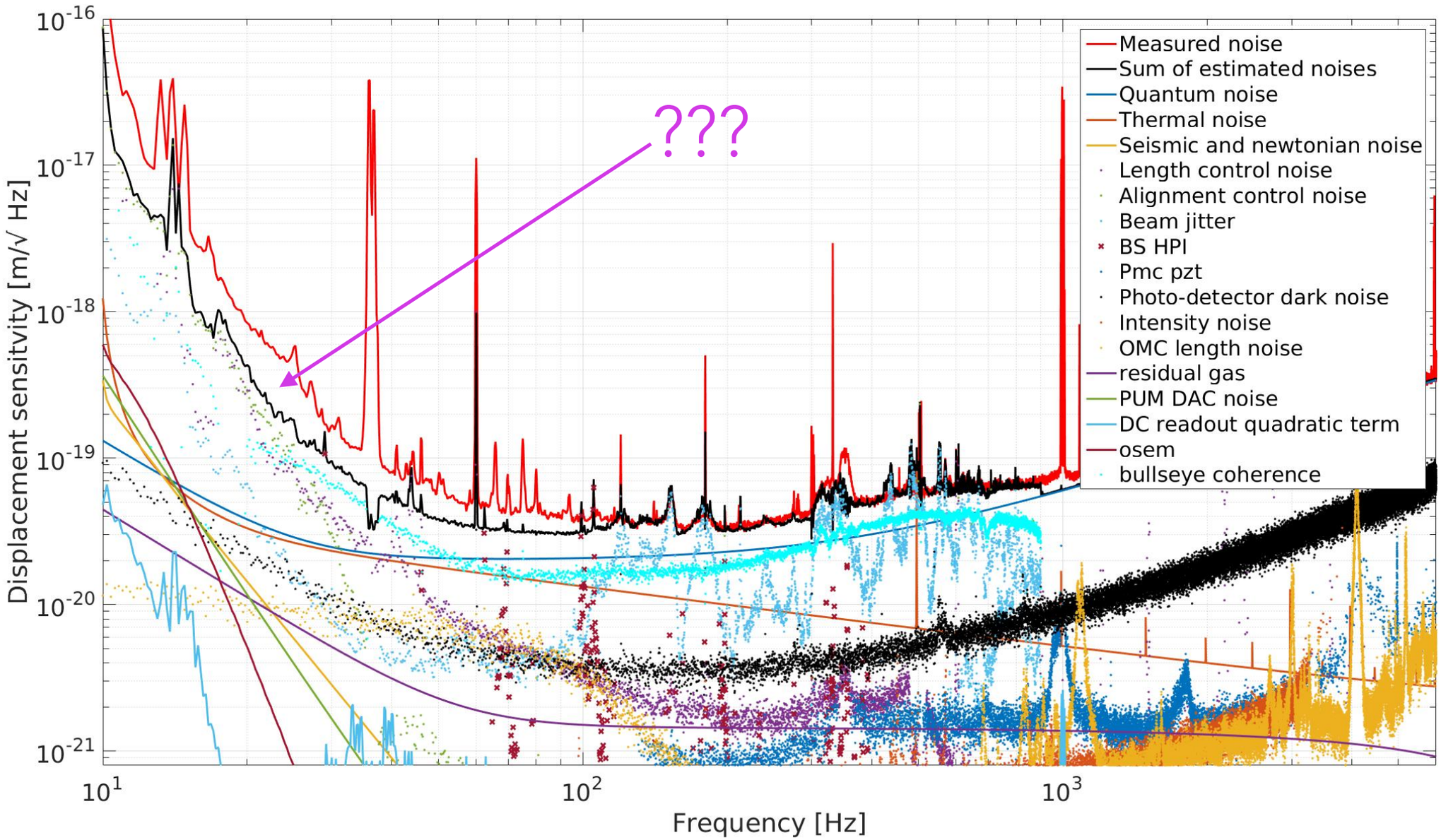
REFINERY SUMMARY

SUBTRACTION CHECKLIST

	Linear	Nonlinear
Stationary	Wiener Filter ✓	Mock Data 60Hz Sidebands ✓
Non-Stationary	Wandering Lines 48Hz Bump ✓	?

REFINERY SUMMARY

SUBTRACTION CHECKLIST



NEXT STEPS

- Promote noise estimate to account for cross-correlations

$$y[k] = \sum_{m=0}^{M-1} a_k[m]x[k-m] \quad \rightarrow \quad y[k] = \sum_{b=0}^{B-1}\sum_{m=0}^{M-1} a_k^b[m]x^b[k-m]$$

$$y[k] = \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} a_k[m,m']x_1[k-m]x_2[k-m'] \quad \rightarrow \quad y[k] = \sum_{b=0}^{B-1}\sum_{b'=0}^{B-1}\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} a_k^{(b,b')}[m,m']x_1^b[k-m]x_2^{b'}[k-m']$$

- Cast a wide net and see what other bilinear noise may be around
- Find a way to search through the potential couplings (combine with DeepClean?)

NEXT STEPS

- Non-stationary and nonlinear filter

$$\hat{\mathbf{a}}_{k+1} = \hat{\mathbf{a}}_k + 2\mu e[k](\mathbf{x_1}[k] \cdot \mathbf{x_2}^T[k])$$

$$= \hat{\mathbf{a}}_k + 2\mu e[k] \begin{pmatrix} x_1[k] \\ x_1[k-1] \\ \vdots \\ x_1[k-M+1] \end{pmatrix} \begin{pmatrix} x_2[k] x_2[k-1] \cdots x_2[k-M+1] \end{pmatrix}$$

- Warp the input instead of the filter. For example, for known nonlinearities (beating of CAL lines), input nonlinear data into a linear filter.

$$\mathbf{a}_{(1,2)}[n+1] = (1-\alpha)\mathbf{a}_{(1,2)}[n] - \frac{2\mu}{\mathbf{x}^T_{(1,2)}[n]\mathbf{x}_{(1,2)}[n] + \psi} e[n]\mathbf{x}_{(1,2)}[n]$$

  - Grid-search through coupling parameters

$$\hat{\mathbf{a}}_{k+1} = \hat{\mathbf{a}}_k + 2\mu e[k]\, f(\mathbf{x_1}[k], \mathbf{x_2}[k])$$

REFINERY SUMMARY

OUTLOOK

- Why not extend the Wiener filter’s success?

- Low-frequency low-hanging fruit?

- Right now, neural networks < analytic filters

- Every bit counts! The 48Hz problem may be fixable and may make you the emperor of detchar.

LIGO

THANK YOU!

REFINERY

GITLAB

OVERLEAF