

PyRing / Ringdown / PyCBC systematics studies

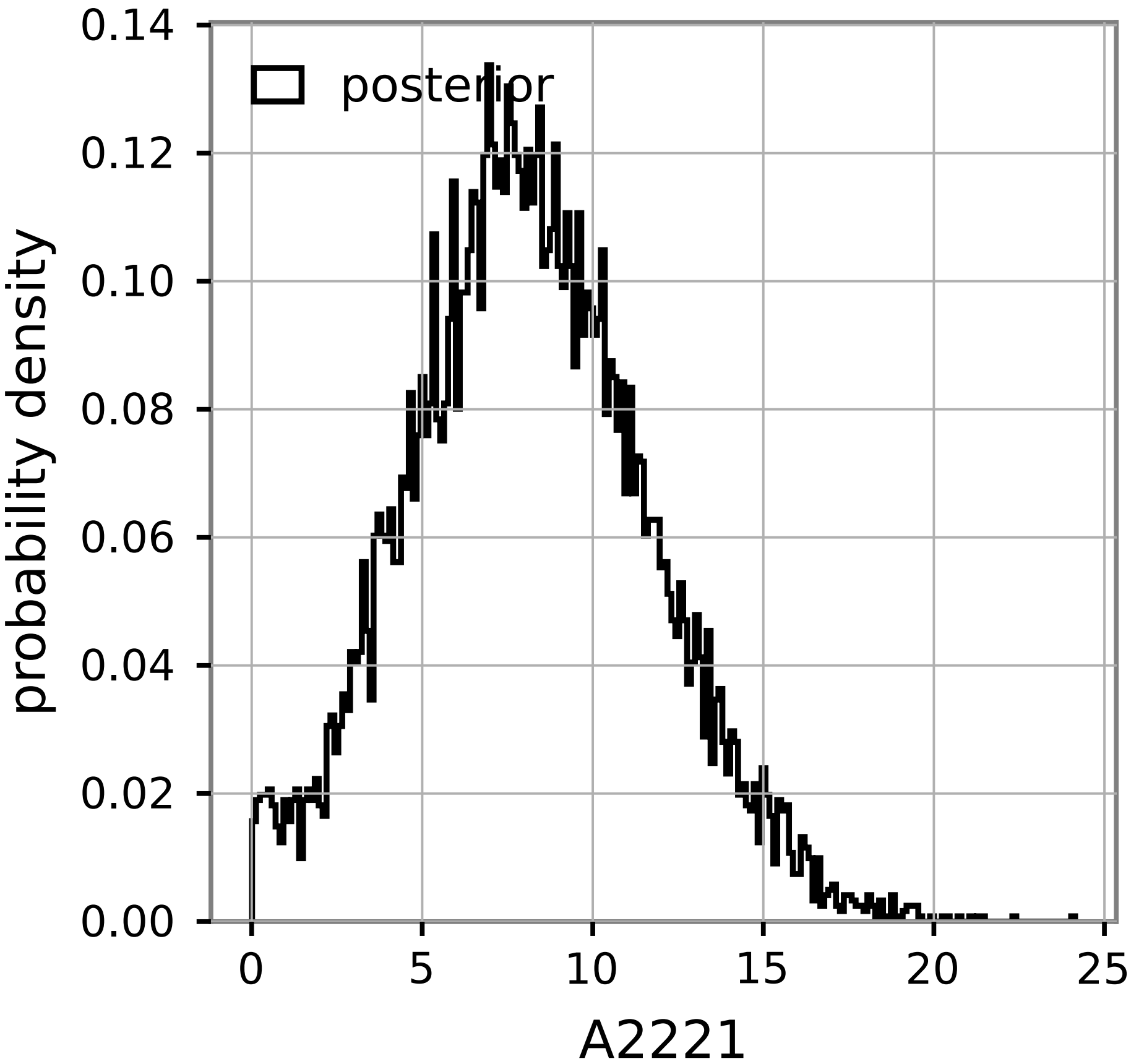
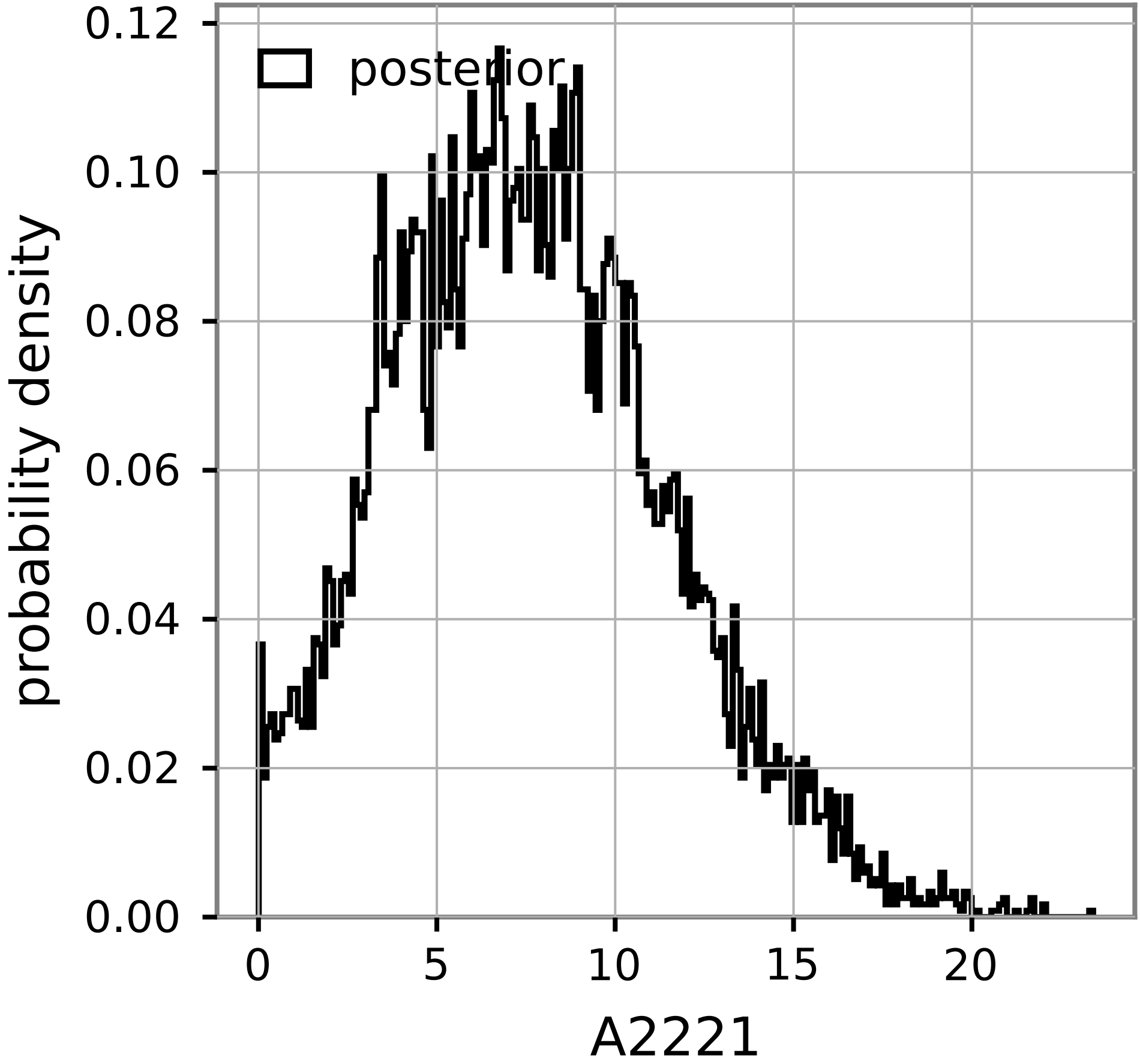
H_{220} : There is only 220 mode

H_{221} : There are $220 + 221$

$$P(d|H_{220}) = P(d|H_{221}, A_{221}=0) \\ = \frac{P(A_{221}=0|d, H_{221})P(d|H_{221})}{P(A_{221}=0|H_{221})}$$

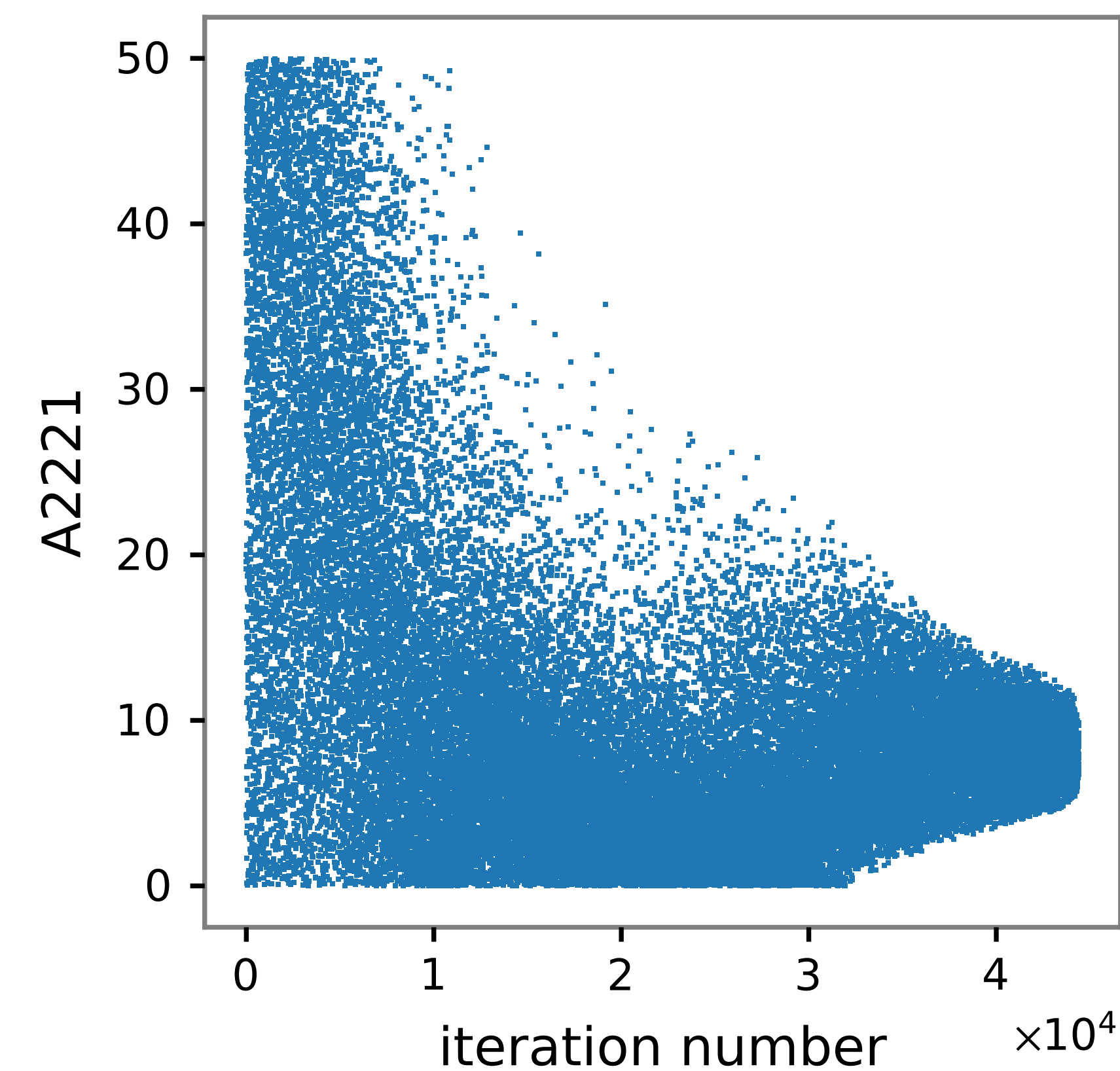
$$\therefore \frac{P(d|H_{220})}{P(d|H_{221})} = \frac{P(A_{221}=0|d, H_{221})}{P(A_{221}=0|H_{221})} \\ = \frac{\text{posterior } A_{221}=0}{\text{prior } A_{221}=0}$$

PyRing results: $t=t_{\text{ref}}$

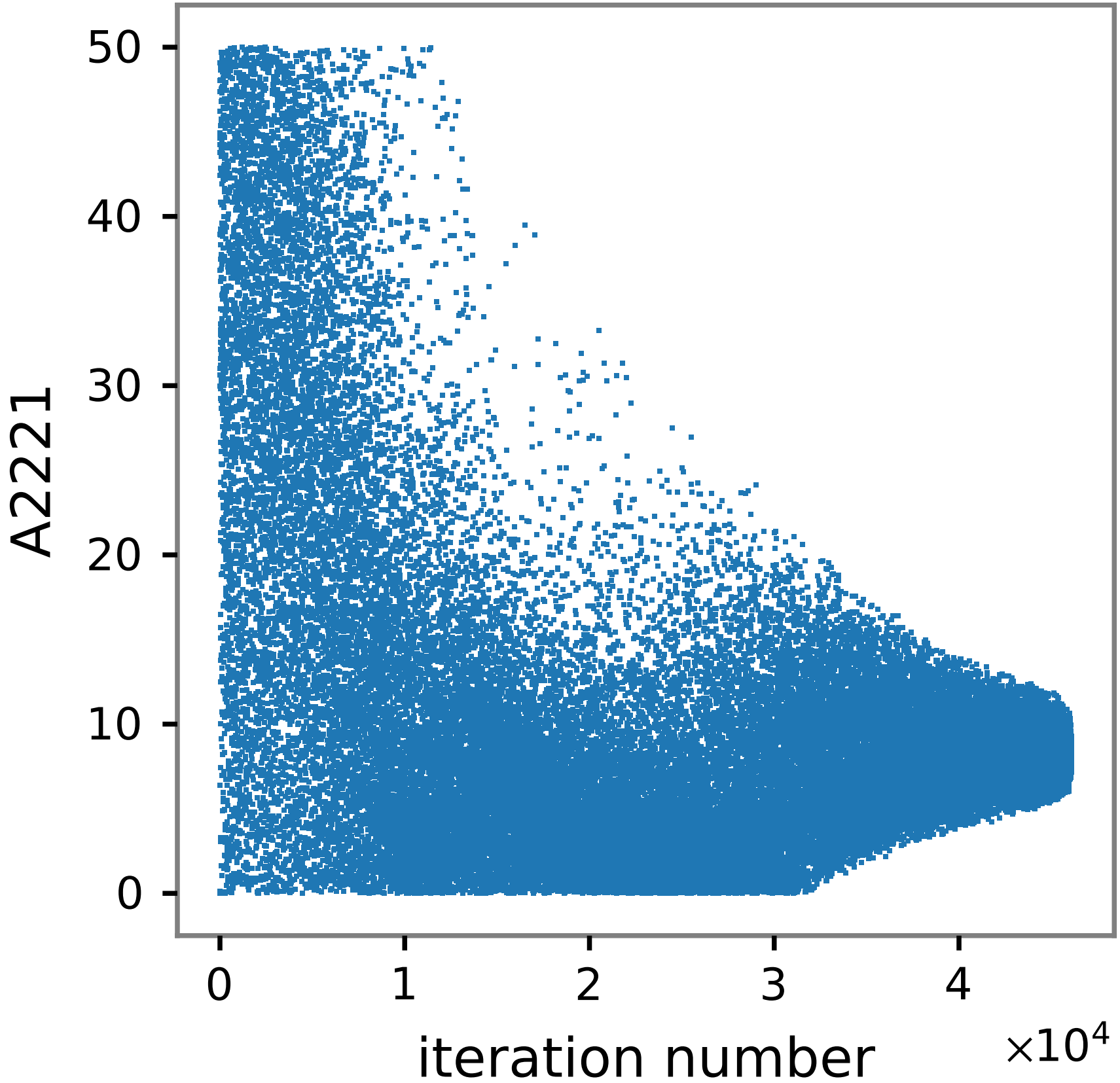


(For a reference: Ringdown uses 0.2 s)

PyRing results: $t=t_{\text{ref}}$

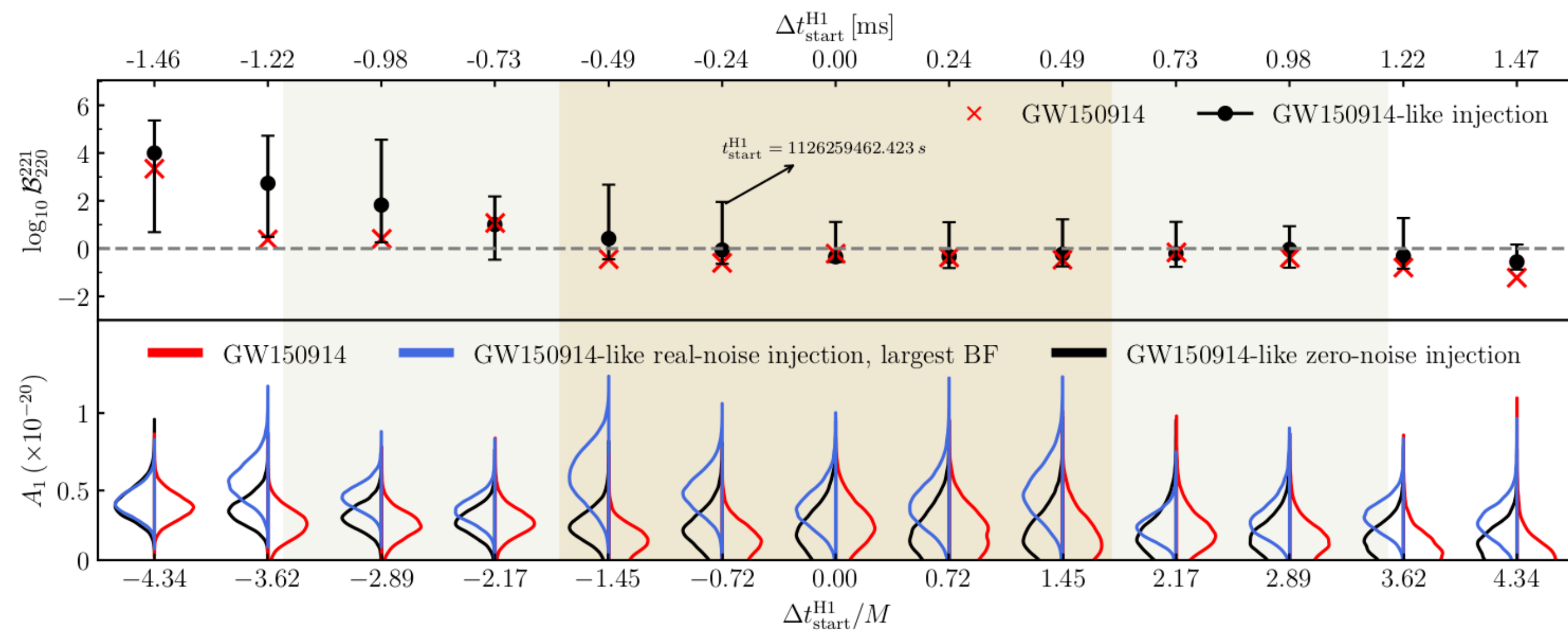


Signal length = 0.1s

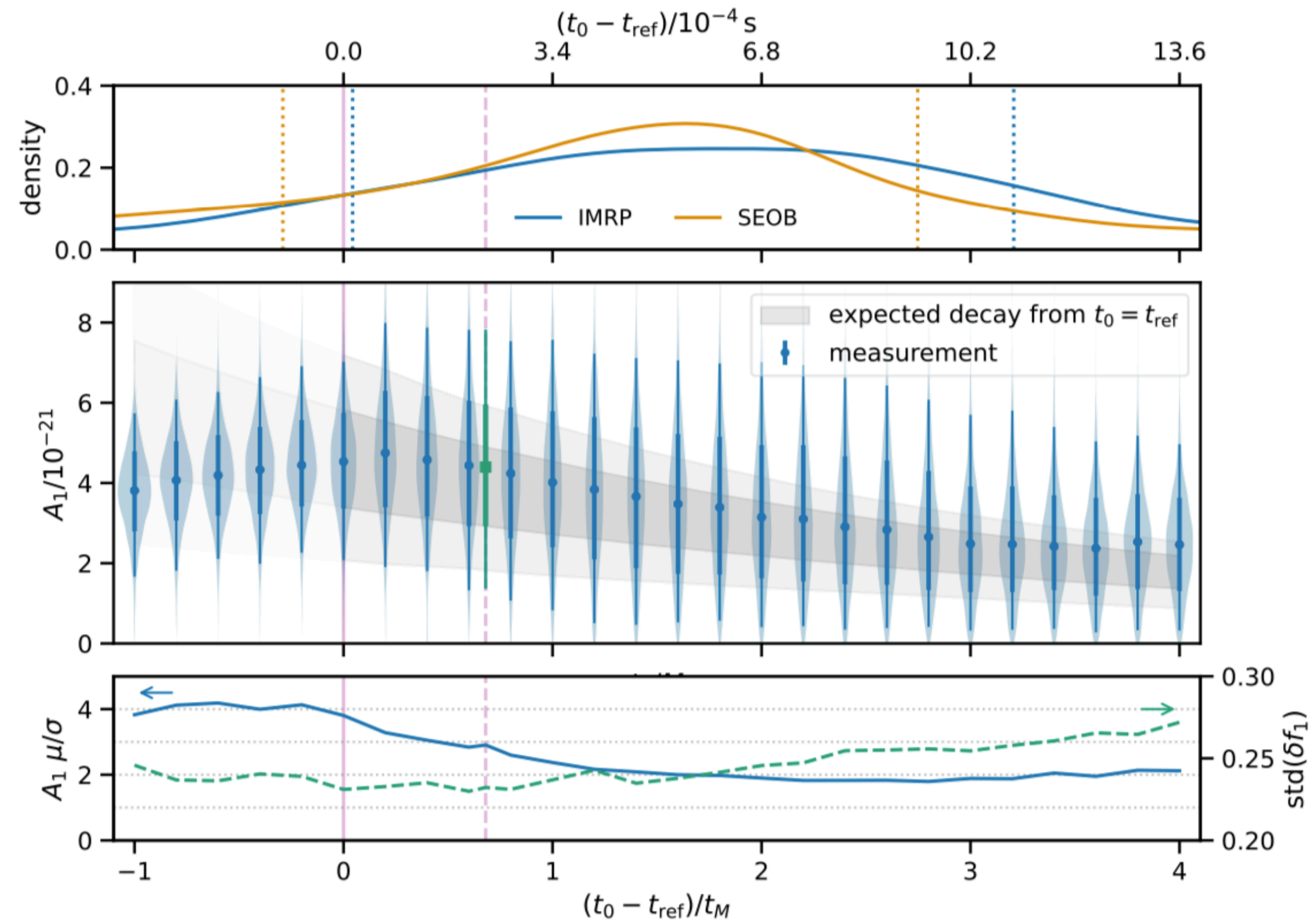


Signal length = 0.2s

(For a reference: Ringdown uses 0.2 s)

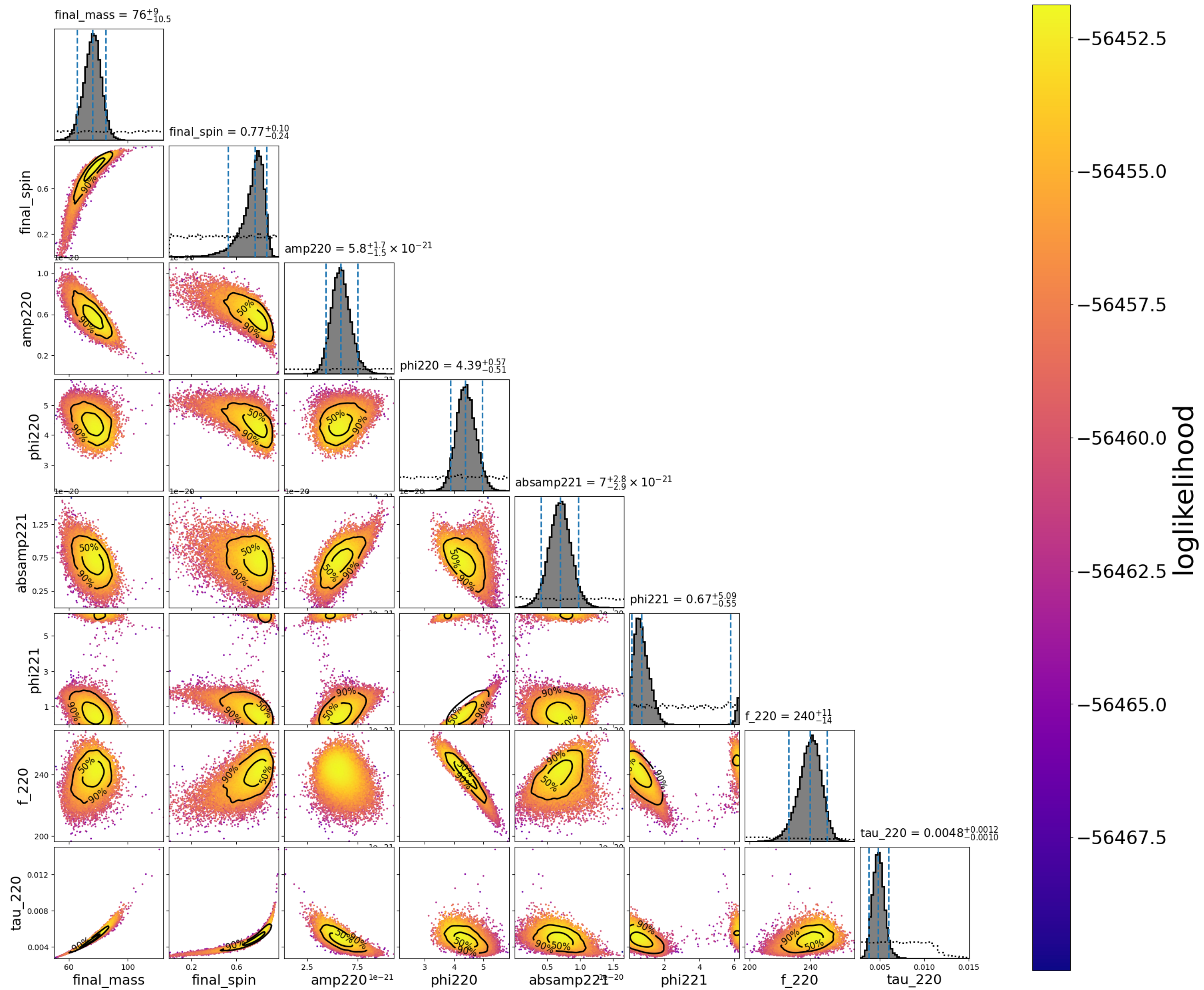


PyRing



Ringdown

PyCBC using PyRing PSD, prior as close to PyRing as possible



Possible systematics:

checklists:

- Waveform (checked! Ringdown and PyRing are consistent!)
- Data conditioning (raw data .gwf/.hdf/.txt checked, all are the same, but are different after being conditioned)
- Sampler (suspicious support for $A_{221} = 0$, which causes the $\ln B$ (221 vs 220) to be low for PyRing)

PyRing Data conditioning: PyRing.noise.loaddata(...)

1. Bandpassing to [20,2038] Hz:

```
if(kwargs['bandpassing']):
    # Bandpassing section.
    sys.stdout.write('Bandpassing the raw strain between [{}, {}] Hz.\n'.format(f_min_bp, f_max_bp))
    # Create a Butterworth bandpass filter between [f_min, f_max] and apply it with the function filtfilt.
    bb, ab = butter(4, [f_min_bp/(0.5*srate_dt), f_max_bp/(0.5*srate_dt)], btype='band')
    strain = filtfilt(bb, ab, rawstrain)
```

2. Compute the ACF as per definition and take the mean:

```
acfs = [acf(x, fft=fft_acf, simple_norm=kwargs['acf-simple-norm']) for x in chunks(times, strain,
    noise_seglen, avoid=triggertime, window=kwargs['window'], alpha=alpha_window)]

if(kwargs['noise-averaging-method']=='mean'):
    ACF = np.mean(np.array(acfs), axis=0)
elif(kwargs['noise-averaging-method']=='median'):
    # FIXME: This option gives rise to a junk spectrum. Currently not understood.
    review_warning()
    ACF = np.median(np.array(acfs), axis=0)
```

3. Compute the Covariance matrix from ACF and take the inverse:

```
# Restrict the ACF on the signal chunk and produce the covariance matrix from the ACF.
if(kwargs['truncate']):
    ACF_signal = ACF[:kwargs['analysis-duration-n']]
    np.savetxt(kwargs['output']+'/Noise/ACF_TD_cropped_{_}_{_}_{_}_{_}_{_}.txt'.format(ifo, int(starttime), int(T
    ), noise_chunk_size, srate, kwargs['analysis-duration']), ACF_signal)
else:
    ACF_signal = ACF[:signal_seglen]
    Covariance_matrix_signal = toeplitz(ACF_signal)
```



```

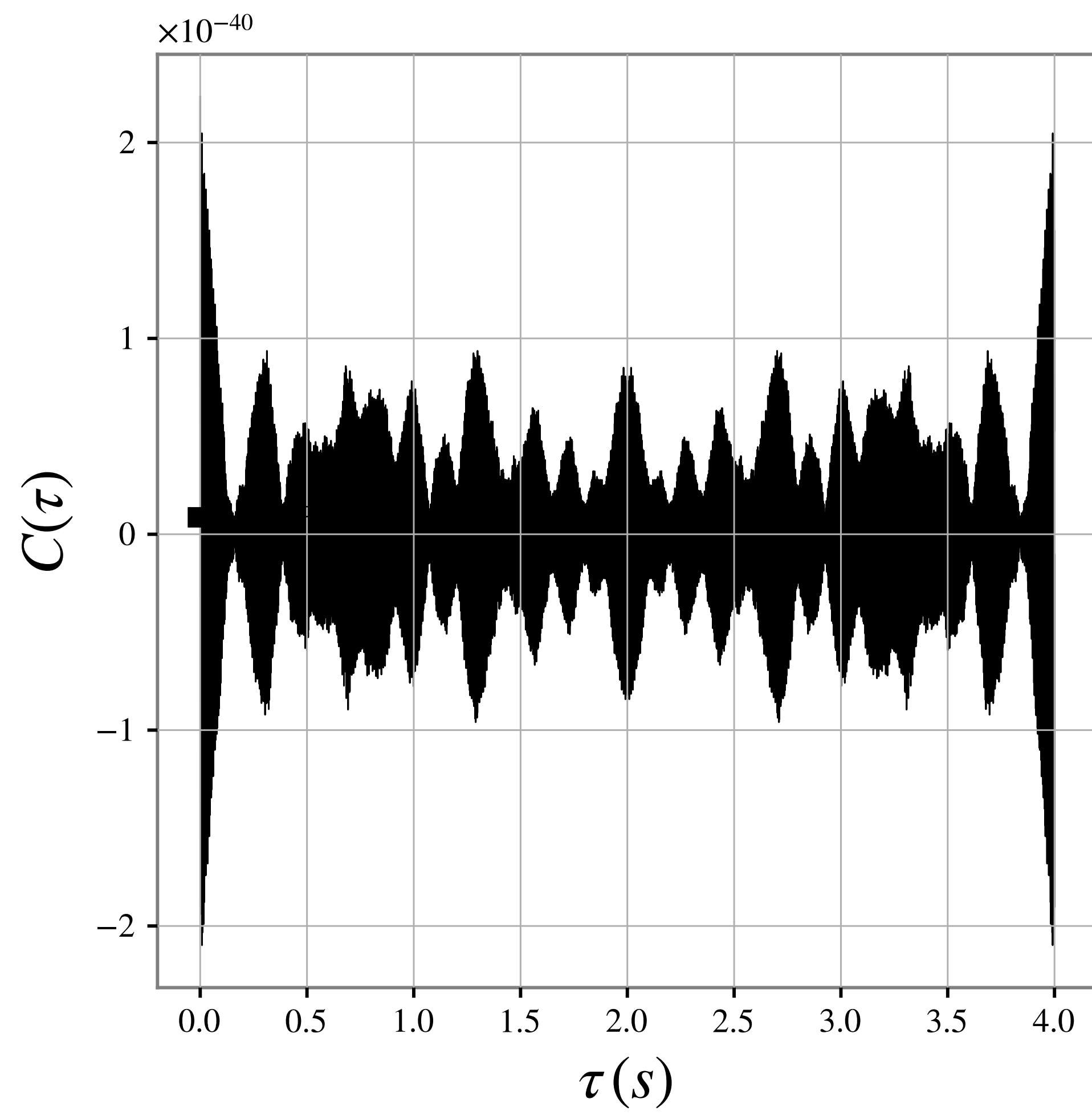
def acf(y, fft=True, simple_norm=False):
    """
    Returns the autocorrelation function:  $R[i] = \sum_n x[n] * x[n+i]$ 
    Returns an array spanned by the index i.
    If `fft`, uses FT method, see section 'Efficient computation' of 'https://en.wikipedia.org/wiki/Autocorrelation#cite\_note-3'
    """

    N = len(y)

    if fft:
        # ACF computation using FFT.
        Y = np.fft.fft(y)
        # We take the real part just to convert the complex output of fft to a real numpy float. The imaginary
        # part is already 0 when coming out of the fft.
        R = np.real(np.fft.ifft(Y * Y.conj()))
        # Divide by an additional factor of 1/N since we are taking two fft and one ifft without unitary
        # normalization, see: https://docs.scipy.org/doc/numpy/reference/routines.fft.html#module-numpy.fft
        acf_normed = R / N
    else:
        # ACF computation without FFT.
        acf_numpy = np.correlate(y, y, mode='full')[N-1:]
        # FIXME: describe the difference between the two normalizations.
        if simple_norm:
            acf_normed = acf_numpy[:N] / N
        else:
            acf_normed = acf_numpy[:N] / (N - np.arange(N))

    return acf_normed

```



Ringdown Data conditioning: Ringdown.Fit.condition_data(...)

To be studied, but they compute PSD and then inversely Fourier transformer the PSD to get ACF

PyStan is like a black box to me

PyCBC Data conditioning:.....

- My hypothesis: PyRing got suspicious support for $A_{221} = 0$ maybe because of combined effects from data conditioning and sampler.
- To justify this, the best way may be to introduce (copy-pasted) time-domain likelihood into PyCBC and run with different data conditioning and different samplers
- Other plans: Insert the ringdown ACF to PyRing