The SXS Catalog of Simulations

Mike Boyle

Outline

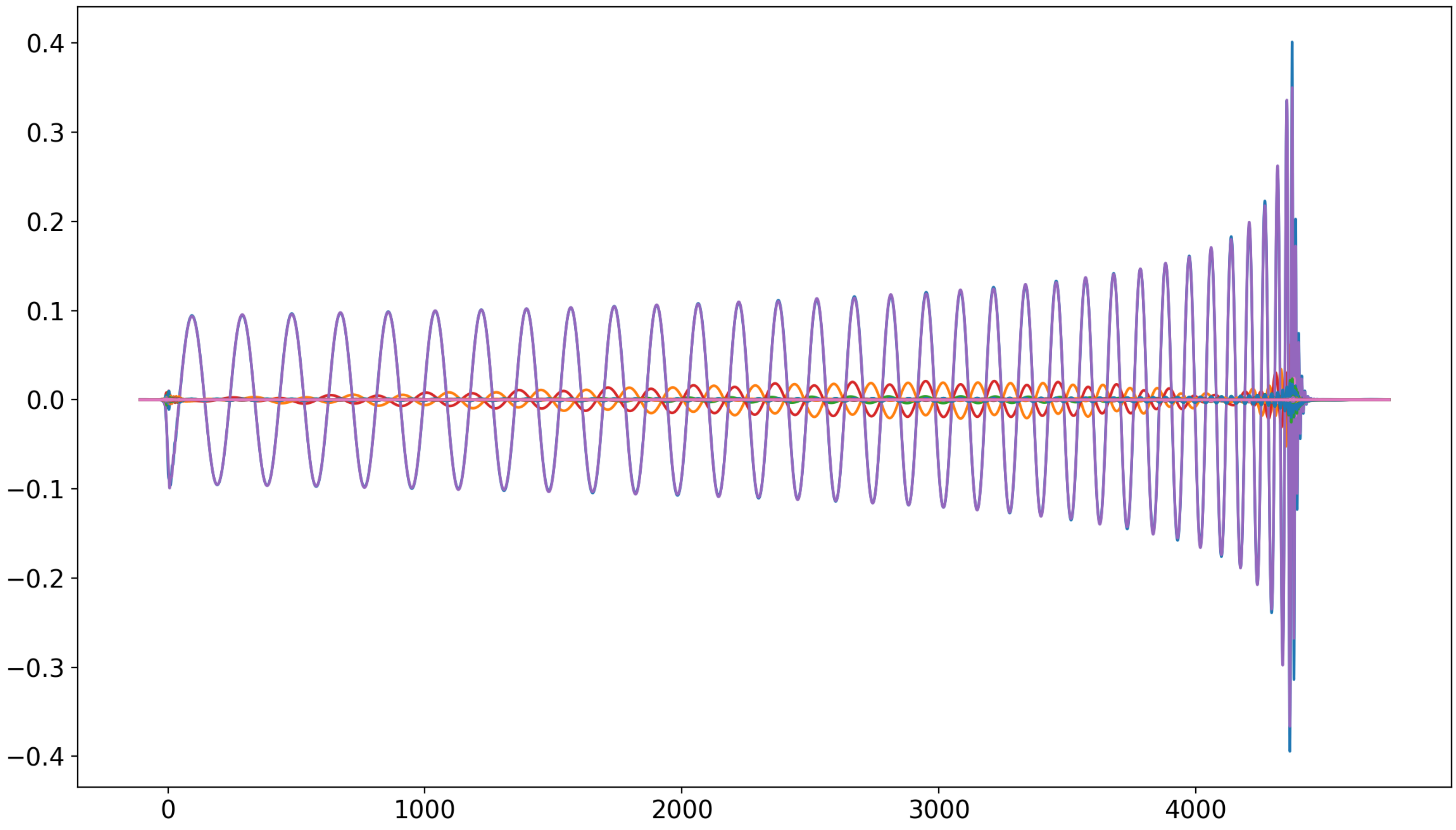
- Simulations
    - SpEC
    - The numbers
    - Post-processing
- The data
    - What's included
    - How to get it
    - Formats
    - Problems
    - Solutions
- Software
    - Backwards compatible
    - Future proof

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import sxs

waveform = sxs.load("SXS:BBH:0123/Lev/rhOverM", extrapolation_order=2, download=True)
plt.plot(waveform.t, waveform.data.real);
```

```
Skipping download from 'https://data.black-holes.org/catalog.json' because local file is newer
Found the following files to load from the SXS catalog:
    SXS:BBH:0123v4/Lev5/rhOverM_Asymptotic_GeometricUnits_CoM.h5
```



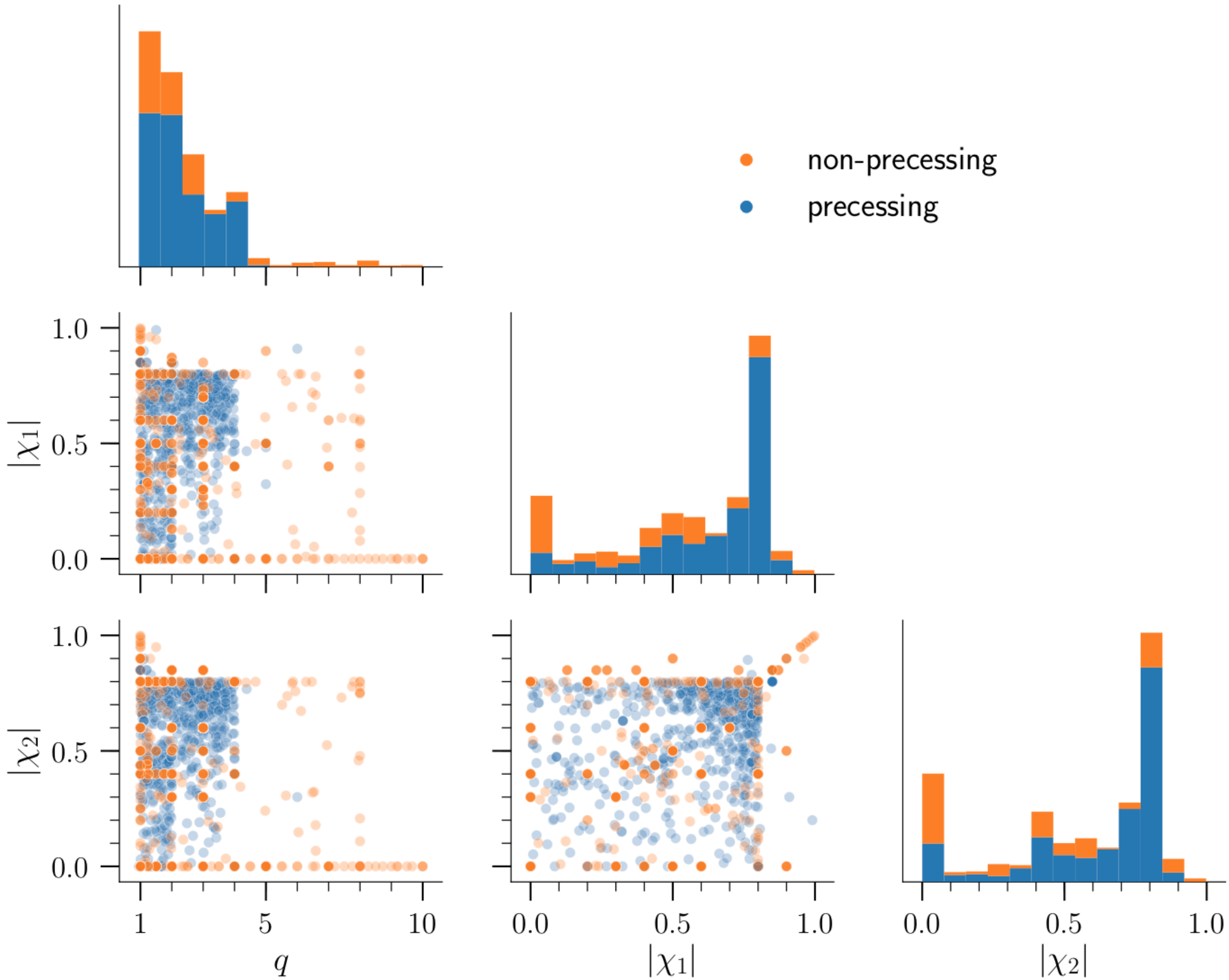
SpEC

- Initial data solves extended conformal thin sandwich equations
- Iterate to reduce eccentricity if desired
- Multi-domain spectral methods
- First-order generalized harmonic with constraint damping
- Adaptive mesh refinement
- Constraint-preserving boundary conditions

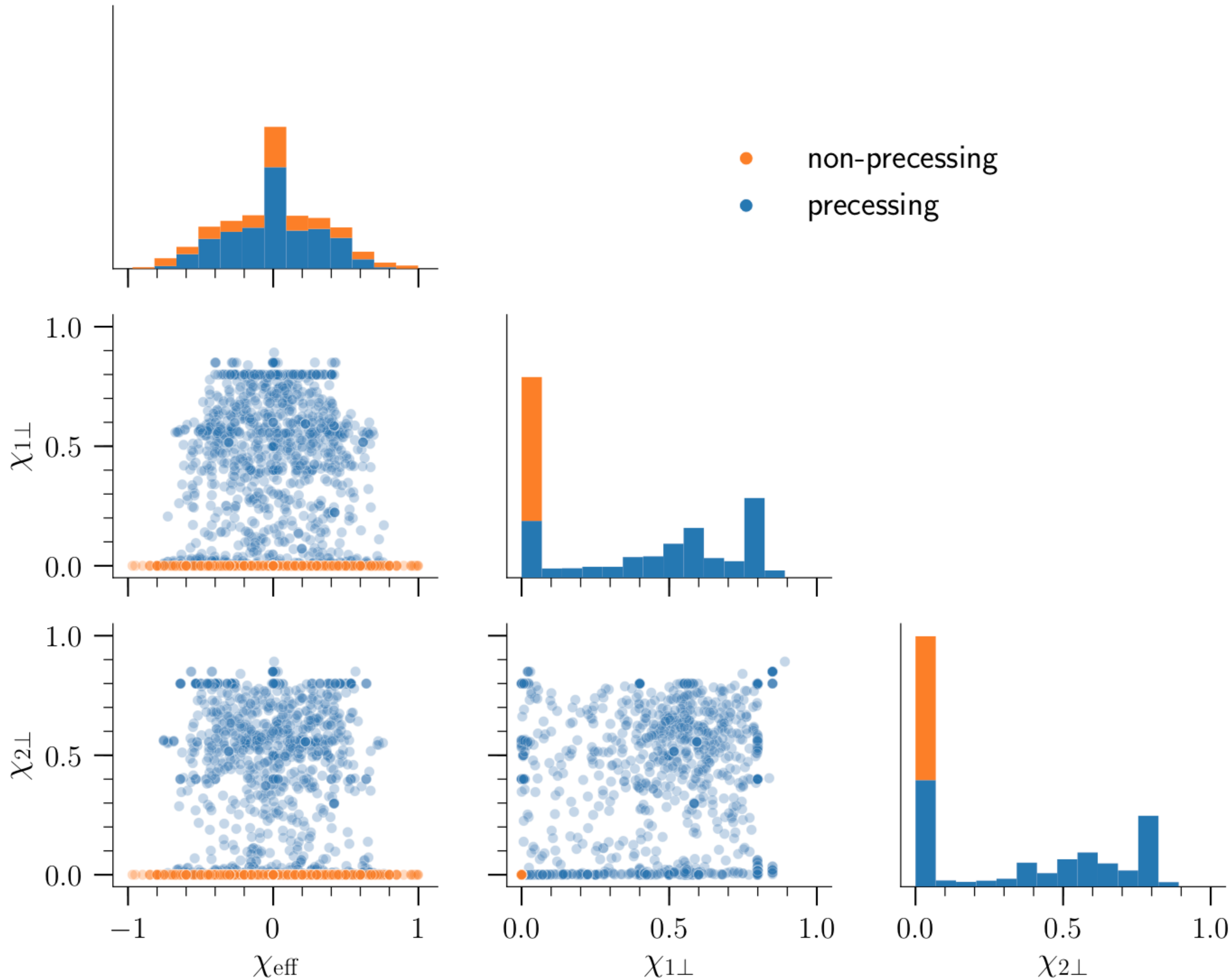
https://arxiv.org/abs/1904.04831 (https://arxiv.org/abs/1904.04831)

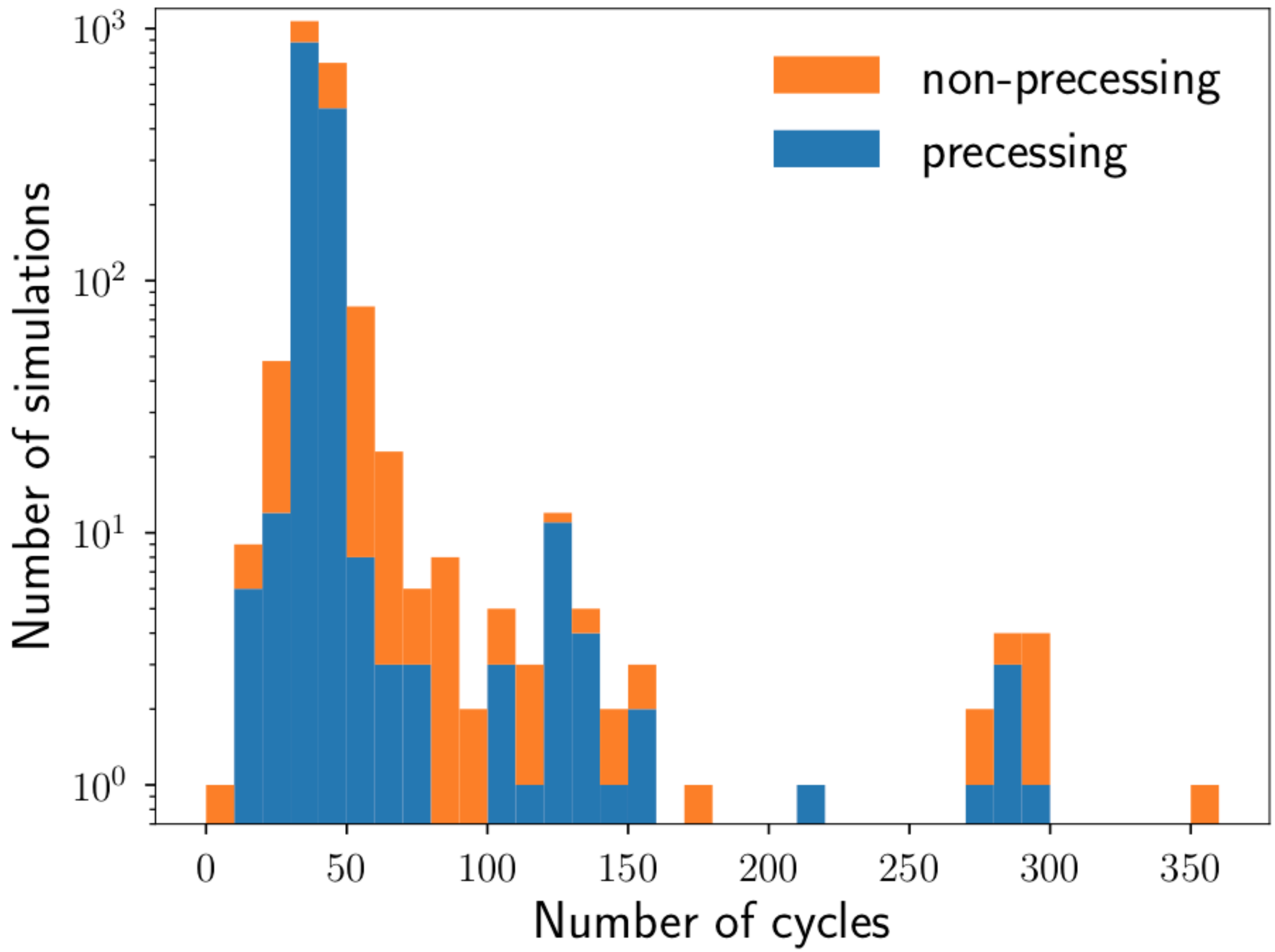
Simulations

- 2 NSNS
- 7 BHNS
- 2019 BBH
    - mass ratios between 1 and 10
    - spin magnitudes up to 0.998
    - 1426 precessing simulations
    - typically ~19 orbits

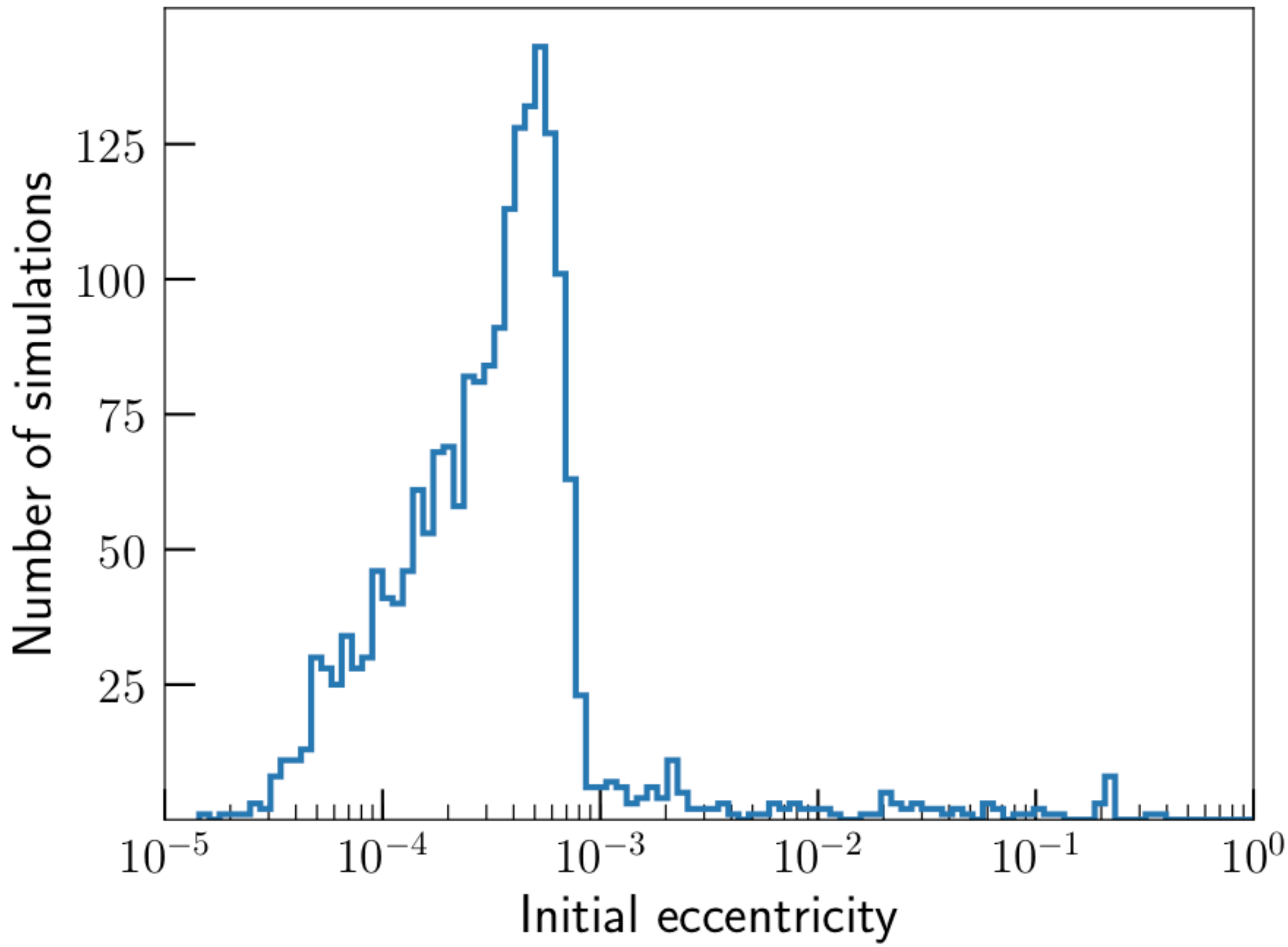


non-precessing
precessing
1.0
0.5
0.0
$|\chi_1|$
$|\chi_2|$
1
5
10
q
0.0
0.5
1.0
$|\chi_1|$
$|\chi_2|$


non-precessing
precessing
1.0
0.5
0.0
$\chi_{1\perp}$
$\chi_{2\perp}$
−1
0
1
$\chi_{\rm eff}$
0.0
0.5
1.0
$\chi_{1\perp}$
$\chi_{2\perp}$



non-precessing
precessing
Number of simulations
Number of cycles
10^3
10^2
10^1
10^0
0
50
100
150
200
250
300
350



Number of simulations
125
100
75
50
25
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
Initial eccentricity

Data extracted during the runs

- Horizon information
    - Coordinate trajectories
    - Areal (irreducible) mass
    - Christodoulou mass
    - Spins
- On a series of ~24 spheres between ~100M and outer boundary
    - RWZ h modes
    - Ψ_4 modes
    - Surface area
    - Average lapse
- (Data for CCE)

Post-processing

- (CCE, soon)
- Extrapolation of Ψ_4 and h
- Center-of-mass correction

Extrapolation

- Define corrected retarded time on each sphere

$$u_{i,j} = \int_0^{T_i} \sqrt{\frac{-1/g_j^{TT}(T)}{1 - 2M_{\mathrm{ADM}}/R_j(T)}}\, dT - R_j(T_i) - 2M_{\mathrm{ADM}} \ln\left[\frac{R_j(T_i)}{2M_{\mathrm{ADM}}} - 1\right],$$

- Fit the waveforms with a simple

$$f(u,r) \approx \sum_{j=0}^{N} \frac{f_{(j)}(u)}{r^j}$$

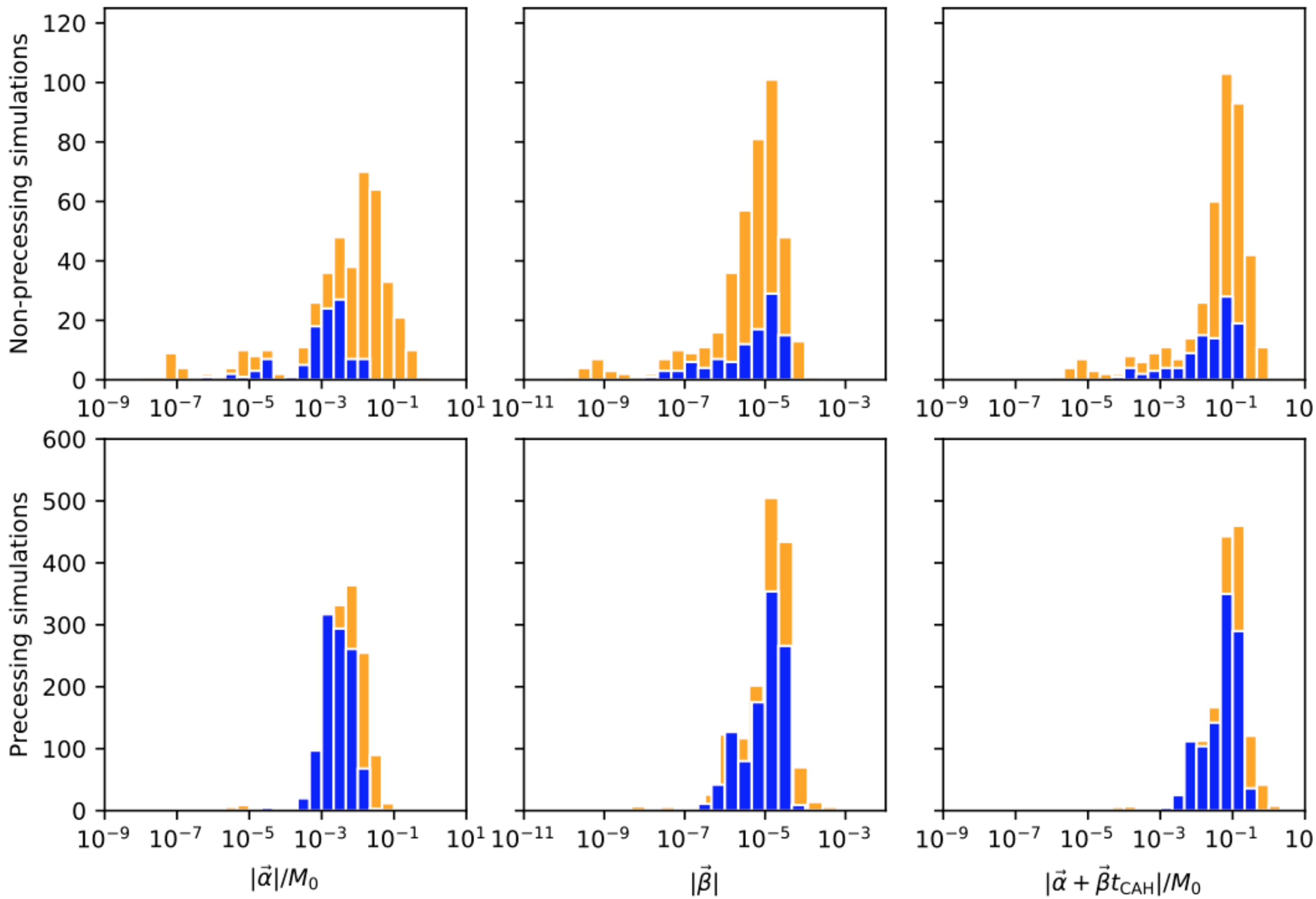
- Extrapolated waveform is $f_{(0)}(u)$

CoM correction

- "Gauge-invariant waveforms" 🙄
- Displacements around 0.1M show up as big wiggles
- General BMS transformations allowed
- We just use boost and translation
- Even worse, we just use the horizon trajectories

https://arxiv.org/abs/1509.00862 (https://arxiv.org/abs/1509.00862)

https://arxiv.org/abs/1904.04842 (https://arxiv.org/abs/1904.04842)



Non-precessing simulations
Precessing simulations
$|\vec{\alpha}|/M_0$
$|\vec{\beta}|$
$|\vec{\alpha}+\vec{\beta}t_{\rm CAH}|/M_0$

The problem

- Total catalog is ~16 TB
- Extrapolated waveform files currently average 95 MB each
- Users who want one waveform from each simulation need ~200 GB
- This is accelerating

The solutions

- Eliminate redundant time data — factor of ~1.45 reduction
- Reduce the number of files — factor of ~8 reduction
  - Eliminate finite-radius files
  - Eliminate non-CoM files
- Store each file more efficiently — factor of ~4 reduction

- Total catalog size falls to ~500 GB
- Typical user needs ~10 GB

Proposed spec

Requirements

- H5 file
  - All datasets stored as XOR-ed UINT64s
  - Attribute `format`
  - Dataset `time`
    - Shape `(n_times,)`
    - Must be strictly monotonic
  - Dataset `modes`
    - Shape `(n_times, 2*n_modes)`
    - Conjugate-pair representation
    - Attribute `ell_min`
    - Attribute `ell_max`
  - Dataset `log_frame` (*optional*)
    - Shape `(0, 3)`, `(1, 3)`, or `(n_times, 3)`
- JSON file
  - same file name with `.h5` → `.json`
  - more details on later slides

Recommendations

- Corotating frame
- Truncation at 10^{-10} of norm
- Add 0.0
- Compression options
- Choices of which files to distribute
- Directory structure and file names

Corotating frame

- Defined as rotating frame that minimizes time dependence of modes
- Compute angular velocity of this frame by solving
$$\langle h|L\, L\cdot\vec{\omega}|h\rangle = -\langle h|L\, \partial_t|h\rangle$$
- The integrate the angular velocity to find a frame
- Constant of integration → align z with dominant eigenvector of $\langle h|L\, L|h\rangle$

https://arxiv.org/abs/1302.2919 (https://arxiv.org/abs/1302.2919)

Conjugate-pair mode representation

- $h_{\ell,m}$ and $h_{\ell,-m}$ are at least somewhat redundant in corotating frame
- Define

$$s_{\ell,m} = \frac{h_{\ell,m} + \bar{h}_{\ell,-m}}{\sqrt{2}}$$

$$d_{\ell,m} = \frac{h_{\ell,m} - \bar{h}_{\ell,-m}}{\sqrt{2}}$$

- One of these will be zero (non-precessing systems) or small (otherwise), and thus compress well

- Store

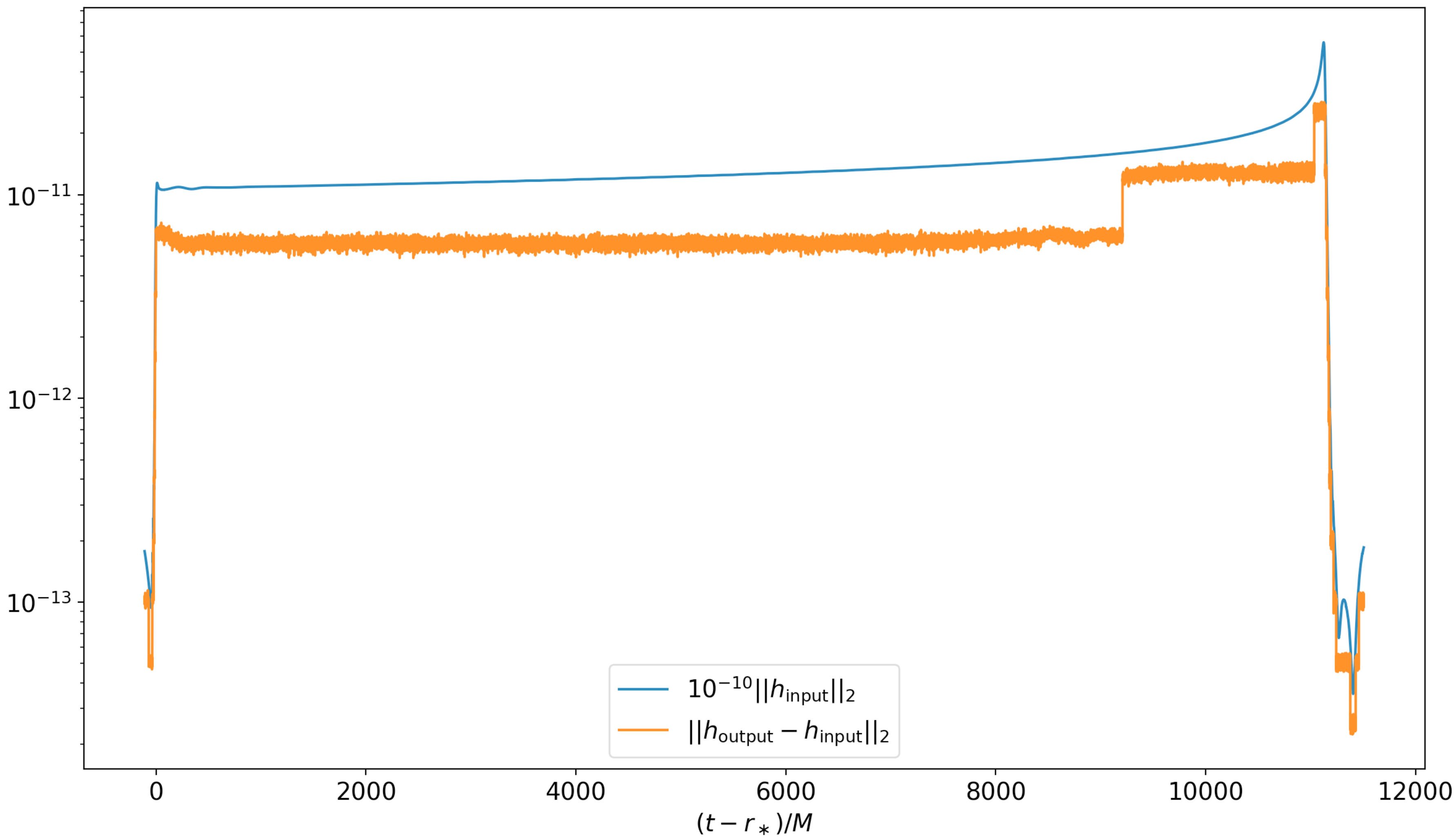
$[d_{2,2}, d_{2,1}, h_{2,0}, s_{2,1}, s_{2,2}, d_{3,3}, d_{3,2}, d_{3,1}, h_{3,0}, s_{3,1}, s_{3,2}, s_{3,3}, \ldots]$

Truncation

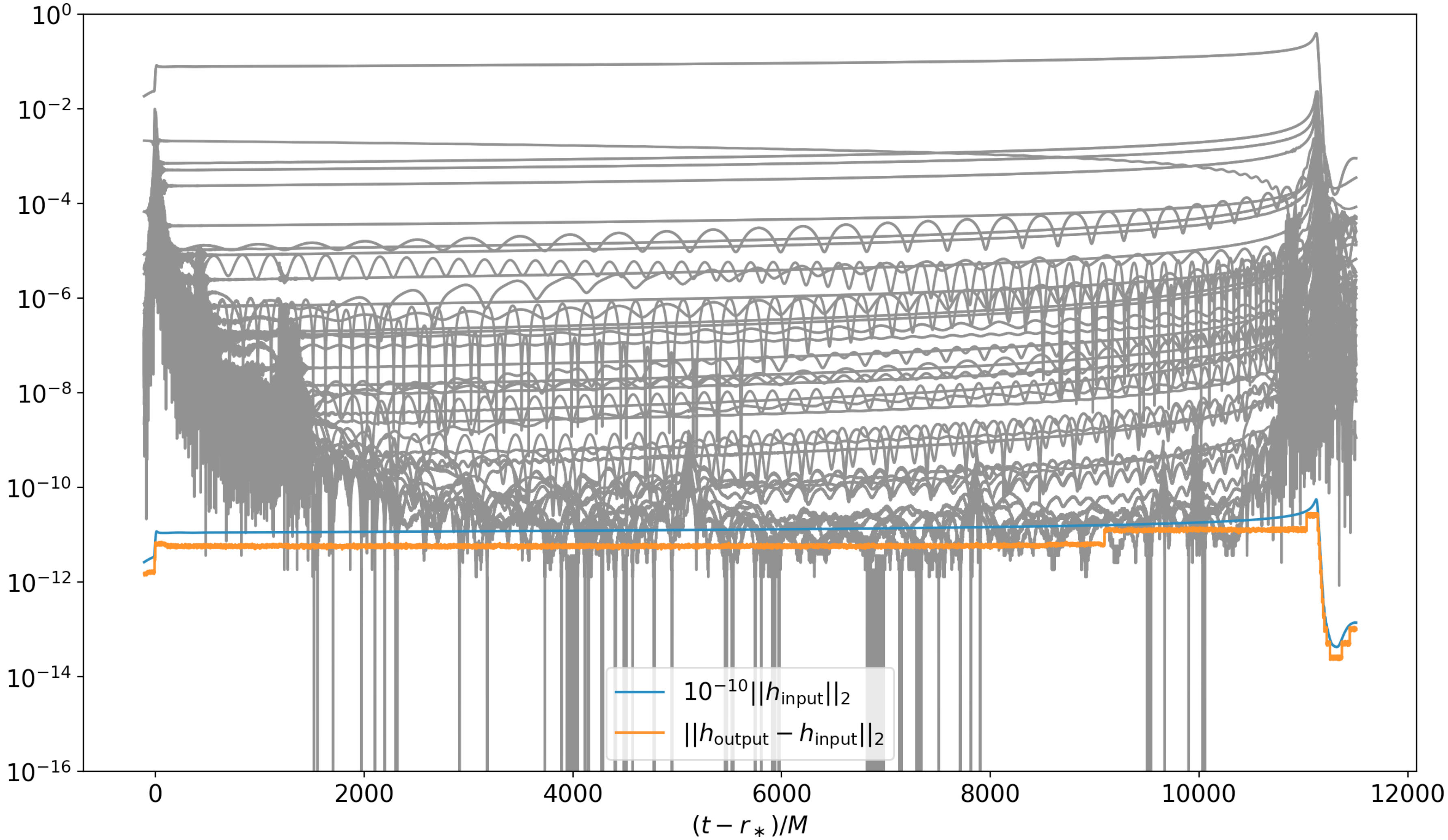
Zero out bits that affect norm at level below some chosen tolerance

- Only non-exact step
- Choose the *relative* tolerance `tol`
- Defines the *absolute* tolerance `tol * norm(data)` as function of time
- Drop bits at lower significance from conjugate-pair modes

```
def truncate(data, n_modes, tol=1e-10):
    tol /= sqrt(n_modes)  # Distribute ~evenly among all modes
    abs_tol = tol * norm(data, axis=1)
    power_of_2 = 2 ** (floor(-log2(abs_tol)))  # 1 / (largest power of 2 <= abs_tol)
    return round(data * power_of_2) / power_of_2
```



10^{-11}
10^{-12}
10^{-13}
0
2000
4000
6000
8000
10000
12000
$(t - r_*)/M$
$10^{-10}||h_{\mathrm{input}}||_2$
$||h_{\mathrm{output}} - h_{\mathrm{input}}||_2$


10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
0
2000
4000
6000
8000
10000
12000
$(t - r_*)/M$
$10^{-10}||h_{\text{input}}||_2$
$||h_{\text{output}} - h_{\text{input}}||_2$

Add 0.0

- Before truncation, many modes were small and noisy
- Signs were basically random
- After truncation, they are now ± 0.0
- The sign doesn't matter, but flips randomly, thus not compressing well
- Get rid of it by adding 0.0

XOR with previous time step

- Bitwise difference between data at successive times
- For ~smooth data, we are left with few bits changing
- This compresses *very* well
- It's trivial to do, and to undo:

Shuffle

- Bytes of similar significance more correlated than consecutive bytes
- Store bytes of given significance consecutively

In [5]:
```
for n in np.random.rand(8): print(f"{n.view(np.uint64):064b}\n")
```

```
0011111111010111011110101111000001100101001000010001000011110010

0011111111011000011100100100011101111111111010011101011010000000100

00111111110100000001001011010100110110110010011010100010100011000

0011111101010000011011000011000101001101110010111011110000000000

0011111111010110101000011110111001110101100110100001110001011000

0011111111101100110101111100001111110011101110110001111111111100

0011111111100100111001111100110110110101000110011111111010001011

00111111111000001110010011010101010100101101010000000001011100101
```

Format

- Corotating frame
- Paired modes
- Truncate
- XOR
- Multishuffle
- Bzip2

Are you scared yet?

Changing to a new format

- Pros:
  - Significant reduction in size
  - ~No cost to add insignificant modes
  - Easier for maintainers
  - Easier for users

- Cons:
  - Different is bad

Solve the problem with code

- `GWFrames` — old and ugly
- `scri` - powerful but complicated
- `sxs` - new and beautiful