What is Discovery?

Is it good? Yes.

Discovery in a nutshell

- Inference tool written in python for pulsar noise analysis and GW searches

- Challenge:
  - Strong covariance between noise processes and GWB
  - Need to search for processes simultaneously
- Identify noise processes in individual pulsars
- Marginalise (analytically or numerically) over nuisance parameters

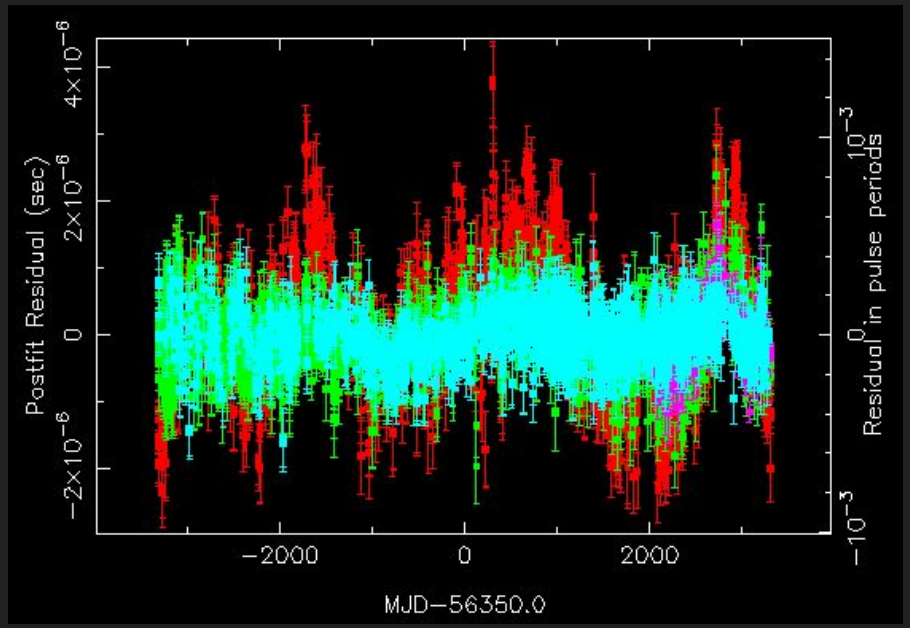
- Discovery is a new GPU-accelerated tool for nHz-frequency GW searches
  - Uses JAX in place of numpy functions, which automatically work on GPUs and enable differentiation with jax.grad

Noise modelling

- We're used to reading in .par and .tim files into Enterprise.
  - Now we write a compact .feather object that stores TOAs, frequencies, residuals, and timing model design matrix
- White noise
  - Uncorrelated in time
- Red noise
  - Time-correlated
  - Frequency-independent
- Dispersion measure variations
  - Time-correlated
  - 1/Frequency**2 scaling
- Other noise processes including
  - Exponential dip events
  - Scattering/chromatic/band noise
- Sample from the posterior probability distribution for these parameters

01_make_feathers.ipynb

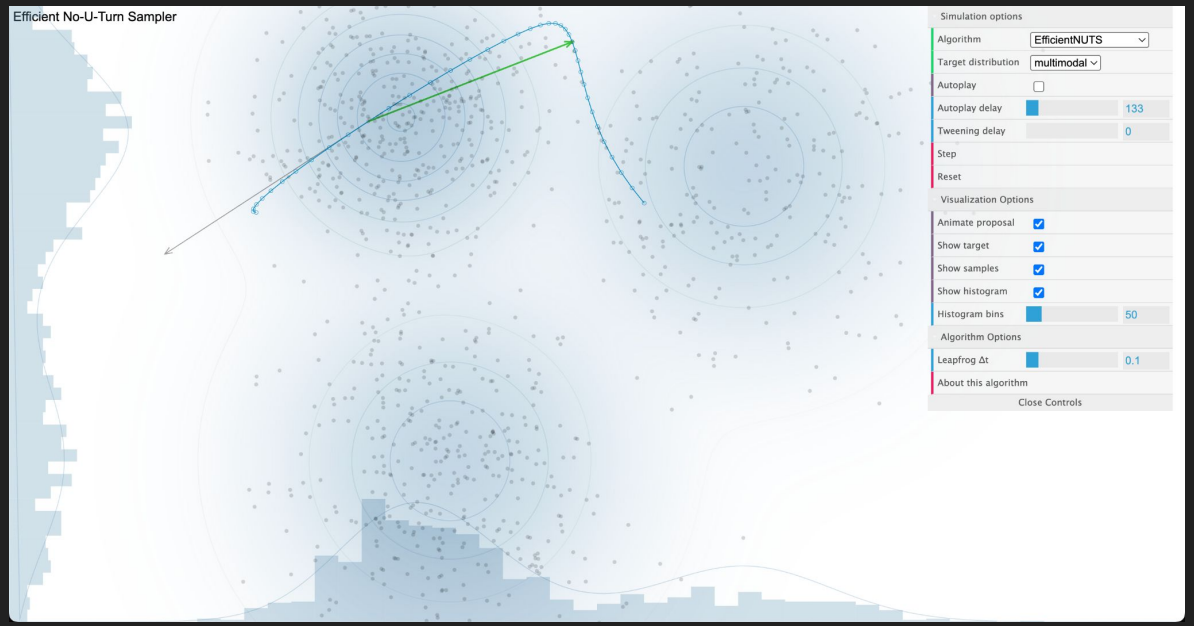
```
In [3]: from enterprise.pulsar import Pulsar

        outdir = "./my_feathers"
        os.makedirs(outdir, exist_ok=True)

        timfile = os.path.join(DATADIR, psrname + ".tim")
        psr_ent = Pulsar(parfile, timfile, ephem="DE440")
        feather_path = os.path.join(outdir, psrname + ".feather")
        psr_ent.to_feather(feather_path)
```



MCMC via Hamiltonian No-U-Turn Sampling (NUTS)

- Simulates a particle moving according to Hamiltonian dynamics
  - Requires derivatives
- At each step, the algorithm computes the position and momentum using the gradient of the log-probability
- No-U-Turn algorithm is an automatic tuner for simulation properties

Particle can travel a long way - the opposite of random-walk.



https://chi-feng.github.io/mcmc-demo/app.html?algorithm=EfficientNUTS&target=multimodal

Red Gaussian processes: Form of the PSD

- Need to choose a functional form
  - E.g. Powerlaw vs “broken-powerlaw” vs free spectrum
- Need to choose the number of Fourier frequencies (components) to model
  - We do this dynamically by selecting the highest-frequency we want to model to

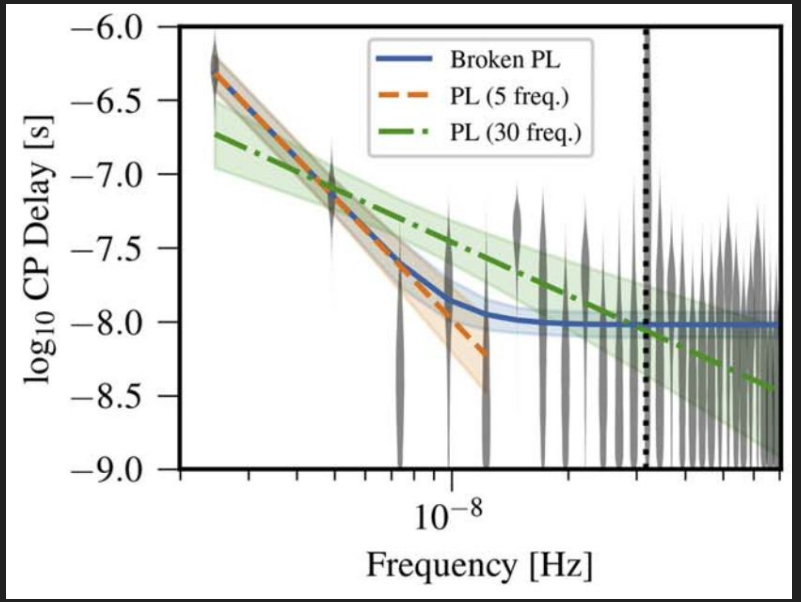


Fig. from NG 12.5 year analysis

Single-pulsar noise analysis

In Discovery!

Imports explained

- Signals.py contains:
  - white noise, red noise, timing model, chromatic noise models
  - PSD functions like powerlaw

- Likelihood contains… likelihood calculations
- Samplers available:
  - Numpyro (NUTS)
  - Jaxns (Nested sampler)

02_basic_noise_model.ipynb

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import jax, jax.random
import discovery as ds
import discovery.signals as signals
import discovery.likelihood as likelihood
import discovery.prior as prior
import discovery.samplers.numpyro as ds_numpyro

DATADIR = "./my_feathers/"
psrname = "J0613-0200"

psr = ds.Pulsar.read_feather(os.path.join(DATADIR, psrname + ".feather"))
print("Loaded", psr.name, "with", len(psr.toas), "TOAs")
```

```
Loaded J0613-0200 with 3067 TOAs
```

Build a model

- Define number of components for Fourier GPs
- Add model components to a list
  - timing model
  - White noise
  - Red noise
  - DM GP

```
In [2]: max_cadence_days = 60
        Tspan = signals.getspan(psr)
        components = int(Tspan / (max_cadence_days * 86400.0))
        print(f"Tspan = {Tspan/86400/365.25:.1f} yr -> {componer

        Tspan = 4.3 yr -> 26 Fourier frequency components
```

```
In [3]: model_components = []
        model_components.append(psr.residuals)                                    # data
        model_components.append(signals.makegp_timing(psr, svd=True))             # timir
        model_components.append(signals.makenoise_measurement(psr, tnequad=True))  # EFAC

        # Achromatic red (spin) noise, power-law Fourier GP
        model_components.append(
            signals.makegp_fourier(psr, signals.powerlaw, components=components, name="rec

        # DM noise: chromatic (freq^-2) power-law Fourier GP (freq^-2 basis)
        model_components.append(
            signals.makegp_fourier(psr, signals.powerlaw, components=components,
                                   fourierbasis=signals.dmfourierbasis, name="dm_gp"))

        model = likelihood.PulsarLikelihood(model_components)
```

Set up and run

- Pass the data and model list to the likelihood function
- Priors are set at a default value. We will change them next
- Set up NUTS sampler and sample

```
model = likelihood.PulsarLikelihood(model_components)

print("Free parameters in the model:")
for p in model.logL.params:
    print("   ", p)
```

```
Free parameters in the model:
    J0613-0200_KAT_MKBF_efac
    J0613-0200_KAT_MKBF_log10_tnequad
    J0613-0200_dm_gp_gamma
    J0613-0200_dm_gp_log10_A
    J0613-0200_red_noise_gamma
    J0613-0200_red_noise_log10_A
```

Set the priors

Every free parameter gets a flat (uniform) prior. We set the ranges

In [4]:
```
prior.priordict_standard.update({
    r"(.*_)?efac":                [0.5, 2.0],
    r"(.*_)?log10_tnequad":       [-10, -5],
    r"(.*_)?red_noise_log10_A.*": [-18, -11],
    r"(.*_)?red_noise_gamma.*":   [0, 7],
    r"(.*_)?dm_gp_log10_A":       [-18, -11],
    r"(.*_)?dm_gp_gamma":         [0, 7],
})
```

Sample the posterior with NUTS

`makemodel_transformed` reparametrizes the parameters to an unconstrained space for efficient Hamiltonian sampling; NUTS then explo
`num_warmup` tunes the sampler; `num_samples` are the kept draws. A real analysis would use more — these keep the workshop fast.

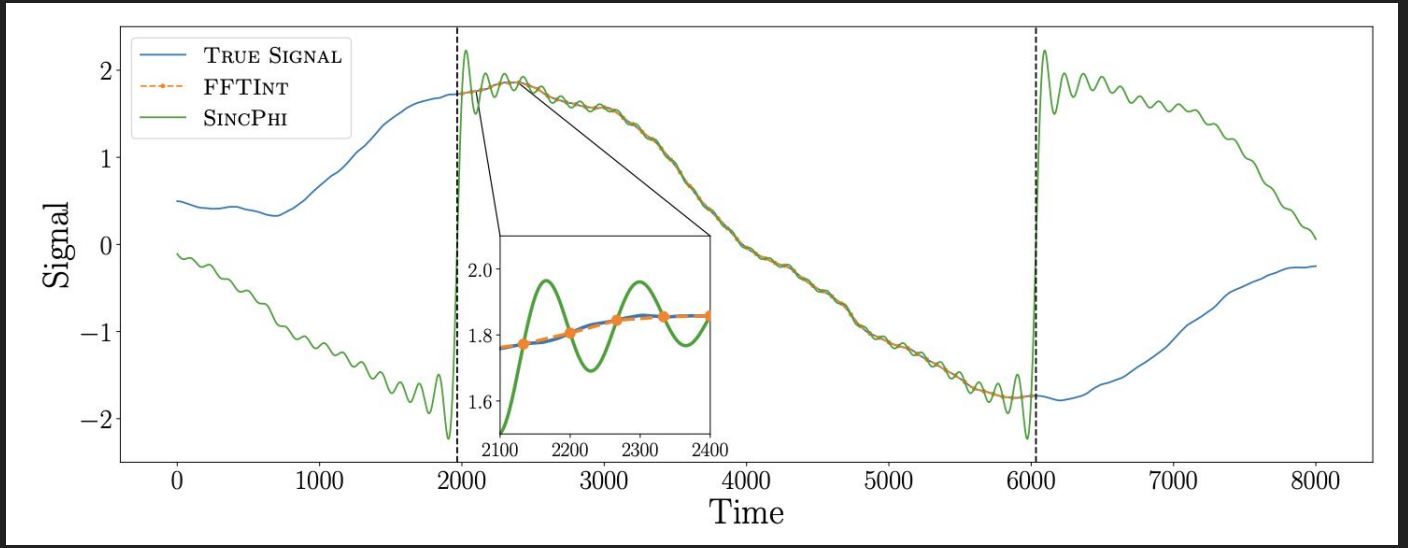
In [5]:
```
npmodel = ds_numpyro.makemodel_transformed(model.logL)
sampler = ds_numpyro.makesampler_nuts(npmodel, num_warmup=200, num_samples=500)

key = jax.random.PRNGKey(123456789)     # fixed seed -> reproducible chain
sampler.run(key)                        # the part that takes a minute or two
```

```
sample: 100%|██████████| 700/700 [01:22<00:00,  8.52it/s, 7 steps of size 3.63e-01. acc. prob=0.83]
```

Other links

- Bayesian inference for pulsar astronomers: https://docs.google.com/presentation/d/1uny3Lui8C-zButnhRxjqcmVxSLt-u1ZUXkfqnzh0rEU/edit?usp=sharing

FFTInt model



Crisostomi et al. (2025)

- Fourier-domain GP: Red processes are approximated using a finite number of sine/cosine frequency components. The basis functions are periodic and global across the whole dataset, so truncating the model shows up as ringing on the boundaries.
- FFTInt: Compute the covariance as a function of time lag on an evenly spaced grid (using an FFT-based method), and then interpolates that covariance to the actual uneven TOAs.