2 · A basic single-pulsar noise model in Discovery

In notebook 1 we made feather files. Now we **fit a noise model** to one pulsar.

A pulsar timing dataset is modelled as a sum of pieces:

$$\mathbf{r} = \underbrace{M\,\boldsymbol{\epsilon}}_{\text{timing model}} + \underbrace{n_{\text{white}}}_{\text{EFAC, EQUAD}} + \underbrace{n_{\text{red}}}_{\text{spin noise (achromatic)}} + \underbrace{n_{\text{DM}}}_{\text{dispersion measure } (\propto \nu^{-2})} + \ldots$$

In this **basic** model we include:

- **timing model** — marginalised analytically (a GP over the design matrix M),
- **white noise** — per-backend `EFAC` (scales the TOA errors) and `EQUAD` (added in quadrature),
- **red noise** — an achromatic power-law Gaussian process (spin noise),
- **DM noise** — a chromatic (ν^{-2}) power-law Gaussian process.

Each power-law GP has two parameters: amplitude `log10_A` and spectral index `gamma`.

We sample the posterior with the **NUTS** Hamiltonian sampler, **save the chain**, pull out the **maximum-likelihood** parameters, and make a **corner plot**.

We build the model with `discovery`'s `signals` tools and sample with NUTS via `numpyro`. We read the chain back with `sampler.to_df()`, compute the likelihood ourselves, and assemble the corner plot with the `corner` package

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
```

How big should the Gaussian processes be?

A power-law GP is represented by a set of Fourier frequencies (sine/cosine pairs). We choose how many from the data span and a maximum cadence: a longer cadence means coarser time resolution and therefore **fewer** components — faster to sample.

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 -> {components} Fourier frequ
```

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

Build the model

A Discovery model is just a **list of components** wrapped in a `PulsarLikelihood`. The first element is the data (residuals); the rest are the signals.

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

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

# 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_

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 explicitly so you can see exactly what is assumed.

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],
})
for p in model.logL.params:
    print(f"   {p:40s} {prior.getprior_uniform(p, {})}")
```

```
  J0613-0200_KAT_MKBF_efac                 [0.5, 2.0]
  J0613-0200_KAT_MKBF_log10_tnequad        [-8.5, -5]
  J0613-0200_dm_gp_gamma                   [0, 7]
  J0613-0200_dm_gp_log10_A                 [-18, -11]
  J0613-0200_red_noise_gamma               [0, 7]
  J0613-0200_red_noise_log10_A             [-18, -11]
```

Sample the posterior with NUTS

`makemodel_transformed` reparametrizes the parameters to an unconstrained space for efficient Hamiltonian sampling; NUTS then explores it. `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_sample

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

```
sample: 100%|██████████| 1500/1500 [11:58<00:00,  2.09it/s, 15 steps of si
ze 3.62e-01. acc. prob=0.81]
```

Save the chain

`sampler.to_df()` returns the posterior as a pandas `DataFrame` with one column per (physical) parameter. The sampler doesn't hand back the likelihood, so we evaluate `model.logL` ourselves at every sample (vectorised with `jax.vmap`) and store it as a `logl` column — we'll need it for the maximum-likelihood point below. Then we pickle the chain so we can reload it later without re-running the sampler.

In [8]:
```
chain = sampler.to_df()
chain = chain[[p for p in model.logL.params]].copy()      # keep just the
# evaluate the (marginalised) log-likelihood at every posterior sample
chain["logl"] = np.asarray(jax.lax.map(model.logL, {p: chain[p].values fo

outdir = "./results"
os.makedirs(outdir, exist_ok=True)
save_base = os.path.join(outdir, psrname + "_basic")
chain.to_pickle(save_base + ".pickle")
print("saved", save_base + ".pickle", " shape:", chain.shape)
chain.head()
```

```
saved ./results/J0613-0200_basic.pickle  shape: (1000, 7)
```

Out[8]:

	J0613-0200_KAT_MKBF_efac	J0613-0200_KAT_MKBF_log10_tnequad	J0613
0	0.996750	-6.657143	
1	1.038061	-6.659991	
2	1.009867	-6.673874	
3	0.987853	-6.635551	
4	0.987089	-6.497728	

Maximum-likelihood parameters

The "best fit" point estimate is the sample with the highest log-likelihood. We also save it to JSON — later analyses (e.g. a common-noise / GWB search) read these per-pulsar values as a fixed noise dictionary.

In [9]:
```
import json

ml_idx = chain["logl"].idxmax()
ml_params = chain.loc[ml_idx].to_dict()

with open(save_base + ".json", "w") as f:
    json.dump(ml_params, f, indent=2)

print("Maximum-likelihood parameters:")
for k, v in ml_params.items():
    if k == "logl":
        continue
    print(f"   {k:40s} = {v:+.3f}")
```

```
Maximum-likelihood parameters:
   J0613-0200_KAT_MKBF_efac                 = +1.006
   J0613-0200_KAT_MKBF_log10_tnequad        = -6.597
   J0613-0200_dm_gp_gamma                   = +2.876
   J0613-0200_dm_gp_log10_A                 = -13.365
   J0613-0200_red_noise_gamma               = +5.849
   J0613-0200_red_noise_log10_A             = -17.741
```

Corner plot

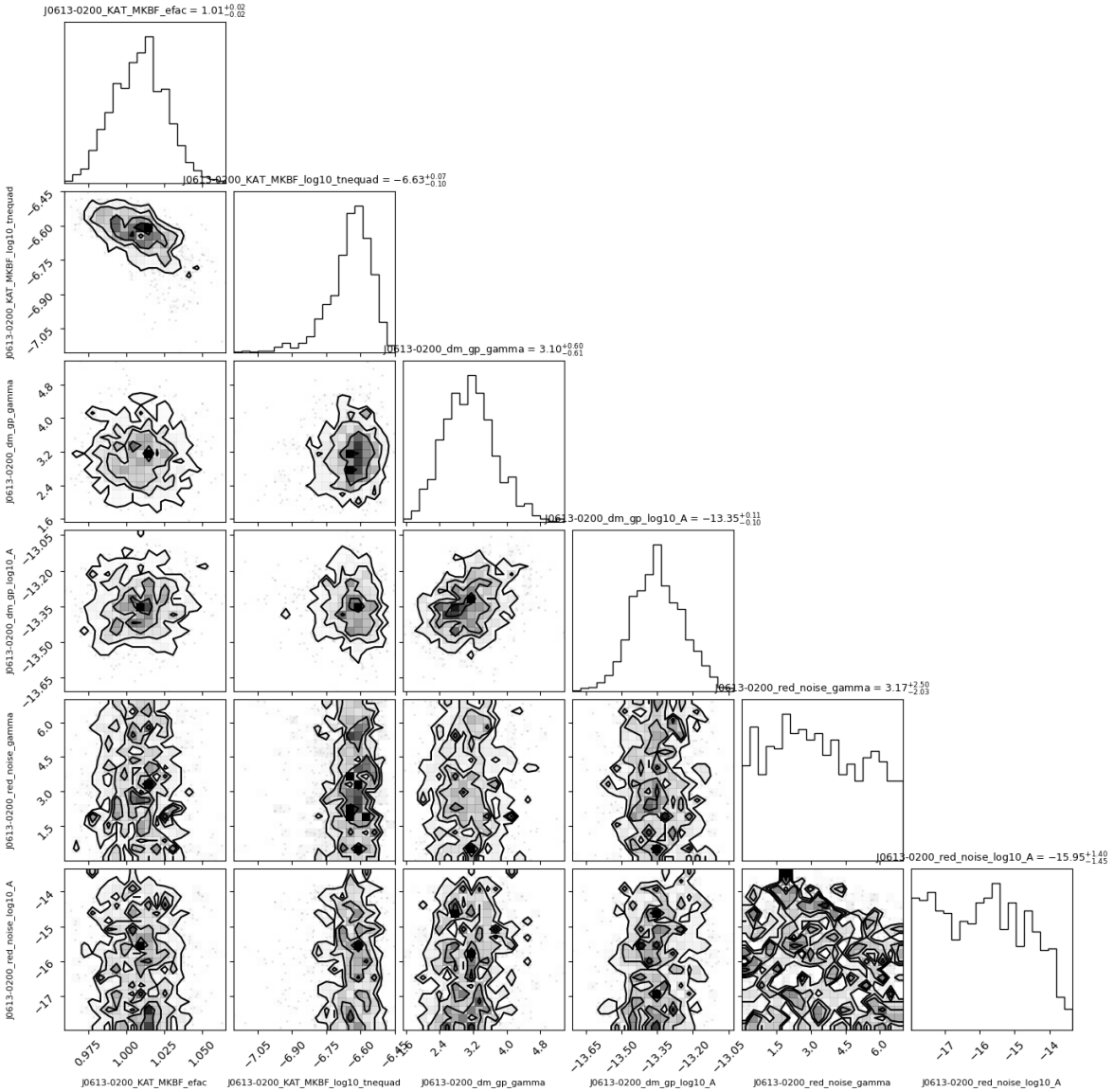
A corner plot shows the 1-D marginal posterior of each parameter (the histograms on the diagonal) and every 2-D joint posterior (the off-diagonal contours). It is the standard way to read off measured values and spot correlations. We build it by hand with the `corner` package.

In [10]:
```
import corner

labels = [c for c in chain.columns if c != "logl"]
fig = corner.corner(
    chain[labels].values,
    labels=labels,
    show_titles=True,
    title_fmt=".2f",
    title_kwargs={"fontsize": 9},
```

```
    label_kwargs={"fontsize": 8},
)
fig.savefig(save_base + "_corner.png", dpi=120, bbox_inches="tight")
plt.show()
```

```
/opt/build_psrsoft/workspace/micromamba/envs/IPTA_Env/lib/python3.11/site-
packages/astropy/config/paths.py:55: AstropyUserWarning: XDG_CONFIG_HOME i
s set to '/home/pulsar/.config', but the default location, /home/pulsar/.a
stropy/config, already exists, and takes precedence. This environment vari
able will be ignored.
  return set_temp_config._get_dir_path(rootname)
```



How to read it

- `red_noise_log10_A`, `red_noise_gamma` — amplitude and slope of the **spin noise**.
- `dm_gp_log10_A`, `dm_gp_gamma` — amplitude and slope of the **DM noise**.
- The `efac` / `log10_tnequad` columns are the per-backend **white-noise** parameters.

If a `log10_A` posterior is a clean peak well above the lower prior edge, that process is **detected**. If instead it piles up against the lower edge of the prior (an upper limit, no peak), the data don't require that process — it is **not**

significant. We explore exactly that in the next notebook.

➡️ **Next:** `03_custom_noise_model.ipynb` — choose your own pulsar and noise terms.