In [1]:
```
%matplotlib inline
```

4 · Bonus: hierarchical (population) noise inference

So far each pulsar has been analysed on its own. But the *pulsars as a population* share statistical properties — e.g. how spin-noise amplitudes and spectral indices are distributed across the array. **Hierarchical Bayesian inference** uses the per-pulsar noise chain made in notebooks 2 & 3 as input, and infers the **hyper-parameters** that describe the population.

The idea

For a single pulsar we measured a posterior on, say, the red-noise parameters $\theta = (\log_{10} A,\, \gamma)$. The hierarchical model says each pulsa itself drawn from a *population distribution* $p(\theta \,|\, \Lambda)$ governed by hyper-parameters Λ (means, spreads, …). Using every pulsar's chain once, we infer Λ:

$$\mathcal{L}(\{d_k\} \,|\, \Lambda) \;\propto\; \prod_{k=1}^{N_{\rm psr}} \frac{1}{n_k} \sum_{i=1}^{n_k} \frac{p(\theta_k^{(i)} \,|\, \Lambda)}{\pi(\theta_k^{(i)})}$$

i.e. for each pulsar we *recycle* its posterior samples $\theta_k^{(i)}$ and re-weight them by the population prior over the original sampling prior – standard "recycling" hyper-likelihood (as in `bilby.hyper`).

Two worked examples are provided for you

These are **bundled in `./bonus_hierarchical/`** (copied here so the workshop is fully self-contained — no `/fred` access needed). contains a ready-to-run `HyperParInf.ipynb`.

> **Requires Bilby.** The hierarchical step uses `bilby.hyper.HyperparameterLikelihood` and the `dynesty` sampler. Install with `pip install bilby dynesty` if they are not already available.

example	dataset	infers population hyper-parameters for
example_1	simulated MPTA realisation 1	red, DM, chromatic, ECORR

example	dataset	infers population hyper-parameters for
example_2	simulated MPTA realisation 2	red, DM, chromatic, ECORR

Each example ships the per-pulsar single-pulsar chains it needs (`chains/HamSampler/<psr>_chain/*.pickle`), the nested-sampli evidences (`chains/NestSampler/<psr>_chain/*.json`), and the injected truth (`Noise_par/`). The `results_*/` output folders a created when you run the notebook. The original notebooks' final **GWB-search** section (which needs ~220 MB of raw feathers) has trimmed from these bundled copies to keep them light.

Both compare the recovered population against the **injected** values stored in `Noise_par/` (these are simulated datasets, so we kn truth).

How to run them

These are standalone notebooks — open them directly (paths inside are already relative, so just launch from the example folder):

```
cd ./bonus_hierarchical/example_2
jupyter notebook HyperParInf.ipynb        # red-noise-only population (start here)

cd ./bonus_hierarchical/example_1
jupyter notebook HyperParInf.ipynb        # full red+DM+chrom+ECORR population
```

The cells below (a) check that the example folders and their inputs are in place, and (b) print the outline of the example so you kno to expect. Start with **example_2** (simpler and faster), then move on to **example_1**.

In [2]:
```
import os, glob

HW = "./bonus_hierarchical"
for ex in ["example_1", "example_2"]:
    d = os.path.join(HW, ex)
    nb = os.path.join(d, "HyperParInf.ipynb")
    npar = len(glob.glob(os.path.join(d, "Noise_par", "*.txt")))
    nchain = len(glob.glob(os.path.join(d, "chains", "HamSampler", "*_chain")))
    print(f"{ex}:")
    print(f"   notebook   : {'FOUND' if os.path.exists(nb) else 'MISSING'}  {nb}")
    print(f"   injections : {npar} pulsars in Noise_par/")
```

```
    print(f"   chains     : {nchain} per-pulsar HamSampler chains bundled")
    print()
```

```
example_1:
   notebook   : FOUND  ./bonus_hierarchical/example_1/HyperParInf.ipynb
   injections : 80 pulsars in Noise_par/
   chains     : 80 per-pulsar HamSampler chains bundled

example_2:
   notebook   : FOUND  ./bonus_hierarchical/example_2/HyperParInf.ipynb
   injections : 80 pulsars in Noise_par/
   chains     : 80 per-pulsar HamSampler chains bundled
```

Peek at what the example notebook does

`HyperParInf.ipynb` (1) reads the single-pulsar chains with the helpers in `hyper_utilis.py`, (2) reads the injected hyper-parame ranges from `Noise_par/`, (3) builds a `bilby` `HyperparameterLikelihood`, and (4) samples the population hyper-parameters an corner-plots them against the injected truth.

The cell below prints the section headings of the example_2 notebook.

In [3]:
```
import json

nb_path = os.path.join("./bonus_hierarchical", "example_2", "HyperParInf.ipynb")
nb = json.load(open(nb_path))
print("Outline of", nb_path, "\n")
for i, c in enumerate(nb["cells"]):
    if c["cell_type"] == "markdown":
        for line in c["source"]:
            s = line.strip()
            if s.startswith("#"):
                print(f"  cell {i:2d}:  {s}")
                break
```

```
Outline of ./bonus_hierarchical/example_2/HyperParInf.ipynb

  cell  0:  # Code to infer pulsar noise population properties
  cell  3:  ## First we need to read single pulsar noise chains
  cell  7:  # Hierarchical Bayesian Inference for Pulsar Populations Properties
  cell  8:  # Red noise hyperparameters
  cell 17:  # DM Noise
  cell 22:  # Chromatic Noise
  cell 27:  # ECORR
  cell 31:  ## (Trimmed) Optional extension: GWB search
```

Optionally: run an example inline with papermill

`papermill` executes a notebook top-to-bottom and writes an executed copy, so you can run example_2 without leaving this noteb
(Skip if `papermill` isn't installed — just open the notebook directly as shown above.)

> The hierarchical step can take a while; it samples a full population posterior.

In [4]:
```
# Uncomment to run example_2 end-to-end and save an executed copy here.
#
# import papermill as pm
# src = os.path.join("./bonus_hierarchical", "example_2", "HyperParInf.ipynb")
# pm.execute_notebook(src, "./example_2_executed.ipynb", cwd=os.path.dirname(src))
# print("done -> ./example_2_executed.ipynb")
print("Open the notebook directly (recommended), or uncomment the papermill call above.")
```

```
Open the notebook directly (recommended), or uncomment the papermill call above.
```

What to look for

- In **example_2**, does the inferred red-noise population (the corner plot of hyper-parameters) bracket the injected red-noise distri drawn from `Noise_par/`?
- In **example_1**, the same question for DM, chromatic and ECORR populations — and notice how combining ~80 pulsars constrai population far better than any single pulsar could.
- Tie it back: the per-pulsar chains feeding these notebooks are exactly the kind you produced in notebooks 2 and 3.

That's the workshop. You've gone from raw `.par` / `.tim` → feather → single-pulsar noise model → your own model selection → population-level inference.