Tutorial: Bilby Bayesian Inference Library in Python for Gravitational-Wave Data Analysis

G. Cella

(Remote, CET)

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or of the European Research Executive Agency (REA). Neither the European Union nor the granting authority can be held responsible for them. The ACME project has received funding from the European Union's Horizon Europe Research and Innovation programme under Grant Agreement No 101131928.

Contents

SECTION 1

Introduction

1.1 Audience and Goals

This tutorial is designed for beginners who want a practical introduction to Bayesian inference and to the `bilby` library in Python, with a focus on gravitational-wave (GW) parameter estimation. The day mixes a short introductory lecture with guided hands-on labs.

1.2 Plans

09:00 – 09:30	Welcome, Setup Check, and Introduction to the Tutorial
09:30 – 10:30	Lecture: Bayesian Inference and Bilby
10:30 – 10:45	Break
10:45 – 12:00	Hands-On Lab 1: Basic Parameter Estimation with Bilby (Linear Model)
12:00 – 13:30	Lunch Break
13:30 – 15:00	Hands-On Lab 2
15:00 – 15:15	Break
15:15 – 17:30	Hands-On Lab 3: Simulated Gravitational-Wave Signal Parameter Estimation
17:30 – 18:00	Wrap-Up, Q&A, and Further Resources

SECTION 2

Prerequisites and Setup (Pre-Tutorial)

To ensure a smooth hands-on experience, participants are required to complete the software setup *before* the tutorial. We strongly recommend using `uv`, a fast and reliable Python package manager developed by Astral, which simplifies environment creation and dependency resolution.

2.1 Install `uv`

If `uv` is not already installed, install it via:

```
$ curl -Ls https://astral.sh/uv/install.sh | sh
```

or, on macOS using Homebrew:

```
$ brew install uv
```

Verify installation:

```
uv --version
```

2.2 Create an Isolated Python Environment

We recommend Python 3.11 (fully compatible with Bilby and LALSuite).

Create a dedicated virtual environment:

```
$ uv venv bilby-tutorial --python 3.11
```

Activate it:

```
$ source bilby-tutorial/bin/activate     # macOS/Linux
$ bilby-tutorial\Scripts\activate        # Windows
```

After activation, your shell prompt should indicate that you are inside the environment.

2.3 Install Core Packages

Install Bilby and required scientific dependencies:

```
$ uv pip install bilby jupyterlab numpy scipy matplotlib pandas
```

This installs:

- `bilby` (Bayesian inference library),
- `dynesty` (default nested sampler),
- standard scientific Python stack.

2.4 Install Gravitational-Wave Dependencies

For the afternoon gravitational-wave lab, install:

```
$ uv pip install gwpy lalsuite
```

These provide:

- detector strain data handling (`gwpy`),
- waveform generation via LALSimulation (`lalsuite`).

Note: Installation of `lalsuite` may take several minutes and may require system build tools on Linux systems.

2.5 Verify Installation

Run the following quick test:

```
$ python -c "import bilby; print(bilby.__version__)"
$ python -c "import bilby.gw; import gwpy; print('GW stack OK')"
```

If no errors appear, the installation is successful.

2.6 Launch JupyterLab

Start JupyterLab inside the activated environment:

```
$ jupyter lab
```

A browser window should open automatically. Keep this environment active during the tutorial.

2.7 Test Access to GWOSC (Open Gravitational-Wave Data)

For optional advanced exercises using public LIGO/Virgo data, we recommend verifying that your system can access the Gravitational-Wave Open Science Center (GWOSC).

Step 1: Test basic internet access from Python Run:

```
$ python -c "import requests;
print(requests.get('https://www.gw-openscience.org').status_code)"
```

A successful response should return:

```
$ 200
```

If this fails, check firewall or proxy settings.

Step 2: Test data download using `gwpy` Run the following short script to download a few seconds of open strain data around the GW150914 event:

```
from gwpy.timeseries import TimeSeries
data = TimeSeries.fetch_open_data("H1", 1126259446, 1126259450)
print(data)
```

This attempts to download 4 seconds of strain data from the LIGO Hanford detector. If successful, a summary of the downloaded time series will be printed.

Step 3: Minimal Bilby compatibility test To ensure that the full GW stack works, test:

```
import bilby
import bilby.gw
print("Bilby GW module loaded successfully")
```

If all steps complete without errors, your system is ready for the gravitational-wave lab.

Common issues:

- Corporate or institutional firewalls blocking HTTPS downloads.
- Missing SSL certificates on older Linux systems.
- Slow network connections causing timeouts.

If problems persist, please contact the organizers before the tutorial.

8. Remote Participation Requirements

- Stable internet connection.
- Zoom installed and tested.
- Working microphone (camera optional).
- Ability to share screen if troubleshooting assistance is required.

Important: Please complete all installation steps before the tutorial. Troubleshooting time during the live session will be limited.

SECTION 3

Lecture (09:30–10:30): Bayesian Inference and GW Context

Format

Slides + live discussion. Questions via chat/voice.

3.1 Core Bayesian Concepts

- **Bayes' theorem and terminology:**

$$p(\theta \mid d) \propto \pi(\theta)\, \mathcal{L}(d \mid \theta),$$

  where θ are the model parameters, d is the observed data, $\pi(\theta)$ is the prior encoding our assumptions about θ before seeing data, $\mathcal{L}(d \mid \theta)$ is the likelihood expressing how well the model explains the data, and $p(\theta \mid d)$ is the posterior, the final inference combining both. This formula is the foundation of Bayesian inference.

- **Evidence and model comparison:** The evidence (or marginal likelihood) is the integral of the likelihood times the prior across all parameters:

$$\mathcal{Z} = \int \pi(\theta)\, \mathcal{L}(d \mid \theta)\, d\theta.$$

  While not needed for parameter estimation, it's essential for comparing models via the Bayes factor. Nested sampling methods compute $\mathcal{Z}$ naturally, which is one reason they're widely used in astrophysics.

- **Posterior summaries:** After sampling, we analyze the posterior to extract meaningful summaries:
  - *Credible intervals* indicate the range of values containing a specified probability mass (e.g., 90%).
  - *Correlations* between parameters reveal model degeneracies.
  - Unlike frequentist methods, Bayesian analysis gives us full distributions, allowing for richer and more informative inferences than point estimates alone.

- **Computational methods:** Since most posteriors are intractable analytically, we use:
  - *Markov Chain Monte Carlo (MCMC)*: builds a chain of samples guided by the posterior shape.
  - *Nested sampling*: explores parameter space by shrinking prior volumes and is particularly effective for multimodal posteriors and computing evidence.

  The goal is to provide intuitive understanding, not detailed algorithmic derivations.

- **Where Bilby fits:** Bilby abstracts the inference workflow by clearly separating:
  - The *problem definition*—specifying priors and likelihoods relevant to the scientific question.
  - The *sampler backend*—e.g., 'dynesty', 'emcee', or 'ultranest'—which is interchangeable with minimal code changes.

  This modular design allows users to focus on modeling while leveraging powerful inference tools under the hood.

GW background and why inference is needed

- GW detectors measure strain data dominated by noise; inference extracts source parameters with uncertainties.
- Parameters in compact binary coalescences:
  - Intrinsic: masses, spins (and tides for neutron stars).
  - Extrinsic: sky location, distance, inclination, polarization, phase, coalescence time.
- Typical dimensionality is high (∼15+ parameters), motivating robust samplers and careful priors.
- Practical note: full analyses can take many hours; in a live tutorial we reduce dimensionality.

SECTION 4

Laboratory 1 (10:45–12:00): Basic Bilby Workflow

Learning objectives

- Simulate synthetic data for a simple model.
- Define a likelihood and priors in Bilby.
- Run a sampler (e.g. `dynesty`) and interpret posterior samples.
- Produce and read diagnostic plots (corner plots).

Notebook outline

1. **Problem:** $y(t) = mt + c + \epsilon$ with Gaussian noise.
2. **Data simulation:** choose $(m_{\text{true}}, c_{\text{true}})$, generate (t, y), add noise, plot.
3. **Model function:** `linear_model(t, m, c)`.
4. **Likelihood:** use Bilby Gaussian likelihood (or briefly show custom likelihood structure).
5. **Priors:** uniform priors on m and c (wide enough to include true values).
6. **Sampling:** run `bilby.run_sampler(..., sampler='dynesty')`.
7. **Results:** `result.plot_corner()`, credible intervals, compare with injected truth, optional best-fit overlay.
8. **Discussion:** effect of prior choices, evidence value, reproducibility and outputs.

4.1 Example 1: Linear Regression

First we include some modules that will be needed

```
import numpy as np
import bilby
from bilby.core.utils import random
```

Data simulation We model data as $y = ax + b + \epsilon$ with Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma)$.

```
# ----------------------------------------------------------
# 1) Simulate data (linear model with Gaussian noise)
# ----------------------------------------------------------
np.random.seed(123)
random.seed(123)

n = 50
x = np.linspace(0.0, 10.0, n)
m_true = 2.0
b_true = -1.0
sigma_true = 0.8
y = m_true * x + b_true + np.random.normal(0.0, sigma_true, size=n)
```

Likelihood: The next step is to define a likelihood Bilby object

```
# ---------------------------
# 2) Define a Bilby likelihood
# ---------------------------
class LinearRegressionLikelihood(bilby.Likelihood):

    def __init__(self, x, y):
        super().__init__(parameters={"m": None, "b": None, "sigma": None})
        self.x = np.asarray(x)
        self.y = np.asarray(y)

    def log_likelihood(self, parameters=None):
```

```
        if parameters is not None:
            self.parameters.update(parameters)

        m = self.parameters["m"]
        b = self.parameters["b"]
        sigma = self.parameters["sigma"]

        model = m * self.x + b
        resid = self.y - model

        return -0.5 * np.sum(
            (resid / sigma) ** 2 + np.log(2.0 * np.pi * sigma**2)
        )

likelihood = LinearRegressionLikelihood(x=x, y=y)
```

Priors: We define uniform priors on a and b:

```
# ----------------------------
# 3) Define priors
# ----------------------------
priors = bilby.core.prior.PriorDict()
priors["m"] = bilby.core.prior.Uniform(
    minimum=-5,
    maximum=5,
    name="m", latex_label="$m$")
priors["b"] = bilby.core.prior.Uniform(
    minimum=-10,
    maximum=10,
    name="b", latex_label="$b$")
priors["sigma"] = bilby.core.prior.LogUniform(
    minimum=1e-3,
    maximum=10,
    name="sigma", latex_label=r"$\sigma$")
```

Sampling: We run ‘dynesty‘ to sample the posterior.

```
# ----------------------------
# 4) Run inference
# ----------------------------
outdir = "outdir"
label = "linear_regression"
bilby.core.utils.setup_logger(outdir=outdir, label=label)
result_dynesty = bilby.run_sampler(
    likelihood=likelihood,
    priors=priors,
    sampler="dynesty",
    nlive=500,
    outdir=outdir + "_dinesty",
    label=label,
    plot=True,
    seed=1234,
)
```

Posterior Analysis: The result object contains samples, evidence, and diagnostics.

```
# ----------------------------
# 5) Postprocess
# ----------------------------
print(result_dynesty)
print("log_evidence =", result_dynesty.log_evidence, "+/-", result_dynesty.log_evidence_err)
result_dynesty.plot_corner()
result_dynesty.save_posterior_samples()
```

4.1.1 Contents of the Output Directory

After the inference run completes, Bilby writes several files to the output directory `outdir`. These files contain the results of the Bayesian inference, diagnostic information about the sampler, and plots that summarize the posterior distributions. The main files generated in this example are listed below.

linear_regression.log

Log file produced during the execution of the run. It contains the full record of the sampler configuration, prior definitions, progress messages, warnings, and summary statistics. This file is useful for debugging and for documenting the exact configuration of the analysis.

linear_regression_result.json

Main result file generated by Bilby. It stores the complete result object in JSON format, including the posterior samples, prior definitions, evidence values, sampler configuration, and metadata describing the run. This file can be reloaded later with Bilby to reproduce plots or perform additional analyses. For reference, the typical structure is:

```
{
        "label": "linear_regression",
        "outdir": "/path/to/outdir",
        "sampler": "dynesty",
        "log_evidence": -123.45,
        "log_evidence_err": 0.12,
        "priors": {
                "m": "Uniform(...)",
                "b": "Uniform(...)",
                "sigma": "LogUniform(...)"
        },
        "posterior": "...",
        "meta_data": { "...": "..." },
        "version": "..."
}
```

linear_regression_posterior_samples.dat

Plain text file containing the posterior samples of the inferred parameters. Each row corresponds to one posterior sample and each column corresponds to a parameter of the model. This file can be used for further analysis or visualization with external tools.

Posterior samples file The file `linear_regression_posterior_samples.dat` contains the posterior samples generated by the inference run. The file is written in plain text format so that it can be easily inspected or processed with external tools.

- **Structure of the file** The file is organized as a table in which each row corresponds to one posterior sample and each column corresponds to one parameter of the model or to an associated quantity recorded by the sampler.
- **Header line** The first line of the file contains the column names. These names identify the parameters and additional diagnostic quantities stored in the file.
- **Parameter columns** For the linear regression example, the file contains columns corresponding to the model parameters

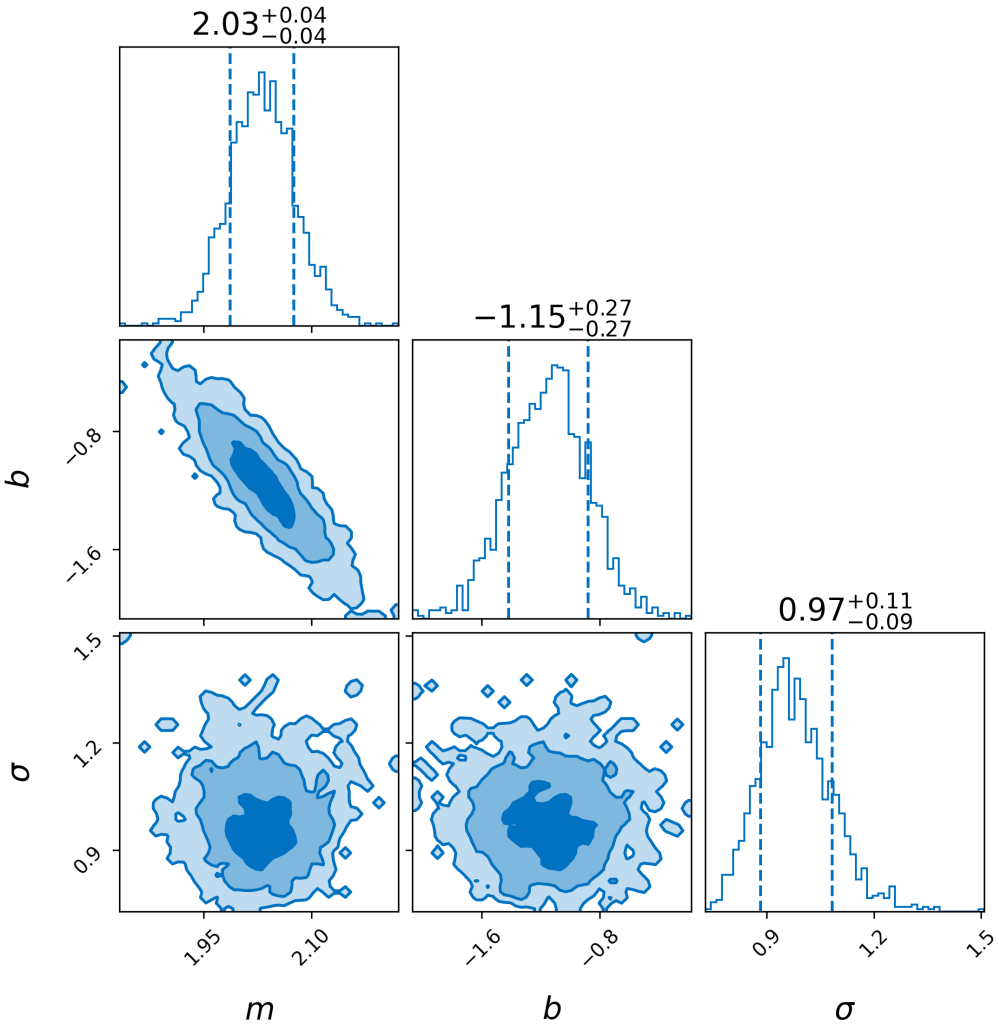
  $$m, \quad b, \quad \sigma,$$

  which represent the slope, the intercept, and the noise standard deviation of the linear model.
- **Sampler-related quantities** Additional columns may contain quantities related to the nested–sampling procedure, such as the log-likelihood value associated with each sample or the statistical weight used during the resampling stage.
- **Posterior samples** Each row represents a sample drawn from the posterior distribution after the nested–sampling output has been converted into approximately independent samples. The collection of all rows therefore provides a Monte Carlo representation of the posterior probability distribution.

- **Use of the file** This format allows the posterior samples to be easily imported into analysis environments such as Python, MATLAB, or R. For example, the file can be read as a table and used to compute posterior means, credible intervals, or custom visualizations.

In practice, the file provides a portable representation of the posterior distribution inferred by the Bayesian analysis, independent of the Bilby result object.

`linear_regression_corner.png`



Corner plot showing the posterior distributions of the inferred parameters and their correlations. The diagonal panels display the one-dimensional marginalized posterior distributions, while the off-diagonal panels show the two-dimensional joint distributions.

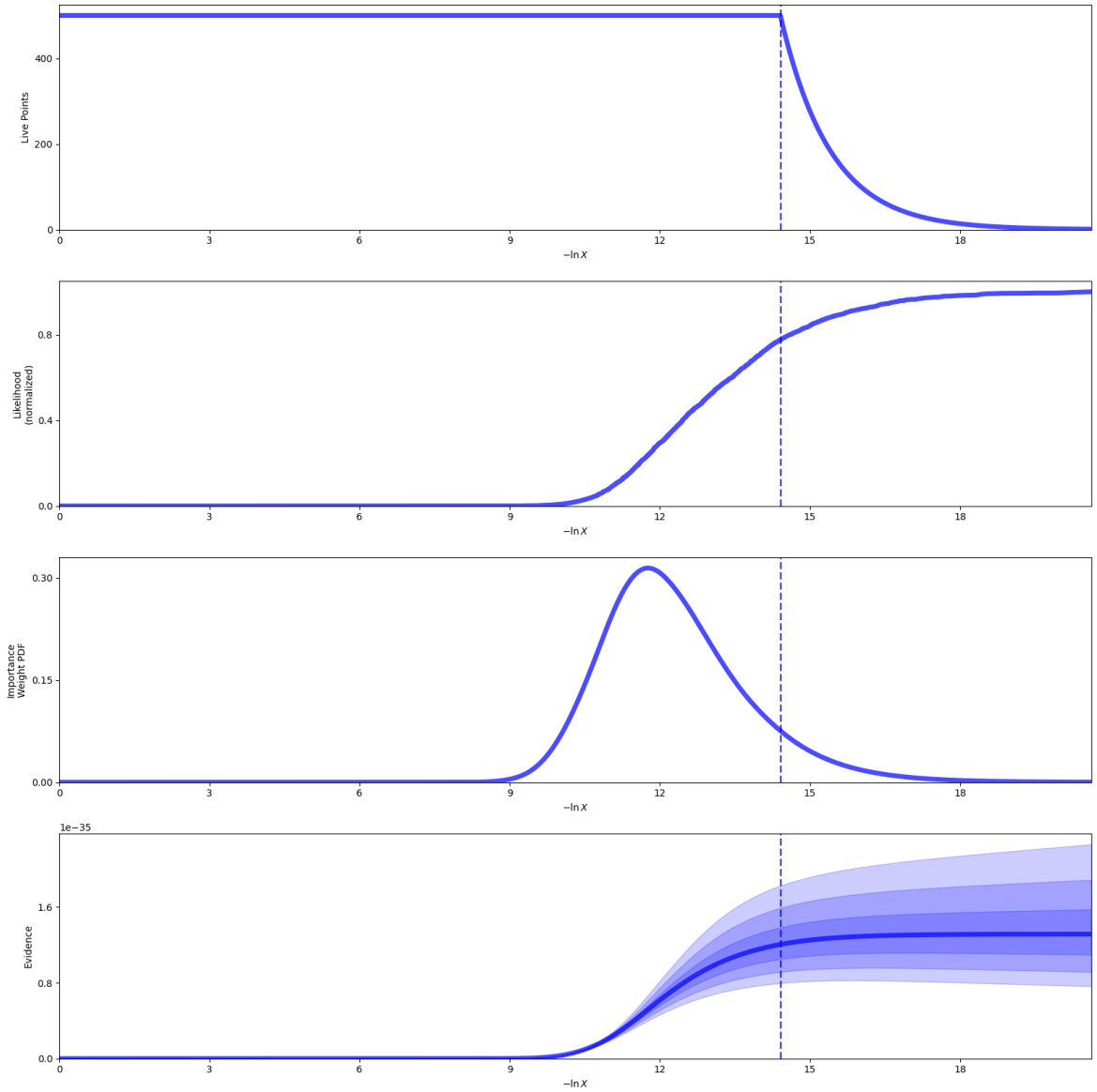
`linear_regression_resume.pickle`

Checkpoint file that stores the state of the nested sampler. If the run is interrupted, Bilby can restart the inference from this file without losing progress.

`linear_regression_dynesty.pickle`

Serialized representation of the Dynesty sampler state and results. This file contains detailed information about the nested-sampling process and can be used for advanced diagnostics or for reconstructing the run.

linear_regression_checkpoint_run.png

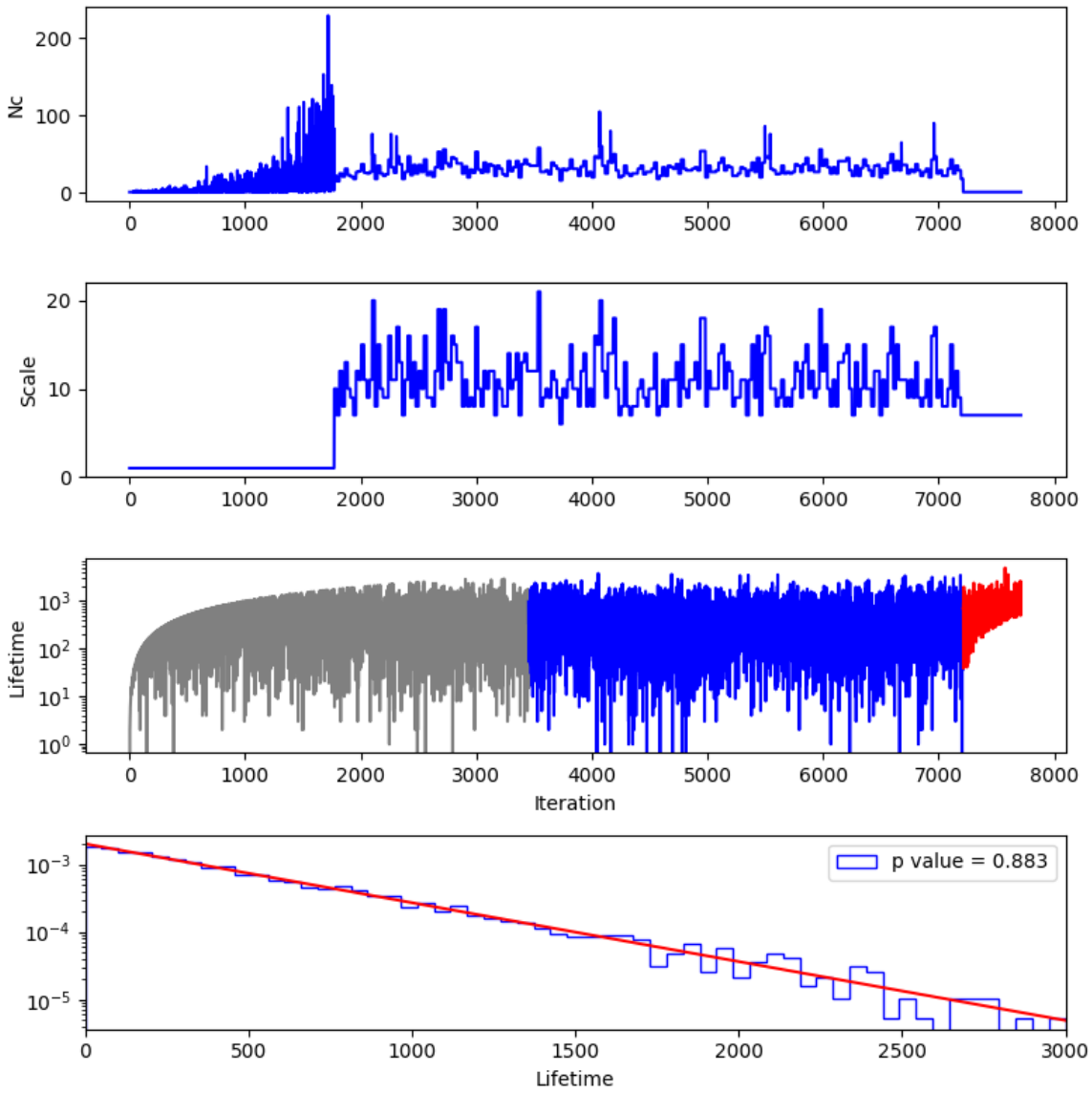


This plot displays four quantities that summarize the progress of the nested–sampling algorithm:

- **Live Points**. These are the active samples maintained by the nested–sampling algorithm. At each iteration the live point with the lowest likelihood is removed and replaced by a new point satisfying the likelihood constraint.
- **Log-likelihood** ($\log L$) Shows the likelihood values of the sampled points during the run. As the algorithm proceeds, the lowest-likelihood live point is replaced, and the sampled likelihood values gradually increase.
- **Log weight** Indicates the statistical weight assigned to each sample in the evidence integral. These weights determine how strongly each sample contributes to the posterior distribution and to the evidence estimate.
- **Log evidence** ($\log Z$) Shows the cumulative estimate of the Bayesian evidence as the run progresses. The curve stabilizes when the algorithm approaches convergence.

All these quantities are plotted as a function of **minue the Log prior volume** ($-\log X$) which represents the remaining fraction of the prior volume being explored. Nested sampling progressively shrinks this volume as it focuses on regions of higher likelihood.

linear_regression_checkpoint_stats.png



This diagnostic plot summarizes several internal quantities that describe how efficiently the nested–sampling algorithm generates new samples during the run. The plot shows the evolution of the following quantities as a function of the iteration number:

- **Number of likelihood calls** (N_c) Represents the number of likelihood evaluations required to obtain a new valid sample that satisfies the likelihood constraint. This quantity measures the computational effort needed to replace the lowest-likelihood live point. Large values of N_c indicate that many trial points were rejected before a suitable sample was found.
- **Scale** Controls the step size used by the internal proposal mechanism (typically a random-walk proposal) that generates new candidate samples. The scale parameter adapts during the run in order to maintain an efficient exploration of the constrained parameter space.
- **Lifetime** Denotes the number of iterations for which a given live point survives before being removed. A live point with a long lifetime remains among the active set for many iterations before it becomes the lowest-likelihood point and is replaced.

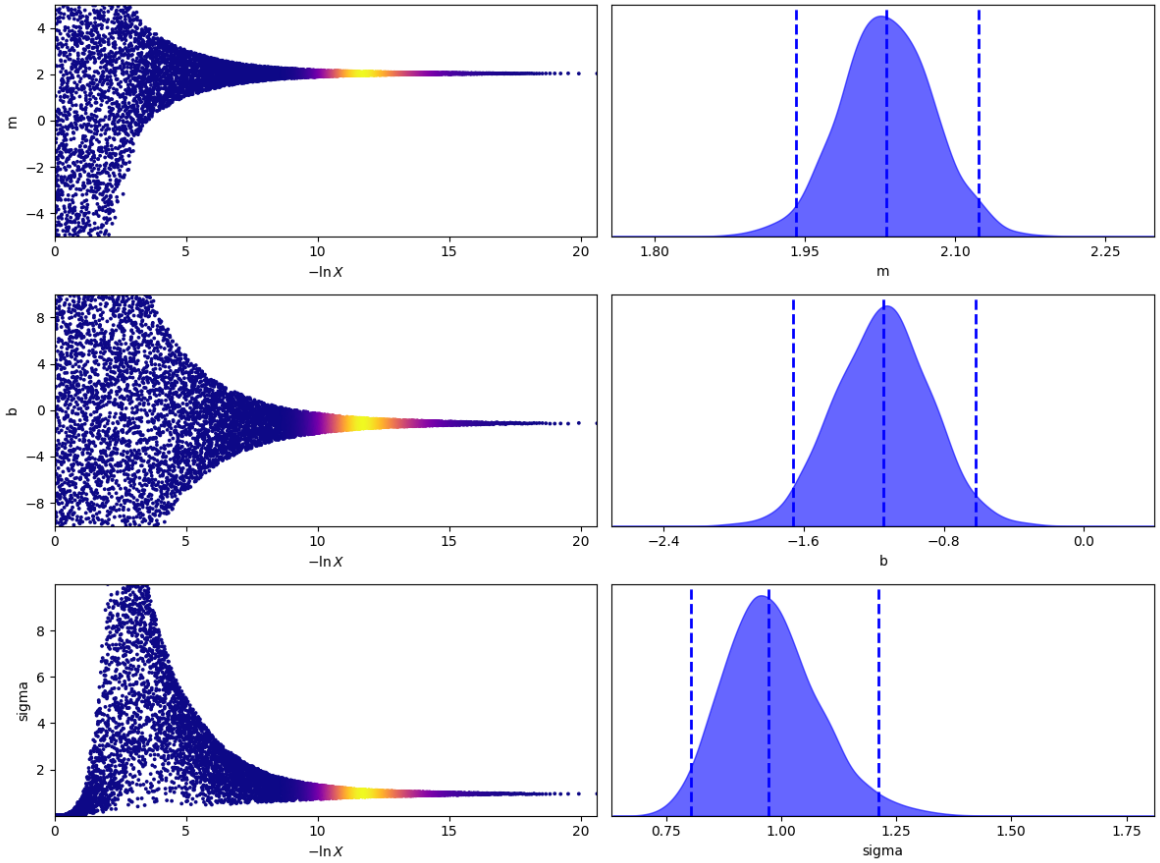
The second panel shows the quantity p as a function of the lifetime:

- **Probability** p Represents the empirical probability distribution of the lifetimes of the live points. This distribution provides a diagnostic check of the statistical behavior of the nested–sampling process.

In an ideal nested–sampling run, the lifetimes follow an approximately exponential distribution, reflecting the stochastic nature of the prior-volume shrinkage.

Together, these statistics provide insight into the efficiency of the sampling procedure and help diagnose potential problems in the exploration of the constrained parameter space.

`linear_regression_checkpoint_trace.png`

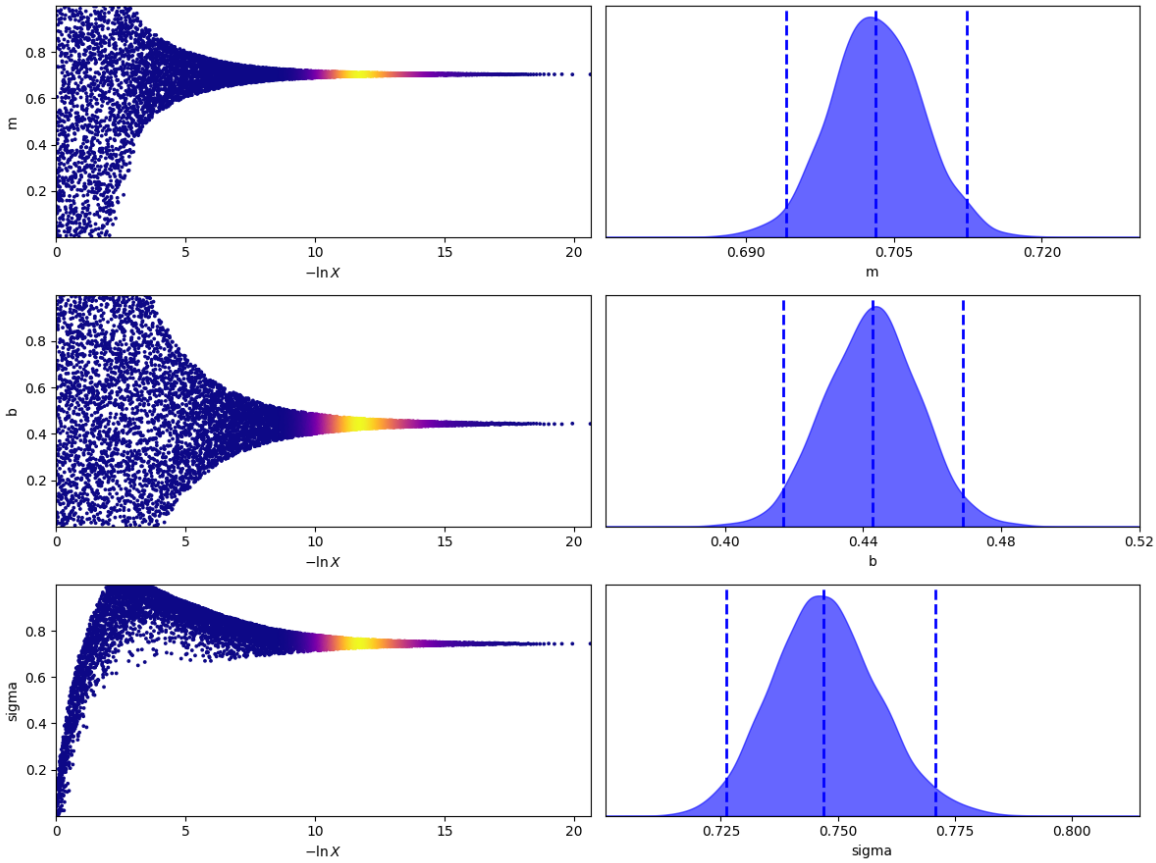


This plot shows the trace of the sampled parameter values during the nested–sampling run. Each panel corresponds to one model parameter and displays how its sampled values evolve as the algorithm progresses. The trace plot allows one to verify that the sampler explores the parameter space broadly at the beginning of the run and gradually concentrates in the region of high posterior probability. Such plots are useful diagnostic tools to check that the sampling process behaves regularly and that the parameter space is explored without pathological behavior. Trace plot showing the evolution of sampled parameter values during the run. It allows one to visually inspect the exploration of the parameter space and identify potential sampling problems. In nested sampling, X denotes the *remaining prior volume*, that is, the fraction of the total prior space that still satisfies the likelihood constraint at a given stage of the algorithm. Initially $X = 1$, corresponding to the full prior volume. As the run progresses and the likelihood threshold increases, the explored region shrinks and X decreases. The quantity $-\ln X$ is therefore a convenient measure of the progressive compression of the prior volume. Since X decreases exponentially during nested sampling, plotting $-\ln X$ produces an approximately linear progression that is easier to interpret. Large values of $-\ln X$ correspond to late stages of the run, when the sampler has focused on the small region of parameter space that contains most of the posterior probability.

The histograms shown on the right side of the trace plot represent the *marginalized posterior distributions* of the corresponding parameters. While the main part of each panel shows how the sampled parameter values evolve during the nested–sampling run, the histogram summarizes the final distribution of those values after the posterior samples have been generated. In other words, the trace illustrates the sampling process, whereas the histogram provides a compact representation of the inferred posterior probability density for each parameter.

In the plot the different colors correspond to the identities of the *live points* used by the nested–sampling algorithm. At any moment the algorithm maintains a fixed number of live points (here $N_{\text{live}} = 500$). When the point with the lowest likelihood is removed, it is replaced by a new sample that inherits the identity of that live point. The color therefore follows the trajectory of that live-point index during the run. In the early phase of the run, only a small number of dead points have been produced and recorded. Since the trace plot is drawn using the sequence of dead points, only a few live-point identities have appeared in the plot so far. Consequently, only a small number of colors are visible. As the run proceeds, more live points are successively removed and replaced, so more identities appear in the trace plot and the number of visible colors increases.

`linear_regression_checkpoint_trace_unit.png`



Normalized version of the trace plot in which parameters are rescaled to comparable units.
This representation helps to compare the behavior of different parameters during the sampling process.

Together, these files provide a complete record of the Bayesian inference run, including the posterior distributions, evidence estimates, and diagnostic information needed to assess the quality of the sampling process.

4.1.2 Commented Output of the Bilby Run

```
Running for label 'linear_regression', output will be saved to 'outdir'
```
Bilby starts the inference run. All output files will be written in the directory ***`outdir`****.*

```
Analysis priors:
```
Bilby prints the prior distributions used in the parameter estimation.

```
m = Uniform(minimum=-5, maximum=5, ...)
```
Prior for the slope parameter m. The parameter is assumed uniformly distributed in the interval $[-5, 5]$.

```
b = Uniform(minimum=-10, maximum=10, ...)
```
Prior for the intercept parameter b.

`sigma = LogUniform(minimum=0.001, maximum=10, ...)`
Prior for the noise standard deviation σ. A log-uniform distribution is appropriate for positive scale parameters.

`Analysis likelihood class: LinearRegressionLikelihood`
Bilby confirms that the custom likelihood class defined in the script is used in the analysis.

`Analysis likelihood noise evidence: nan`
No separate noise-only model is defined for this likelihood, so the noise evidence cannot be computed.

`Single likelihood evaluation took 7.234e-05 s`
Estimated computational cost of a single likelihood evaluation.

`Using sampler Dynesty with kwargs \{nlive: 500, ...\}`
*The nested sampler **`dynesty`** is initialized with 500 live points.*

`Using dynesty version 3.0.0`
Version of the Dynesty sampler used in the run.

`Generating initial points from the prior`
Initial live points for the nested sampler are drawn from the prior distributions.

`Using the bilby-implemented ensemble rwalk sampling`
Bilby uses its random-walk proposal method to explore the parameter space.

`7193it ... ncall:1.9e+05 eff:3.8\% logz=-80.41 +/- 0.15 dlogz:0.104`
Intermediate progress report showing the number of iterations, likelihood calls, efficiency, and current estimate of the log evidence.

`7211it ... logz=-80.32 +/- 0.15 dlogz:0.000192`
Final progress line. The remaining evidence contribution is negligible and the run converges.

`Rejection sampling nested samples to obtain 1562 posterior samples`
Nested-sampling points are converted into posterior samples using rejection sampling.

`Sampling time: 0:00:23.594295`
Total runtime of the nested-sampling stage.

`Summary of results:`
Bilby prints a concise summary of the statistical inference results.

`nsamples: 1562`
Total number of posterior samples obtained after resampling.

`ln_noise_evidence: nan`
No noise-only evidence was computed for this likelihood.

`ln_evidence: -80.317 +/- 0.181`
Estimated log evidence of the regression model.

`ln_bayes_factor: nan`
The Bayes factor cannot be computed because no reference noise model was provided.

`log_evidence = -80.31701187345487 +/- 0.1810091921631402`
*Same evidence value printed directly from the **`result`** object with higher numerical precision.*

`Writing samples file to .../linear_regression_posterior_samples.dat`
Posterior samples are written to a text file for further analysis.

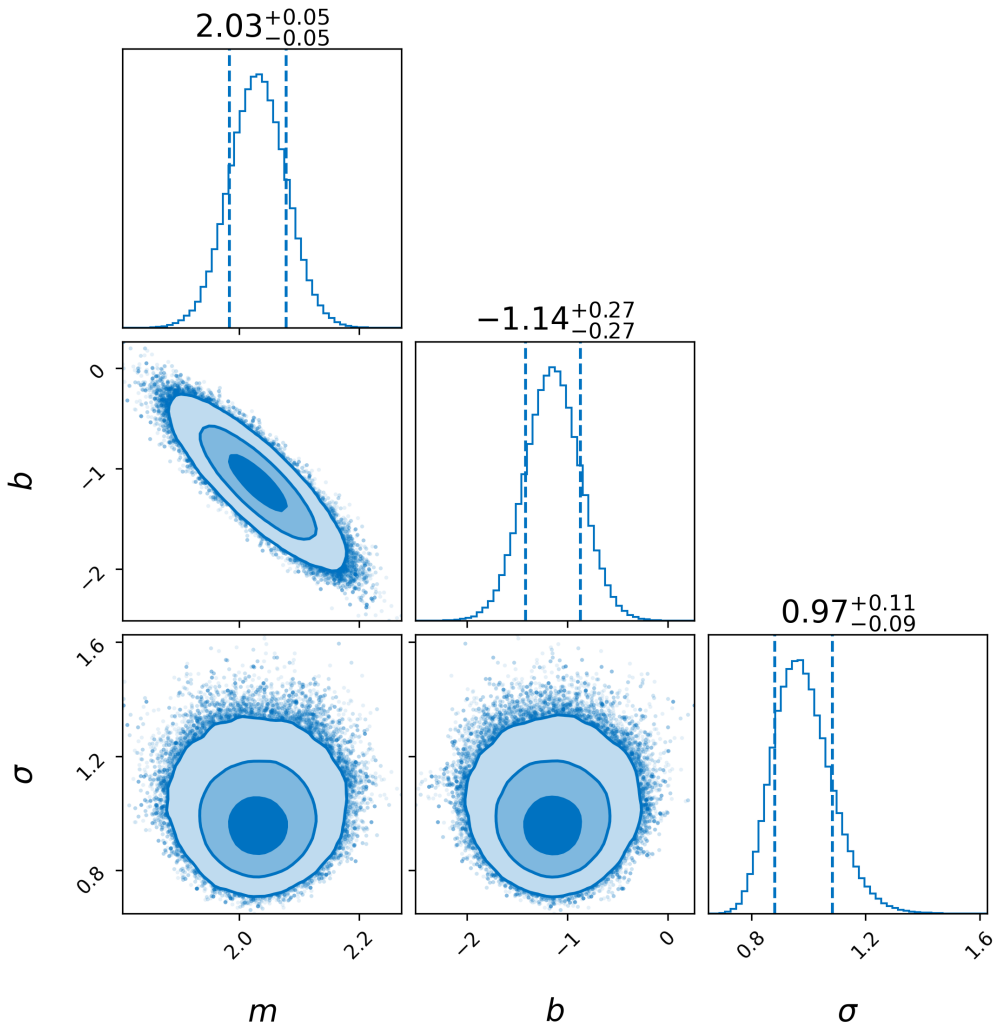
4.1.3 Example: switch dynesty -> emcee (ensemble MCMC)

`linear_regression_emcee_corner.png`

TO change the sampler:

```
result_emcee = bilby.run_sampler( likelihood=likelihood, priors=priors,
↪  sampler="emcee", nwalkers=200, nsteps=4000, outdir=outdir, label=label +
↪  "_emcee", plot=True, )
```

where nwalkers and nsteps are emcee configuration parameters.



For reference, corner plot showing the posterior distributions of the inferred parameters and their correlations in this case.

4.2 Example 2: Quasi-Degenerate Parameters with Exploratory Exercises

This example demonstrates a quasi-degenerate model using two parameters, a and b, that contribute nearly indistinguishably to the observed signal. When parameters are degenerate, the posterior exhibits strong correlations and possible ridges, making inference and sampling more challenging.

The model is:

$$y = a(x + \delta) + b(x - \delta) + \epsilon,$$

with Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and small δ.

4.2.1 Code and Inference

```python
import numpy as np
import bilby
from bilby.core.utils import random

random.seed(131)
np.random.seed(42)
n = 50

x = np.linspace(-5, 5, n)
delta = 0.05
a_true, b_true, sigma = 2.0, 2.0, 0.5
y = a_true * (x + delta) + b_true * (x - delta) + np.random.normal(0, sigma, size=x.size)
```

```python
class DegenerateLikelihood(bilby.Likelihood):
    def __init__(self, x, y):
        super().__init__(parameters={"a": None, "b": None, "sigma": None})
        self.x = np.asarray(x)
        self.y = np.asarray(y)

    def log_likelihood(self, parameters=None):
        if parameters is not None:
            self.parameters.update(parameters)
        a = self.parameters["a"]
        b = self.parameters["b"]
        sigma = self.parameters["sigma"]
        model = a * (self.x + delta) + b * (self.x - delta)
        return -0.5 * np.sum(((self.y - model) / sigma)**2 + np.log(2 * np.pi * sigma**2))


    def noise_log_likelihood(self): # (1)
        sigma = self.parameters["sigma"]
        if sigma is None or sigma <= 0:
            return -np.inf
        return -0.5 * np.sum((self.y/ sigma)**2 + np.log(2 * np.pi * sigma**2))



priors = bilby.core.prior.PriorDict()
priors["a"] = bilby.core.prior.Uniform(0, 4, name="a")
priors["b"] = bilby.core.prior.Uniform(0, 4, name="b")
priors["sigma"] = bilby.core.prior.LogUniform(1e-2, 2, name="sigma")

likelihood = DegenerateLikelihood(x, y)
outdir = "outdir"
label  = "quasi-degenerate"

result = bilby.run_sampler(
    likelihood=likelihood,
    priors=priors,
    sampler="dynesty",
    nlive=500,
    outdir=outdir + "_dynesty",
    label=label + "_dynesty",
    plot=True,
    seed=1234,
)
print(result)
print("log_evidence =", result.log_evidence, "+/-", result.log_evidence_err)
result.plot_corner()
result.save_posterior_samples()
```

(1) Log likelihood for the noise-only model. Here the signal is zero.

Questions and Observations

- What features in the corner plot indicate parameter degeneracy?
- How tightly are a and b constrained individually, versus jointly?
- Does increasing δ to 0.5 or 1.0 reduce the degeneracy? Try and compare posteriors.
- How sensitive is the evidence (log marginal likelihood) to δ?
- What happens if we impose a prior constraint like $a = b$ or $a > b$?

Suggested Experiments

1. **Sampler comparison:** Try running with `emcee`, `zeus`, and `ultranest`. Compare efficiency and convergence.

2. **Prior strength:** Use tighter priors on a and b. Observe whether the posterior becomes more distinguishable.
3. **Alternative model:** Reparametrize using $a + b$ and $a - b$. Does this help sampling?
4. **Posterior diagnostics:** Plot marginal and joint distributions, credible regions, and correlation coefficients.
5. **Likelihood exploration:** Plot $\log \mathcal{L}$ over a grid in (a, b) space to visualize the degeneracy ridge.

4.3 Example 3: Model Comparison with Bilby

Bilby supports Bayesian model comparison by computing the marginal likelihood, or *evidence*, Z, for each model. The ratio of evidences (Bayes factor) quantifies support for one model over another:

$$\text{Bayes factor} = \frac{Z_1}{Z_2}, \quad \log \text{BF} = \log Z_1 - \log Z_2$$

Samplers such as `dynesty`, `ultranest`, and `pymultinest` compute this quantity natively. MCMC-based samplers like `emcee` do not.

4.4 Linear vs Quadratic Fit

Suppose we observe data that may follow either a linear or a quadratic model:

$$\text{Model 1 (Linear):} \quad y = mx + b, \quad \text{Model 2 (Quadratic):} \quad y = ax^2 + bx + c$$

```python
import numpy as np
import bilby
from bilby.core.utils import random

random.seed(131)
np.random.seed(42)
n = 50

# Simulate quadratic data
x = np.linspace(-5, 5, n)
y = 0.5 * x**2 + x + 1 + np.random.normal(0, 2.0, size=x.size)

# Likelihoods
class LinearLike(bilby.Likelihood):
    def __init__(self, x, y):
        super().__init__(parameters={"m": None, "b": None, "sigma": None})
        self.x, self.y = x, y

    def log_likelihood(self, parameters=None):
        if parameters:
            self.parameters.update(parameters)
        m, b, sigma = self.parameters["m"], self.parameters["b"], self.parameters["sigma"]
        model = m * self.x + b
        return -0.5 * np.sum(((self.y - model) / sigma)**2 + np.log(2*np.pi*sigma**2))

class QuadLike(bilby.Likelihood):
    def __init__(self, x, y):
        super().__init__(parameters={"a": None, "b": None, "c": None, "sigma": None})
        self.x, self.y = x, y

    def log_likelihood(self, parameters=None):
        if parameters:
            self.parameters.update(parameters)
        a, b, c, sigma = self.parameters["a"], self.parameters["b"], self.parameters["c"],
        ↪  self.parameters["sigma"]
        model = a * self.x**2 + b * self.x + c
        return -0.5 * np.sum(((self.y - model) / sigma)**2 + np.log(2*np.pi*sigma**2))

# Priors
priors_linear = bilby.core.prior.PriorDict()
priors_linear["m"] = bilby.core.prior.Uniform(-10, 10)
```

```
priors_linear["b"] = bilby.core.prior.Uniform(-10, 10)
priors_linear["sigma"] = bilby.core.prior.LogUniform(1e-1, 10)

priors_quad = bilby.core.prior.PriorDict()
priors_quad["a"] = bilby.core.prior.Uniform(-5, 5)
priors_quad["b"] = bilby.core.prior.Uniform(-10, 10)
priors_quad["c"] = bilby.core.prior.Uniform(-10, 10)
priors_quad["sigma"] = bilby.core.prior.LogUniform(1e-1, 10)

# Run samplers
result_linear = bilby.run_sampler(
    LinearLike(x, y),
    priors_linear,
    sampler="dynesty",
    nlive=500,
    label="linear_model",
    outdir="model_comp",
    seed=0,
    plot=False
    )

result_quad = bilby.run_sampler(
    QuadLike(x, y),
    priors_quad,
    sampler="dynesty",
    nlive=500,
    label="quad_model",
    outdir="model_comp",
    seed=0,
    plot=False)

# Compare evidences
log_bayes_factor = result_quad.log_evidence - result_linear.log_evidence
print(f"log Bayes factor (quadratic vs linear): {log_bayes_factor:.2f}")
```

Interpretation

Bilby reports the log-evidence for each model:

- `result.log_evidence`: $\log Z$
- `result.log_evidence_err`: standard error

You can compute:

```
log_bayes_factor = result_quad.log_evidence - result_linear.log_evidence
```

Values of $\log \mathrm{BF} > 5$ strongly favor the more complex model; values near 0 suggest no significant preference.

Exercise

- What happens if the data are truly linear? Try simulating $y = 3x - 2 + \epsilon$.
- Increase noise level to test sensitivity to overfitting.
- Try using `ultranest` or `pymultinest` and compare log-evidence estimates.

SECTION 5

Lab 2 (13:30-15:00): Custom Likelihood Implementation in Bilby

In the **Bilby** framework, the likelihood is the mathematical core that evaluates how well a specific set of model parameters describes the observed data. While the library provides specialized likelihoods for gravitational waves, its power lies in its ability to be extended for any statistical model through **custom likelihood classes**.

5.1 The Framework

The implementation of a custom likelihood in Bilby follows an object-oriented approach. Users define a class that inherits from the base `Likelihood` class found in `bilby.core`. This structure ensures that the custom likelihood is immediately compatible with all supported samplers, such as Dynesty or Bilby MCMC.

To implement a likelihood, a user must typically address three components:

1. **Initialization (`__init__`)**: Passing the data (e.g., x and y values) and the signal model to the class instance.
2. **The Parameter Dictionary**: Defining which parameters the likelihood expects to receive from the sampler.
3. **The Log-Likelihood Method**: Defining the mathematical function that returns the natural logarithm of the likelihood.

5.2 Non Gaussian stochastic background

The model we are interested to is a very simple model for a non Gaussian stochastic background. It is given by

$$y^{(1)} = h + \epsilon^{(1)} \tag{1}$$
$$y^{(2)} = h + \epsilon^{(2)} \tag{2}$$

where $\epsilon^{(1)} \sim \mathcal{N}(0, \sigma_1^2)$, $\epsilon^{(2)} \sim \mathcal{N}(0, \sigma_2^2)$ are the noises of two detectors while $h \sim p_+\mathcal{N}(\mu, \sigma^2) + p_-\mathcal{N}(-\mu, \sigma^2)$ is the stochastic signal, a Gaussian mixture parameterized by μ, $\sigma^2 > 0$ and $p_+ + p_- = 1$ and $p_+, p_- \in [0, 1]$.

5.2.1 Likelihood

The likelihood is defined by $\mathcal{L} = P(y^{(1)}, y^{(2)}|\sigma_1^2, \sigma_2^2, \sigma^2, \mu, p)$. This is a multivariate Gaussian mixture with

$$\boldsymbol{y} \equiv \begin{pmatrix} y^{(1)} \\ y^{(2)} \end{pmatrix} \sim p_+\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{C}) + p_-\mathcal{N}(-\boldsymbol{\mu}, \boldsymbol{C}) \tag{3}$$
$$\boldsymbol{\mu} = \begin{pmatrix} \mu \\ \mu \end{pmatrix} \tag{4}$$
$$\boldsymbol{C} = \begin{pmatrix} \sigma^2 + \sigma_1^2 & \sigma^2 \\ \sigma^2 & \sigma^2 + \sigma_2^2 \end{pmatrix} \tag{5}$$

The likelihood is given by

$$\mathcal{L} = \frac{1}{(2\pi\sqrt{\det \boldsymbol{C}})^n}\left[p_+ e^{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})} + p_- e^{-\frac{1}{2}(\boldsymbol{y}+\boldsymbol{\mu})^T\boldsymbol{C}^{-1}(\boldsymbol{y}+\boldsymbol{\mu})}\right] \tag{6}$$
$$\boldsymbol{C}^{-1} = \frac{1}{\det \boldsymbol{C}}\begin{pmatrix} \sigma^2 + \sigma_2^2 & -\sigma^2 \\ -\sigma^2 & \sigma^2 + \sigma_1^2 \end{pmatrix} \tag{7}$$
$$\det \boldsymbol{C} = \sigma_1^2\sigma_2^2 + \sigma^2(\sigma_1^2 + \sigma_2^2) \tag{8}$$

from which we obtain the log–likelihood

$$\log \mathcal{L} = \log\left[p_+ e^{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})} + p_- e^{-\frac{1}{2}(\boldsymbol{y}+\boldsymbol{\mu})^T\boldsymbol{C}^{-1}(\boldsymbol{y}+\boldsymbol{\mu})}\right] - n\log 2\pi - \frac{n}{2}\log\det \boldsymbol{C} \tag{9}$$

We start simulating a sample of data:

```
import numpy as np
import bilby
from bilby.core.utils import random

random.seed(131)
np.random.seed(42)
ndata = 50000

bilby.core.utils.setup_logger(
    outdir="logging",
    label="nong",
    log_level="DEBUG",   # or "INFO"
    print_version=True,
)


sigma1_true = 0.5
sigma2_true = 0.5
sigma_true = 0.4
mu_true = 0.4
p_true = 0.5

injection_parameters= {
    "sigma1" : sigma1_true,
    "sigma2" : sigma2_true,
    "sigma"  : sigma_true,
    "mu" : mu_true,
    "p" : p_true
}


# Simulate data
x = np.linspace(0, 1, ndata)
h  = p_true * np.random.normal(mu_true, sigma_true, size=(ndata)) + (1.0-p_true) *
↪  np.random.normal(-mu_true, sigma_true, size=(ndata))
y1 = np.random.normal(0, sigma1_true, size=(ndata)) + h
y2 = np.random.normal(0, sigma2_true, size=(ndata)) + h
y =  np.column_stack((y1, y2))  # (1)
```

Now we can implement the likelihood:

```
# Likelihood
class NonGaussian(bilby.Likelihood):
    def __init__(self, x, y):
        super().__init__(parameters={"sigma1": None, "sigma2": None, "sigma": None, "mu": None,
        ↪  "p": None})
        self.x = np.asarray(x)
        self.y = np.asarray(y)

    def log_likelihood(self, parameters=None):
        if parameters:
            self.parameters.update(parameters)

        var1 = self.parameters["sigma1"] ** 2
        var2 = self.parameters["sigma2"] ** 2
        var  = self.parameters["sigma"] ** 2
        mu = self.parameters["mu"]
        p = self.parameters["p"]

        idetC = 1.0/(var1 * var2 + var * (var1 + var2))

        # Inverse covariance / metric matrix
        Cinv = np.array(
            [[(var+var2)*idetC, -var*idetC],
            [-var*idetC, (var+var1)*idetC]]
        )
```

```
        # Two partial log likelihoods
        u = self.y - mu
        logL1 = -0.5 * np.einsum('ki,ij,kj->', u, Cinv, u)
        u = self.y + mu
        logL2 = -0.5 * np.einsum('ki,ij,kj->', u, Cinv, u)
        deltaL = logL1-logL2
        if deltaL > 0.0:
            logL = logL1 + np.log(p + (1.-p) * np.exp(-deltaL))
        else:
            logL = logL2 + np.log(p * np.exp(deltaL) + (1.-p))
        logL -= np.size(self.x)*(np.log(2 * np.pi) - 0.5* np.log(idetC))
        return logL
```

Then we set the priors and we run the sampler

```
priors = bilby.core.prior.PriorDict()
priors["sigma1"] = bilby.core.prior.LogUniform(1e-2, 5)
priors["sigma2"] = bilby.core.prior.LogUniform(1e-2, 5)
priors["sigma"] = bilby.core.prior.LogUniform(1e-2, 5)
priors["mu"] = bilby.core.prior.Uniform(0, 5)  # (2)
priors["p"] = bilby.core.prior.Uniform(0, 1)

result = bilby.run_sampler(
    likelihood=NonGaussian(x, y),
    priors=priors,
    sampler = "dynesty",
    nlive = 500,
    label = "nongauss_model",
    outdir = "outdir",
    seed = 1234,
    plot = True,
    print_progress=True,
    checkpoint_every=600,   # seconds
    resume=True,
    )

print(result)
print("log_evidence =", result.log_evidence, "+/-", result.log_evidence_err)
result.plot_corner(truth=injection_parameters)
result.save_posterior_samples()
```

5.3 Advanced Applications in the Sources

The sources explicitly highlight that this custom framework allows for sophisticated scientific analyses beyond simple curves:

- **Complex Error Structures**: Bilby supports fitting models to data where uncertainties exist in both the independent (x) and dependent (y) variables.
- **Diverse Astrophysical Signals**: The library is used to implement likelihoods for non-CBC signals, such as **supernovae** and the **remnants of binary neutron star mergers**.
- **Modular Signal Models**: The syntax is designed so that users can easily "change the signal model" while keeping the same likelihood structure, facilitating rapid hypothesis testing.

By following this modular pattern, users can leverage Bilby's "expert-level parameter estimation infrastructure" for virtually any scientific domain.

SECTION 6

Lab 3 (15:15–17:30): GW Parameter Estimation

Key ideas

- **Waveform generation:** use LALSimulation-based models through Bilby waveform generators.
- **Detector data objects:** configure interferometers (e.g. H1/L1), PSD-based simulated noise, signal injection.
- **GW likelihood:** `GravitationalWaveTransient` with Gaussian stationary noise assumption.
- **Priors:** start from physically motivated GW prior dictionaries and narrow where appropriate for tutorial speed.
- **Runtime strategy:** reduced parameter set, modest sampler settings, and fallback to precomputed results if needed.

Learning objectives

- Configure detectors and generate simulated strain data with an injected GW signal.
- Define a reduced set of sampling parameters and corresponding priors.
- Run Bayesian parameter estimation with Bilby and interpret posteriors and correlations.

Notebook outline

1. **Injection parameters:** choose a binary black hole configuration (often non-spinning for speed), set a reference time.
2. **Waveform generator:** pick a fast approximant and sensible frequency settings (e.g. $f_{\min} \sim 20$ Hz).
3. **Interferometers:** create H1/L1 list, set strain from PSDs, inject signal, check SNR.
4. **Choose parameters to infer:** e.g. chirp mass, mass ratio, distance, inclination, coalescence time (or a subset of 4–5).
5. **Priors:** physically motivated priors, with narrowed ranges for tutorial runtime; fix unused parameters.
6. **Likelihood:** `bilby.gw.likelihood.GravitationalWaveTransient`.
7. **Sampling:** run `bilby.run_sampler` with `dynesty`; monitor progress and runtime.
8. **Results:** corner plots, credible intervals, injection recovery, distance–inclination degeneracy discussion.
9. **Optional:** posterior waveform overlay or reconstructed signal checks.

6.1 Simplified GW Inference with Bilby

To illustrate GW parameter inference with Bilby, we use a synthetic dataset modeled on a compact binary coalescence (CBC) with reduced dimensionality. We simulate strain data using Bilby's built-in waveform generator and recover parameters using nested sampling.

Gravitational-Wave Signal Model and Parameterization

In this tutorial we consider the standard compact binary coalescence (CBC) model used in gravitational-wave data analysis. The strain measured by a detector I can be written as

$$d_I(t) = h_I(t;\boldsymbol{\theta}) + n_I(t), \tag{10}$$

where $h_I(t;\boldsymbol{\theta})$ is the gravitational-wave signal, $n_I(t)$ is detector noise, and $\boldsymbol{\theta}$ denotes the set of source parameters.

1. Detector Response

The detector does not measure the two GW polarizations directly. Instead,

$$h_I(t;\boldsymbol{\theta}) = F_I^+(\alpha,\delta,\psi)\, h_+(t;\boldsymbol{\theta}) + F_I^\times(\alpha,\delta,\psi)\, h_\times(t;\boldsymbol{\theta}), \tag{11}$$

where:

- F_I^+ and $F_I^\times$ are antenna pattern functions,
- α and δ are right ascension and declination,
- ψ is the polarization angle,
- h_+ and $h_\times$ are the two GW polarizations.

The antenna patterns depend only on sky location and detector orientation.

2. Intrinsic Parameters

Intrinsic parameters determine the dynamics of the binary.

Component Masses Let m_1 and m_2 be the source-frame component masses, with $m_1 \geq m_2$.

Total mass:

$$M = m_1 + m_2 \tag{12}$$

Mass ratio:

$$q = \frac{m_2}{m_1}, \quad 0 < q \leq 1 \tag{13}$$

Chirp mass (best measured combination):

$$\mathcal{M} = \frac{(m_1 m_2)^{3/5}}{M^{1/5}} \tag{14}$$

The leading-order inspiral phase evolution depends primarily on $\mathcal{M}$:

$$\frac{df}{dt} \propto \mathcal{M}^{5/3} f^{11/3} \tag{15}$$

This explains why chirp mass is tightly constrained by data.

Spins Each component has a dimensionless spin vector:

$$\boldsymbol{\chi}_i = \frac{\boldsymbol{S}_i}{m_i^2} \tag{16}$$

For aligned-spin systems, the effective inspiral spin parameter is

$$\chi_{\rm eff} = \frac{m_1\chi_{1z} + m_2\chi_{2z}}{M}. \tag{17}$$

This parameter significantly affects phase evolution.

3. Extrinsic Parameters

Extrinsic parameters affect amplitude, phase, and detector projection.

Luminosity Distance Amplitude scales as

$$h \propto \frac{1}{D_L}, \tag{18}$$

where D_L is the luminosity distance.

Inclination Let θ_{JN} be the angle between the orbital angular momentum and the line of sight.
The polarizations scale approximately as:

$$h_+ \propto \frac{1+\cos^2\theta_{JN}}{2}, \tag{19}$$

$$h_\times \propto \cos\theta_{JN}. \tag{20}$$

This leads to the well-known distance–inclination degeneracy.

Coalescence Phase and Time The phase at merger is ϕ_c.
The coalescence time t_c enters as a time shift:

$$h(t;t_c) = h(t-t_c). \tag{21}$$

In the frequency domain, this produces a phase factor:

$$\tilde{h}(f;t_c) = \tilde{h}(f)\, e^{-2\pi i f t_c}. \tag{22}$$

4. Full Parameter Vector A typical binary black hole parameter vector is

$$\boldsymbol{\theta} = \{m_1, m_2, \boldsymbol{\chi}_1, \boldsymbol{\chi}_2, D_L, \alpha, \delta, \theta_{JN}, \psi, \phi_c, t_c\}. \tag{23}$$

In practice, this corresponds to ~ 15 parameters.

5. Likelihood Function Assuming stationary Gaussian noise with power spectral density $S_n(f)$, the log-likelihood is

$$\log \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2}\sum_I \left\langle d_I - h_I(\boldsymbol{\theta}) \,|\, d_I - h_I(\boldsymbol{\theta}) \right\rangle_I, \tag{24}$$

where the noise-weighted inner product is

$$\langle a|b\rangle = 4\,\mathrm{Re}\int_0^\infty \frac{\tilde{a}(f)\tilde{b}^*(f)}{S_n(f)}\,df. \tag{25}$$

This is the likelihood implemented internally by `bilby.gw.likelihood.GravitationalWaveTransient`.

This parameterization separates:

- **Intrinsic physics** (masses, spins),
- **Geometry and projection** (distance, sky position, inclination),
- **Timing and phase**.

In the tutorial lab, we typically reduce dimensionality by fixing selected parameters and sampling only a subset (e.g. $\mathcal{M}$, q, D_L, θ_{JN}, t_c) to ensure feasible runtime.

6.2 Code: Simplified Binary Black Hole Signal

We will experiment now with a minimal Bilby example for parameter estimation on a simulated binary black hole signal. This script does the following:

1. Sets up reproducible random numbers and logging.
2. Defines an injected compact-binary-coalescence signal.
3. Builds a waveform generator for that signal model.
4. Creates simulated detector noise for one interferometer (H1).
5. Injects the signal into the simulated noise.
6. Defines a prior distribution for the parameters we want to infer.
7. Fixes all other parameters to their true injected values.

8. Builds the gravitational-wave likelihood.
9. Runs the dynesty nested sampler.
10. Produces summary output and posterior plots.

```python
# Import the Bilby library.
# Bilby is the main Bayesian inference package we will use.
import bilby

# Import NumPy.
# We do not need much NumPy here, but it is commonly useful in GW scripts.
import numpy as np


# ============================================================
# 1. REPRODUCIBILITY AND OUTPUT CONFIGURATION
# ============================================================

# Set the random seed used by Bilby.
# This makes the example reproducible:
# if you run the script again with the same environment, you should get
# very similar random noise and similar sampling behavior.
bilby.core.utils.random.seed(2026)

# Directory where Bilby will save output files:
# posterior samples, plots, metadata, checkpoints, and logs.
outdir = "gw_example"

# Label used as a prefix for many output files.
# For example, Bilby will create files whose names include this label.
label = "binary_bh_detailed"

# Set up the Bilby logger.
# This prints progress information to the terminal and writes a log file
# in the output directory.
bilby.utils.setup_logger(outdir=outdir, label=label)


# ============================================================
# 2. DATA SEGMENT AND WAVEFORM SETTINGS
# ============================================================

# Total duration of the strain data segment, in seconds.
# A 4-second segment is a safe and simple choice for a BBH tutorial.
duration = 4.0

# Sampling frequency of the strain time series, in Hz.
# This means the data are sampled 1024 times per second.
sampling_frequency = 1024.0

# Lowest frequency used in the analysis, in Hz.
# Below this frequency, the data and waveform are ignored.
minimum_frequency = 20.0

# Reference frequency used internally by the waveform model.
# Spin and phase conventions are defined relative to this frequency.
reference_frequency = 50.0


# ============================================================
# 3. DEFINE THE TRUE INJECTION PARAMETERS
# ============================================================

# These are the "true" source parameters used to generate the synthetic signal.
# Later, we will ask Bilby to recover some of them from the data.
```

```
injection_parameters = dict(
    # Component masses of the binary black hole, in solar masses.
    mass_1=36.0,
    mass_2=29.0,

    # Dimensionless spin magnitudes of the two black holes.
    a_1=0.4,
    a_2=0.3,

    # Tilt angles between each spin and the orbital angular momentum.
    # Zero means aligned with the orbital angular momentum.
    tilt_1=0.0,
    tilt_2=0.0,

    # Azimuthal spin angles used in precessing-spin parameterizations.
    phi_12=0.0,
    phi_jl=0.0,

    # Luminosity distance to the source, in megaparsecs.
    luminosity_distance=500.0,

    # Inclination angle between the line of sight and the orbital angular momentum.
    theta_jn=0.4,

    # Orbital phase at coalescence.
    phase=1.3,

    # Geocentric coalescence time in GPS seconds.
    # This is the reference merger time at the Earth's center.
    geocent_time=1126259642.413,

    # Right ascension and declination of the source position on the sky.
    ra=1.375,
    dec=-1.2108,

    # Polarization angle.
    psi=2.659,
)


# ============================================================
# 4. DEFINE THE WAVEFORM MODEL
# ============================================================

# These arguments tell Bilby which waveform approximant to use and
# how to configure it.
waveform_arguments = dict(
    # IMRPhenomPv2 is a phenomenological inspiral-merger-ringdown model
    # with simple precession support.
    waveform_approximant="IMRPhenomPv2",

    # Reference frequency used by the waveform model.
    reference_frequency=reference_frequency,

    # Lowest frequency at which the waveform is generated.
    minimum_frequency=minimum_frequency,
)

# Create the waveform generator object.
# This object knows how to produce gravitational-wave strain in the frequency domain
# for a given set of source parameters.
waveform_generator = bilby.gw.waveform_generator.WaveformGenerator(
    # Length of the data segment to be matched.
    duration=duration,
```

```
    # Sampling rate of the data.
    sampling_frequency=sampling_frequency,

    # The source model that generates the frequency-domain waveform.
    # This function converts physical parameters into h(f).
    frequency_domain_source_model=bilby.gw.source.lal_binary_black_hole,

    # The waveform settings defined above.
    waveform_arguments=waveform_arguments,
)


# ============================================================
# 5. CREATE A DETECTOR AND SIMULATED NOISE
# ============================================================

# Create an "empty" interferometer corresponding to LIGO Hanford (H1).
# "Empty" means it has detector metadata but no strain data yet.
ifo = bilby.gw.detector.get_empty_interferometer("H1")

# Set the detector's analysis lower-frequency cutoff.
ifo.minimum_frequency = minimum_frequency

# Generate simulated Gaussian noise from the detector power spectral density.
# This populates the interferometer with synthetic strain data.
ifo.set_strain_data_from_power_spectral_density(
    # Sampling frequency of the generated strain time series.
    sampling_frequency=sampling_frequency,

    # Total duration of the data segment.
    duration=duration,

    # Start time of the data segment.
    # We center the merger inside the segment by choosing
    # start_time = merger_time - duration/2.
    start_time=injection_parameters["geocent_time"] - duration / 2,
)

# Inject the synthetic gravitational-wave signal into the detector data.
# The detector response includes the antenna pattern and time delay
# appropriate for H1 and the source sky location.
ifo.inject_signal(
    # Object that generates the waveform.
    waveform_generator=waveform_generator,

    # True source parameters of the injected signal.
    parameters=injection_parameters,
)

# Bilby likelihoods usually expect a list of interferometers,
# even if there is only one detector.
interferometers = [ifo]


# ============================================================
# 6. BUILD THE PRIOR DISTRIBUTION
# ============================================================

# Start from Bilby's standard BBH prior dictionary.
# This loads a default set of priors commonly used for binary black hole analyses.
priors = bilby.gw.prior.BBHPriorDict()

# IMPORTANT:
# The default BBH prior dictionary often contains a mass parameterization
# in terms of chirp_mass and mass_ratio.
```

```
# If we want to sample directly in mass_1 and mass_2, we must remove
# the redundant mass coordinates.
#
# Otherwise Bilby will complain that the sampling set is redundant.
priors.pop("chirp_mass", None)
priors.pop("mass_ratio", None)

# Define the prior for mass_1.
# We choose a simple uniform prior between 20 and 60 solar masses.
priors["mass_1"] = bilby.core.prior.Uniform(
    minimum=20.0,
    maximum=60.0,
    name="mass_1",
)

# Define the prior for mass_2.
# Again, a simple uniform prior, slightly narrower.
priors["mass_2"] = bilby.core.prior.Uniform(
    minimum=20.0,
    maximum=50.0,
    name="mass_2",
)

# Define the prior for luminosity distance.
# This tutorial uses a simple uniform prior in distance.
# In more realistic analyses, other choices are common.
priors["luminosity_distance"] = bilby.core.prior.Uniform(
    minimum=100.0,
    maximum=1000.0,
    name="luminosity_distance",
)

# Define the prior for coalescence time.
# We allow the merger time to vary only in a narrow window around the true value.
priors["geocent_time"] = bilby.core.prior.Uniform(
    minimum=injection_parameters["geocent_time"] - 0.1,
    maximum=injection_parameters["geocent_time"] + 0.1,
    name="geocent_time",
)

# List of parameters that we want to sample.
# These are the only free parameters in this tutorial.
free_parameters = {
    "mass_1",
    "mass_2",
    "luminosity_distance",
    "geocent_time",
}

# For every parameter in the prior dictionary:
# if it is not one of the free parameters and it exists in the injection dictionary,
# we fix it to the true injected value.
#
# In Bilby, assigning a number directly as a prior entry effectively means:
# "do not sample this parameter; keep it fixed to this value."
for key in list(priors.keys()):
    if key not in free_parameters and key in injection_parameters:
        priors[key] = injection_parameters[key]

# Optional sanity check:
# this prints whether Bilby still sees redundant sampling coordinates.
# It should print False.
print("Has redundant keys?", priors.test_has_redundant_keys())

# Validate the prior against the waveform duration and minimum frequency.
```

```
# This is useful for CBC analyses because some prior choices can imply signals
# that do not fit properly inside the chosen data segment.
priors.validate_prior(
    duration=duration,
    minimum_frequency=minimum_frequency,
)


# ============================================================
# 7. BUILD THE LIKELIHOOD
# ============================================================

# Construct the standard transient GW likelihood for Gaussian detector noise.
# This compares the detector data to the model waveform for each sampled point
# in parameter space.
likelihood = bilby.gw.likelihood.GravitationalWaveTransient(
    # List of detectors used in the analysis.
    interferometers=interferometers,

    # Waveform generator that produces model templates.
    waveform_generator=waveform_generator,
)


# ============================================================
# 8. RUN THE SAMPLER
# ============================================================

# Run nested sampling with dynesty.
# Bilby returns a Result object containing posterior samples, evidence estimates,
# metadata, and helper plotting methods.
result = bilby.run_sampler(
    # Likelihood object defined above.
    likelihood=likelihood,

    # Prior dictionary.
    priors=priors,

    # Choose dynesty as the sampler backend.
    sampler="dynesty",

    # Number of live points.
    # More live points usually improve exploration but cost more time.
    nlive=500,

    # Stopping criterion for nested sampling.
    # Larger values stop earlier; smaller values push for more precision.
    dlogz=0.5,

    # Directory for output files.
    outdir=outdir,

    # Label used in output filenames.
    label=label,

    # Store the true injected values in the Result object.
    # This is useful because Bilby can use them in plots.
    injection_parameters=injection_parameters,

    # Ask Bilby to produce standard plots automatically.
    plot=True,

    # Resume from previous checkpoints if they exist.
    resume=True,
```

```
    # Clean up some temporary files when possible.
    clean=True,
)


# ============================================================
# 9. PRINT A SHORT SUMMARY
# ============================================================

# Print the Bilby Result object.
# This usually includes summary information about the run.
print(result)

# Print summary statistics for the posterior samples of the main inferred parameters.
# The posterior is stored as a pandas DataFrame.
print(
    result.posterior[
        ["mass_1", "mass_2", "luminosity_distance", "geocent_time"]
    ].describe()
)


# ============================================================
# 10. PRODUCE A CORNER PLOT
# ============================================================

# Make a corner plot of the posterior distribution.
# Because we passed injection_parameters to run_sampler, Bilby can mark
# the true injected values on the plot.
result.plot_corner()
```

6.2.1 Comments and Interpretation

This example infers only a subset of parameters (masses, distance, coalescence time) and fixes the rest. This is useful for fast demonstrations and avoids degeneracies.

- The injected signal mimics a binary black hole merger with moderate spins and SNR.
- Posteriors can be viewed using `result.plot_corner()` or saved via `result.save_posterior_samples()`.

6.2.2 Questions

- What is the correlation between m_1 and m_2 in the posterior? How does it reflect mass ratio uncertainty?
- Is d_L (distance) well constrained? What affects this?
- Does adding θ_{JN} as a free parameter increase uncertainties? Why?
- Try fixing only total mass or chirp mass—how does this change the posterior shapes?

6.2.3 Suggested Experiments

1. **Switch samplers:** Run with `emcee` and compare to `dynesty`.
2. **Signal recovery test:** Shift the true distance or masses—how robust is the recovery?
3. **Noise-only run:** Remove the injection and see how the posterior behaves.
4. **Add detector:** Include L1 or Virgo—how does sky localization improve?

6.3 Code: Binary Black Hole Signal on a Detector Network

A simple Bilby example for parameter estimation on a simulated binary black hole signal using three detectors:

- H1 = LIGO Hanford
- L1 = LIGO Livingston

- V1 = Virgo

This script does the following:

1. Sets up reproducible random numbers and logging.
2. Defines an injected compact-binary-coalescence signal.
3. Builds a waveform generator for that signal model.
4. Creates simulated detector noise for a three-detector network.
5. Injects the same astrophysical signal into all detectors.
6. Defines priors for a small set of parameters to infer.
7. Fixes the remaining parameters to their true injected values.
8. Builds the gravitational-wave likelihood for the detector network.
9. Runs the dynesty nested sampler.
10. Produces summary output and posterior plots.

This version shows how Bilby works with a detector network, which is the normal situation in gravitational-wave astronomy.

```
# Import the Bilby library.
# This is the main package used for Bayesian inference.
import bilby

# Import NumPy.
# It is not heavily used here, but it is often useful in tutorial scripts.
import numpy as np


# ============================================================
# 1. REPRODUCIBILITY AND OUTPUT CONFIGURATION
# ============================================================

# Set Bilby's random seed.
# This makes the random noise generation and some stochastic parts
# of the run reproducible.
bilby.core.utils.random.seed(2026)

# Directory where Bilby will save all output products.
# These include logs, JSON files, posterior samples, plots, and checkpoints.
outdir = "gw_example_3det"

# Label used in file names produced by Bilby.
label = "binary_bh_three_detectors"

# Create a logger so that Bilby prints progress information
# to the terminal and writes a log file in the output directory.
bilby.utils.setup_logger(outdir=outdir, label=label)


# ============================================================
# 2. DATA SEGMENT AND WAVEFORM SETTINGS
# ============================================================

# Length of the analyzed strain data segment, in seconds.
duration = 4.0

# Sampling frequency of the detector strain data, in Hz.
sampling_frequency = 1024.0

# Lower cutoff frequency used in the analysis, in Hz.
minimum_frequency = 20.0
```

```
# Reference frequency used internally by the waveform model.
reference_frequency = 50.0


# ============================================================
# 3. DEFINE THE TRUE INJECTION PARAMETERS
# ============================================================

# These are the true physical parameters of the injected source.
# Bilby will generate a synthetic GW signal from them.
injection_parameters = dict(
    # Component masses in solar masses.
    mass_1=36.0,
    mass_2=29.0,

    # Dimensionless spin magnitudes.
    a_1=0.4,
    a_2=0.3,

    # Spin tilt angles relative to the orbital angular momentum.
    tilt_1=0.0,
    tilt_2=0.0,

    # Azimuthal spin angles for the precessing-spin parameterization.
    phi_12=0.0,
    phi_jl=0.0,

    # Luminosity distance in megaparsecs.
    luminosity_distance=500.0,

    # Inclination angle between the orbital angular momentum and the line of sight.
    theta_jn=0.4,

    # Orbital phase at coalescence.
    phase=1.3,

    # Geocentric merger time in GPS seconds.
    geocent_time=1126259642.413,

    # Sky position of the source.
    ra=1.375,
    dec=-1.2108,

    # Polarization angle.
    psi=2.659,
)


# ============================================================
# 4. DEFINE THE WAVEFORM MODEL
# ============================================================

# These settings specify the waveform approximant and some frequency choices.
waveform_arguments = dict(
    # A commonly used phenomenological inspiral-merger-ringdown model.
    waveform_approximant="IMRPhenomPv2",

    # Frequency at which some parameters are defined.
    reference_frequency=reference_frequency,

    # Lowest frequency included in the waveform generation.
    minimum_frequency=minimum_frequency,
)
```

```
# Build the waveform generator.
# This object converts source parameters into a model waveform h(f).
waveform_generator = bilby.gw.waveform_generator.WaveformGenerator(
    # Total duration of the waveform/data segment.
    duration=duration,

    # Sampling frequency of the data.
    sampling_frequency=sampling_frequency,

    # The source model that generates frequency-domain BBH waveforms.
    frequency_domain_source_model=bilby.gw.source.lal_binary_black_hole,

    # Additional waveform settings.
    waveform_arguments=waveform_arguments,
)


# ============================================================
# 5. CREATE A THREE-DETECTOR NETWORK
# ============================================================

# Create an interferometer network containing:
#   H1 = Hanford
#   L1 = Livingston
#   V1 = Virgo
#
# Bilby automatically loads the detector geometry and design sensitivity
# information associated with these instruments.
interferometers = bilby.gw.detector.InterferometerList(["H1", "L1", "V1"])

# Loop over each detector in the network.
for ifo in interferometers:
    # Set the low-frequency cutoff for this detector.
    ifo.minimum_frequency = minimum_frequency

    # Generate synthetic Gaussian noise for this detector from its PSD.
    # Each detector gets its own independent noise realization.
    ifo.set_strain_data_from_power_spectral_density(
        sampling_frequency=sampling_frequency,
        duration=duration,

        # We place the merger roughly at the center of the segment.
        start_time=injection_parameters["geocent_time"] - duration / 2,
    )

# Inject the same astrophysical signal into the whole detector network.
# Bilby computes the correct detector response separately for H1, L1, and V1,
# taking into account:
#   - antenna patterns
#   - detector locations
#   - arrival-time delays across Earth
interferometers.inject_signal(
    waveform_generator=waveform_generator,
    parameters=injection_parameters,
)


# ============================================================
# 6. BUILD THE PRIOR DISTRIBUTION
# ============================================================

# Start from Bilby's default BBH prior dictionary.
priors = bilby.gw.prior.BBHPriorDict()

# IMPORTANT:
```

```
# The default BBH prior dictionary often includes the mass sector
# in terms of chirp_mass and mass_ratio.
#
# In this tutorial we instead want to sample directly in mass_1 and mass_2,
# so we must remove the redundant mass coordinates.
priors.pop("chirp_mass", None)
priors.pop("mass_ratio", None)

# Define a uniform prior on the primary mass.
priors["mass_1"] = bilby.core.prior.Uniform(
    minimum=20.0,
    maximum=60.0,
    name="mass_1",
)

# Define a uniform prior on the secondary mass.
priors["mass_2"] = bilby.core.prior.Uniform(
    minimum=20.0,
    maximum=50.0,
    name="mass_2",
)

# Define a uniform prior on luminosity distance.
priors["luminosity_distance"] = bilby.core.prior.Uniform(
    minimum=100.0,
    maximum=1000.0,
    name="luminosity_distance",
)

# Define a narrow prior on geocentric merger time.
priors["geocent_time"] = bilby.core.prior.Uniform(
    minimum=injection_parameters["geocent_time"] - 0.1,
    maximum=injection_parameters["geocent_time"] + 0.1,
    name="geocent_time",
)

# In this example we now also allow the sky position to vary.
# This is much more meaningful with three detectors than with one detector,
# because a detector network can constrain sky location using time delays
# and antenna-pattern differences.
priors["ra"] = bilby.core.prior.Uniform(
    minimum=0.0,
    maximum=2.0 * np.pi,
    name="ra",
    boundary="periodic",
)

# Declination must lie between -pi/2 and +pi/2.
# A cosine-like sky prior is often used in practice, but for a simple
# tutorial we keep the example explicit and readable.
priors["dec"] = bilby.core.prior.Cosine(
    name="dec"
)

# Define the set of free parameters that will be sampled.
free_parameters = {
    "mass_1",
    "mass_2",
    "luminosity_distance",
    "geocent_time",
    "ra",
    "dec",
}

# Fix all remaining parameters to the true injected values.
```

```
# In Bilby, assigning a number directly means that parameter is fixed.
for key in list(priors.keys()):
    if key not in free_parameters and key in injection_parameters:
        priors[key] = injection_parameters[key]

# Print a quick diagnostic to check whether redundant coordinates remain.
# This should print False.
print("Has redundant keys?", priors.test_has_redundant_keys())

# Validate the prior against the waveform duration and minimum frequency.
# This is a good habit in CBC analyses.
priors.validate_prior(
    duration=duration,
    minimum_frequency=minimum_frequency,
)


# ============================================================
# 7. BUILD THE NETWORK LIKELIHOOD
# ============================================================

# Construct the standard Gaussian-noise transient GW likelihood.
# Since we pass a detector network, the likelihood is built from the joint
# contribution of H1, L1, and V1 together.
likelihood = bilby.gw.likelihood.GravitationalWaveTransient(
    interferometers=interferometers,
    waveform_generator=waveform_generator,
)


# ============================================================
# 8. RUN THE SAMPLER
# ============================================================

# Run nested sampling using dynesty.
# The result object will contain posterior samples, evidence estimates,
# metadata, and plotting utilities.
result = bilby.run_sampler(
    likelihood=likelihood,
    priors=priors,
    sampler="dynesty",

    # Number of live points.
    # For a 6D tutorial problem like this, 500 is a reasonable starting point,
    # though more live points may give smoother posteriors.
    nlive=500,

    # Nested-sampling stopping criterion.
    dlogz=0.5,

    outdir=outdir,
    label=label,

    # Store the true injection values in the output result.
    injection_parameters=injection_parameters,

    # Ask Bilby to generate standard plots automatically.
    plot=True,

    # Resume from a previous checkpoint if available.
    resume=True,

    # Remove temporary files when possible.
    clean=True,
)
```

```
# ============================================================
# 9. PRINT A SHORT SUMMARY
# ============================================================

# Print the Bilby result summary.
print(result)

# Print descriptive statistics for a few posterior parameters.
print(
    result.posterior[
        ["mass_1", "mass_2", "luminosity_distance", "geocent_time", "ra", "dec"]
    ].describe()
)


# ============================================================
# 10. MAKE A CORNER PLOT
# ============================================================

# Produce a corner plot of the posterior.
# Since we passed injection_parameters to run_sampler, Bilby can mark
# the true injected values on the figure.
result.plot_corner()
```

6.3.1 Comments and Interpretation

This example illustrates a simple Bayesian parameter–estimation analysis of a binary black hole (BBH) signal using the `bilby` framework and a network of three gravitational-wave detectors: LIGO Hanford (H1), LIGO Livingston (L1), and Virgo (V1). The script generates synthetic detector data, injects a gravitational-wave signal corresponding to a compact binary coalescence, and attempts to recover the source parameters using nested sampling.

The waveform model used in the analysis is `IMRPhenomPv2`, a phenomenological inspiral–merger–ringdown model that includes spin precession in an approximate way. The waveform is generated in the frequency domain using the function `lal_binary_black_hole`, which provides the strain amplitude and phase predicted by general relativity for a binary black hole system with the specified parameters.

A synthetic data segment is created for each detector using a Gaussian noise realization consistent with the detector power spectral density. The signal is then injected into the data using the detector response appropriate for the source sky location and polarization. The detector network is therefore sensitive to relative time delays between detectors and to the different antenna patterns of the instruments.

The likelihood function used in the inference is the standard Gaussian-noise transient likelihood implemented in `bilby.gw.likelihood.GravitationalWaveTransient`. This likelihood compares the data in each detector with the predicted waveform and combines the contributions from the entire detector network.

The parameters inferred in this example are

- the component masses m_1 and m_2,
- the luminosity distance d_L,
- the geocentric merger time t_c,
- the sky position (α, δ).

All other parameters are fixed to their injected values. This reduces the dimensionality of the parameter space and allows the inference problem to run quickly, which is useful for demonstration and teaching purposes.

The nested–sampling algorithm `dynesty` is used to explore the posterior distribution. The sampler evolves a set of "live points" in parameter space, gradually concentrating them in regions of high likelihood. The result is an estimate of the posterior probability distribution of the parameters as well as the Bayesian evidence for the model.

Because the analysis uses three detectors, the inference can exploit differences in signal arrival time and antenna response to constrain the sky position. This illustrates one of the central ideas of gravitational-wave astronomy: localization of sources relies on a detector network rather than on a single instrument.

6.3.2 Questions

1. Why is a network of detectors necessary to determine the sky position of a gravitational-wave source? What information do multiple detectors provide that a single detector cannot?
2. The likelihood assumes stationary Gaussian noise. What aspects of real gravitational-wave detector noise might violate this assumption?
3. The example samples directly in the component masses m_1 and m_2. What alternative parameterizations of the mass sector are commonly used in gravitational-wave analyses, and why might they improve sampling efficiency?
4. The prior for the luminosity distance is chosen to be uniform in d_L. What physical assumptions are implicit in this choice? What alternative distance priors are often used?
5. The script fixes most parameters (such as spin angles and inclination) to their injected values. How would allowing these parameters to vary change the dimensionality and complexity of the inference problem?
6. The nested sampler uses a fixed number of live points. How does increasing the number of live points affect the accuracy and runtime of the inference?
7. Examine the corner plot produced by the script. Which parameters show correlations in the posterior distribution, and what physical degeneracies might explain them?

6.3.3 Suggested Experiments

The script provides a starting point for exploring gravitational-wave parameter estimation. Several modifications can be made to investigate how different assumptions affect the inference.

1. **Change the signal strength.** Increase or decrease the luminosity distance of the injected source and observe how the signal-to-noise ratio and posterior uncertainties change.
2. **Use a single detector.** Repeat the analysis using only one interferometer (for example H1). Compare the resulting posterior distributions with the three-detector case and examine how the sky localization degrades.
3. **Allow additional parameters to vary.** Extend the set of sampled parameters by allowing the inclination angle θ_{jn} or polarization angle ψ to vary. Observe how degeneracies between distance and inclination appear in the posterior.
4. **Modify the mass prior.** Replace the component-mass priors with a prior on chirp mass $\mathcal{M}$ and mass ratio q. Compare the sampling efficiency and the structure of the posterior distributions.
5. **Change the waveform model.** Replace `IMRPhenomPv2` with another waveform approximant (for example `IMRPhenomD`). Examine whether the recovered parameters change significantly.
6. **Increase the sampler resolution.** Increase the number of live points used by the nested sampler and compare the smoothness and stability of the posterior distributions.
7. **Vary the data duration.** Change the duration of the data segment and verify that the prior validation step still succeeds. Observe how the data window affects the analysis.

These experiments help illustrate how Bayesian inference, detector networks, waveform modeling, and prior assumptions interact in gravitational-wave data analysis.

6.4 Code: Binary Black Hole Signal on a Detector Network and inclination and polarization parameters

A Bilby example for parameter estimation on a simulated binary black hole signal using THREE detectors:

- H1 = LIGO Hanford
- L1 = LIGO Livingston
- V1 = Virgo

This version is a step beyond the previous three-detector example. Now we allow Bilby to sample not only:

- mass_1

- mass_2
- luminosity_distance
- geocent_time
- ra
- dec

but also:

- theta_jn (binary inclination)
- psi (polarization angle)

This begins to show real network parameter-estimation structure:

- the sky position is constrained by timing and antenna-pattern differences
- the inclination angle theta_jn affects the relative strength of the plus and cross polarizations
- the polarization angle psi rotates the polarization basis seen by the detectors
- luminosity_distance and theta_jn can show degeneracy
- psi matters only because the detectors have different antenna responses

```
# Import Bilby, the Bayesian inference library used in this example.
import bilby

# Import NumPy.
# We use it here mainly for pi and for angular prior limits.
import numpy as np


# ============================================================
# 1. REPRODUCIBILITY AND OUTPUT CONFIGURATION
# ============================================================

# Fix the random seed used by Bilby.
# This makes the synthetic noise realization reproducible and helps
# you get similar results when rerunning the tutorial.
bilby.core.utils.random.seed(2026)

# Directory where Bilby will store output products.
# This folder will contain logs, JSON metadata, posterior samples,
# checkpoint files, and plots.
outdir = "gw_example_3det_theta_psi"

# A label attached to output filenames.
label = "binary_bh_three_detectors_theta_psi"

# Initialize Bilby's logger.
# This prints run information to screen and also writes it to a log file.
bilby.utils.setup_logger(outdir=outdir, label=label)


# ============================================================
# 2. ANALYSIS SETTINGS
# ============================================================

# Total duration of the data segment in seconds.
# A 4-second segment is a practical and common tutorial choice for a BBH signal.
duration = 4.0

# Sampling frequency in Hz.
# This means the time series has 1024 samples per second.
sampling_frequency = 1024.0
```

```
# Minimum analysis frequency in Hz.
# Frequencies below this are ignored in the likelihood and waveform.
minimum_frequency = 20.0

# Reference frequency used by the waveform model.
reference_frequency = 50.0


# ============================================================
# 3. TRUE INJECTION PARAMETERS
# ============================================================

# These are the physical parameters of the synthetic source that we inject
# into the detector network.
#
# Later, Bilby will try to infer a subset of them from the noisy data.
injection_parameters = dict(
    # Component masses in solar masses.
    mass_1=36.0,
    mass_2=29.0,

    # Dimensionless spin magnitudes.
    a_1=0.4,
    a_2=0.3,

    # Spin tilt angles relative to the orbital angular momentum.
    # We keep them fixed and aligned in this tutorial.
    tilt_1=0.0,
    tilt_2=0.0,

    # Additional precession angles.
    phi_12=0.0,
    phi_jl=0.0,

    # Luminosity distance in megaparsecs.
    luminosity_distance=500.0,

    # Inclination angle between the line of sight and orbital angular momentum.
    # This is now one of the free parameters in the inference.
    theta_jn=0.4,

    # Orbital phase at coalescence.
    phase=1.3,

    # Geocentric coalescence time in GPS seconds.
    geocent_time=1126259642.413,

    # Sky position.
    ra=1.375,
    dec=-1.2108,

    # Polarization angle.
    # This is also now a free parameter in the inference.
    psi=2.659,
)


# ============================================================
# 4. WAVEFORM MODEL
# ============================================================

# Waveform options passed to the source model.
waveform_arguments = dict(
    # A precessing phenomenological BBH waveform model.
```

```
    waveform_approximant="IMRPhenomPv2",

    # Frequency at which some source parameters are defined.
    reference_frequency=reference_frequency,

    # Lowest frequency used for waveform generation.
    minimum_frequency=minimum_frequency,
)

# Create the waveform generator.
# This object knows how to build the model waveform h(f) for any sampled
# parameter set.
waveform_generator = bilby.gw.waveform_generator.WaveformGenerator(
    duration=duration,
    sampling_frequency=sampling_frequency,
    frequency_domain_source_model=bilby.gw.source.lal_binary_black_hole,
    waveform_arguments=waveform_arguments,
)


# ============================================================
# 5. THREE-DETECTOR NETWORK
# ============================================================

# Build a detector network containing Hanford, Livingston, and Virgo.
# Bilby loads the detector geometry and noise model information for each.
interferometers = bilby.gw.detector.InterferometerList(["H1", "L1", "V1"])

# Loop over the detectors to generate synthetic Gaussian noise for each.
for ifo in interferometers:
    # Set the detector-specific lower analysis frequency.
    ifo.minimum_frequency = minimum_frequency

    # Populate this detector with simulated strain noise drawn from its PSD.
    ifo.set_strain_data_from_power_spectral_density(
        sampling_frequency=sampling_frequency,
        duration=duration,

        # Place the merger roughly at the center of the time segment.
        start_time=injection_parameters["geocent_time"] - duration / 2,
    )

# Inject the same astrophysical source into the detector network.
# Bilby automatically computes:
# - the detector response
# - time-of-flight delays between detectors
# - antenna pattern projections
interferometers.inject_signal(
    waveform_generator=waveform_generator,
    parameters=injection_parameters,
)


# ============================================================
# 6. PRIORS
# ============================================================

# Start from Bilby's standard BBH prior dictionary.
priors = bilby.gw.prior.BBHPriorDict()

# IMPORTANT:
# The default BBHPriorDict usually contains chirp_mass and mass_ratio
# as the mass sampling coordinates.
#
# In this tutorial we want to sample directly in mass_1 and mass_2.
```

```
# So we remove the redundant mass variables to avoid an error.
priors.pop("chirp_mass", None)
priors.pop("mass_ratio", None)

# ----------------------------------------------------------------
# Mass priors
# ----------------------------------------------------------------

# Uniform prior on the primary mass.
priors["mass_1"] = bilby.core.prior.Uniform(
    minimum=20.0,
    maximum=60.0,
    name="mass_1",
)

# Uniform prior on the secondary mass.
priors["mass_2"] = bilby.core.prior.Uniform(
    minimum=20.0,
    maximum=50.0,
    name="mass_2",
)

# ----------------------------------------------------------------
# Distance prior
# ----------------------------------------------------------------

# Uniform prior on luminosity distance.
# This is simple and explicit for tutorial purposes.
priors["luminosity_distance"] = bilby.core.prior.Uniform(
    minimum=100.0,
    maximum=1000.0,
    name="luminosity_distance",
)

# ----------------------------------------------------------------
# Coalescence time prior
# ----------------------------------------------------------------

# Narrow uniform prior around the true merger time.
priors["geocent_time"] = bilby.core.prior.Uniform(
    minimum=injection_parameters["geocent_time"] - 0.1,
    maximum=injection_parameters["geocent_time"] + 0.1,
    name="geocent_time",
)

# ----------------------------------------------------------------
# Sky-position priors
# ----------------------------------------------------------------

# Uniform prior in right ascension from 0 to 2*pi.
# The parameter is periodic, so we set boundary="periodic".
priors["ra"] = bilby.core.prior.Uniform(
    minimum=0.0,
    maximum=2.0 * np.pi,
    name="ra",
    boundary="periodic",
)

# Cosine prior in declination.
# This corresponds to an isotropic sky-position distribution.
priors["dec"] = bilby.core.prior.Cosine(name="dec")

# ----------------------------------------------------------------
# Inclination and polarization priors
# ----------------------------------------------------------------
```

```
# Sine prior for theta_jn.
# This is the standard isotropic prior for an orientation angle
# measured from 0 to pi.
#
# Why Sine?
# Because for random source orientations, the probability density
# is proportional to sin(theta).
priors["theta_jn"] = bilby.core.prior.Sine(name="theta_jn")

# Uniform prior in polarization angle psi.
# A common range is [0, pi], because polarization is periodic
# with period pi in this context.
priors["psi"] = bilby.core.prior.Uniform(
    minimum=0.0,
    maximum=np.pi,
    name="psi",
    boundary="periodic",
)

# ------------------------------------------------------------
# Free and fixed parameters
# ------------------------------------------------------------

# Define which parameters we want to sample in this tutorial.
free_parameters = {
    "mass_1",
    "mass_2",
    "luminosity_distance",
    "geocent_time",
    "ra",
    "dec",
    "theta_jn",
    "psi",
}

# All other parameters that appear in the prior dictionary and also
# exist in the injection dictionary are fixed to their injected values.
#
# This means Bilby will not sample them.
for key in list(priors.keys()):
    if key not in free_parameters and key in injection_parameters:
        priors[key] = injection_parameters[key]

# Print a diagnostic check for redundant coordinates.
# This should be False.
print("Has redundant keys?", priors.test_has_redundant_keys())

# Validate the prior against the waveform duration and frequency cutoff.
# This is a good practice in CBC analyses.
priors.validate_prior(
    duration=duration,
    minimum_frequency=minimum_frequency,
)


# ============================================================
# 7. LIKELIHOOD
# ============================================================

# Build the standard transient GW likelihood for Gaussian detector noise.
# Because we pass a three-detector network, the likelihood is the joint
# likelihood over H1, L1, and V1.
likelihood = bilby.gw.likelihood.GravitationalWaveTransient(
    interferometers=interferometers,
```

```
    waveform_generator=waveform_generator,
)


# ============================================================
# 8. RUN THE SAMPLER
# ============================================================

# Run dynesty nested sampling.
# This explores the posterior distribution over the free parameters.
result = bilby.run_sampler(
    likelihood=likelihood,
    priors=priors,
    sampler="dynesty",

    # Number of live points.
    # Since we now sample 8 parameters, this run is more expensive
    # than the previous examples.
    #
    # 500 is fine for a tutorial, but if the posterior looks rough,
    # increasing this to 1000 can help.
    nlive=500,

    # Evidence stopping criterion.
    dlogz=0.5,

    outdir=outdir,
    label=label,

    # Store the true injected values in the result object.
    # This is useful for making plots with truth markers.
    injection_parameters=injection_parameters,

    # Ask Bilby to generate standard output plots automatically.
    plot=True,

    # Allow checkpoint resume.
    resume=True,

    # Clean temporary files when possible.
    clean=True,
)


# ============================================================
# 9. PRINT A SUMMARY
# ============================================================

# Print Bilby's summary of the run.
print(result)

# Print simple descriptive statistics for the sampled parameters.
print(
    result.posterior[
        [
            "mass_1",
            "mass_2",
            "luminosity_distance",
            "geocent_time",
            "ra",
            "dec",
            "theta_jn",
            "psi",
        ]
    ].describe()
```

```
)


# ============================================================
# 10. CORNER PLOT
# ============================================================

# Make a corner plot of the posterior samples.
# Because injection_parameters were passed to run_sampler,
# Bilby can show the true values on the plot.
result.plot_corner()
```

6.4.1 Comments and Interpretation

This example represents a useful intermediate step in learning how to use `Bilby` for gravitational-wave parameter estimation. It is still simple enough to run quickly and to understand line by line, but it already contains most of the conceptual ingredients of a realistic analysis.

The most important change compared with a single-detector example is that the inference is now performed with a **network of detectors**. In this script the network consists of

$$\text{H1 (LIGO Hanford)}, \qquad \text{L1 (LIGO Livingston)}, \qquad \text{V1 (Virgo)}.$$

A detector network allows the data to constrain not only the intrinsic structure of the waveform but also the geometry of the source. In particular, the network can exploit differences in

- arrival time of the signal at each detector,
- antenna response of each detector to the two gravitational-wave polarizations,
- orientation of the detectors on Earth.

These differences encode information about the sky location, the source orientation, and the polarization of the signal.

Free parameters in the example The script samples eight parameters:

`mass_1`, `mass_2`, `luminosity_distance`, `geocent_time`, `ra`, `dec`, `theta_jn`, `psi`.

These parameters play different physical roles.

Mass parameters The component masses mainly determine the *phase evolution* of the waveform. They affect the inspiral rate, the merger frequency scale, and the overall time–frequency structure of the signal. Consequently, masses are usually constrained by the detailed shape of the waveform.

Luminosity distance The parameter `luminosity_distance` primarily controls the amplitude of the signal. However, distance is not measured independently. It is correlated with the inclination angle of the binary because a more favorably oriented system can appear stronger than a less favorably oriented one at the same distance.

Coalescence time The parameter `geocent_time` defines the merger time at the Earth's center. In a detector network, small differences in arrival time between detectors provide information about the sky location.

Sky position The parameters `ra` and `dec` specify the source position on the celestial sphere. With a single detector these parameters are poorly constrained, but a network can restrict the possible sky region using arrival-time differences and antenna-pattern variations.

Inclination and polarization The inclination angle `theta_jn` describes the orientation of the binary relative to the observer. It strongly affects the relative amplitudes of the two gravitational-wave polarizations.

The polarization angle `psi` rotates the polarization basis relative to the detectors. This parameter becomes measurable only in a detector network because different detectors project the polarizations differently.

Role of the priors The priors used in the script reflect simple geometric assumptions:

- a sine prior for `theta_jn`, corresponding to isotropic orientations,
- a cosine prior for `dec`, corresponding to isotropic sky positions,
- periodic priors for angular variables such as `ra` and `psi`.

These choices encode physical information about random orientations and sky locations.

Expected qualitative results Students should expect that

- the masses and coalescence time are relatively well constrained,
- the sky location is restricted but not perfectly localized,
- the luminosity distance shows a broader posterior,
- a correlation appears between distance and inclination,
- the polarization angle may remain weakly constrained.

Such behavior is typical in gravitational-wave parameter estimation and reflects both detector geometry and physical degeneracies.

6.4.2 Questions

The following questions can help guide discussion and understanding after running the example.

Understanding the code

1. Why do we use a detector network instead of a single interferometer?
2. Why is independent noise generated for each detector?
3. Why is the same astrophysical signal injected into all detectors?
4. Why is the merger time placed near the center of the data segment?
5. Why must `chirp_mass` and `mass_ratio` be removed when sampling directly in `mass_1` and `mass_2`?

Physics interpretation

1. Which parameters are mainly constrained by the phase evolution of the waveform?
2. Which parameters are constrained primarily by the detector network geometry?
3. Why can a network constrain sky location better than a single detector?
4. Why are luminosity distance and inclination often correlated?
5. Why does the polarization angle become meaningful only in a multi-detector analysis?

Bayesian interpretation

1. What is the difference between fixing a parameter and giving it a narrow prior?
2. Why is it useful to validate the prior against the signal duration?
3. What does a tilted contour in a corner plot represent?
4. Does a broad posterior always indicate poor data quality, or can it reflect a physical degeneracy?

Conceptual questions

1. Which parameter in this example do you expect to be measured most precisely?
2. Which parameter is likely to be measured least precisely?
3. Why might the sky localization still cover a relatively large region even with three detectors?

6.4.3 Suggested Experiments

The following exercises help illustrate how different elements of the analysis affect the results.

Experiment 1: Compare detector networks Run the analysis with

- one detector (H1),
- two detectors (H1 and L1),
- three detectors (H1, L1, and V1).

Compare the posterior distributions for sky location and polarization. This illustrates how detector networks improve parameter constraints.

Experiment 2: Fix and free the inclination Run the code once with `theta_jn` fixed to its injected value and once with it sampled. Compare the posterior for luminosity distance. This demonstrates the well-known distance–inclination degeneracy.

Experiment 3: Fix and free the polarization angle Repeat the analysis with `psi` fixed and then free. Observe how the constraints on other extrinsic parameters change.

Experiment 4: Change the injected inclination Modify the injected value of `theta_jn` and repeat the analysis for several orientations (for example nearly face-on and nearly edge-on). Compare the resulting posterior distributions.

Experiment 5: Change the source distance Repeat the injection at different luminosity distances. Examine how the signal-to-noise ratio affects the posterior width and sky localization.

Experiment 6: Modify the priors Experiment with broader or narrower priors for distance or masses. Observe how these choices affect runtime, sampling efficiency, and posterior structure.

Experiment 7: Use alternative mass parameters Rewrite the analysis so that the sampled parameters are `chirp_mass` and `mass_ratio` instead of the component masses. Compare sampling efficiency and interpretability of the results.

Experiment 8: Increase dimensionality Allow additional parameters such as spin magnitude or orbital phase to vary. Observe how increasing the dimensionality affects runtime and posterior correlations.

Experiment 9: Change sampler settings Modify the nested-sampling parameters (for example the number of live points). Compare runtime and smoothness of the posterior distributions.

Experiment 10: Remove the truth markers Run the analysis without passing the injection parameters to the sampler. Compare the resulting plots and discuss why truth markers are useful for simulated data but unavailable in real observations.

SECTION 7

Wrap-Up and Q&A (17:30–18:00)

Recap

- Bayesian workflow: priors + likelihood → posterior samples.
- Bilby pattern reused from toy model to GW inference.
- Practical skills: running samplers, reading corner plots, diagnosing runtime and prior effects.

Common pitfalls and tips

- Priors too narrow or excluding the true region; priors too broad leading to slow convergence.
- Mismatched parameter names between priors, likelihood, and waveform model.
- Missing LALSuite components or incompatible installations; verify imports early.
- Use saved outputs (results JSON, plots) for reproducibility and sharing.

Pointers for self-study (after the tutorial)

- Real data analyses using open strain data (GWOSC), replacing injection with data fetch and PSD estimation.
- Alternative samplers and sampler settings; cross-checking results.
- Scaling up with `bilby_pipe` and cluster workflows.
- Hierarchical inference with `bilby.hyper` (population inference).

SECTION A

References and Resources (Suggested Reading)

- Bilby documentation (installation, examples, API reference): https://lscsoft.docs.ligo.org/bilby/
- Bilby paper:Gregory Ashton et al. "Bilby: A user-friendly Bayesian inference library for gravitational-wave astronomy". In: *Astrophys. J. Suppl.* 241.2 (2019), p. 27. DOI: 10.3847/1538-4365/ab06fc. arXiv: 1811.02042 [astro-ph.IM].
- Validation/application study: Isobel M. Romero-Shaw et al. "Bayesian inference for compact binary coalescences with bilby: validation and application to the first LIGO-Virgo catalog". In: *Mon. Not. Roy. Astron. Soc.* 499.3 (2020), pp. 3295–3319. DOI: 10.1093/mnras/staa2850. arXiv: 2006.00714 [astro-ph.IM].
- GW Open Science Center (GWOSC) workshops and data access: https://www.gw-openscience.org/

Questions and Answers

A.1 Defining Priors in the `bilby.core` Module

In the **Bilby** library, priors are primarily handled through the **`bilby.core`** module, which provides the essential infrastructure for general Bayesian inference. According to the sources, the process of defining priors is designed to be **user-friendly**, offering a **straightforward syntax** that is accessible even to beginners.

Based on the documentation and the library's architecture, here is how priors are defined and managed:

- **Fundamental Role:** Priors are a core component of the parameter estimation workflow, used to represent the state of knowledge about a parameter **before** the data is considered.
- **General Infrastructure:** The **`bilby.core`** API includes a variety of built-in, standard prior distributions suitable for a wide range of general inference problems.
- **Extensive Examples:** The library includes dedicated learning resources, such as the **"Prior Distributions with Bilby"** example, which demonstrates how to implement and visualize these distributions.
- **Specialization:** While **`bilby.core`** manages general distributions, Bilby also provides specialized **transient gravitational-wave priors** (e.g., for mass, spin, and distance) through the **`bilby.gw`** module.

The modularity of the library allows users to easily select from existing distributions or define their own to suit specific signal models or scientific requirements.

Implementing Custom Likelihood Functions in Bilby

In the **Bilby** library, implementing custom likelihood functions is a core feature designed to allow users to extend the library's capabilities beyond standard gravitational-wave models. The architecture is highly modular, with the **`bilby.core`** module providing the base infrastructure for general Bayesian inference problems.

Based on the sources, you can implement custom likelihoods by following these principles:

- **General Inference Framework:** While Bilby is optimized for compact binary coalescence events, its **`bilby.core`** API is built to handle more general statistical problems. This allows users to define likelihood functions for any model where a probability of the data given the parameters can be calculated.
- **Extending Signal Models:** The library provides a suite of examples specifically illustrating how to **implement new likelihood functions** and how to change the underlying signal model. These examples cover diverse cases such as:
  - **Supernovae** and the remnants of **binary neutron star mergers**.
  - **General Curve Fitting:** Specifically, examples exist for fitting models to data that contain both **x and y errors**.
- **Documentation Resources:** Detailed guidance is available in the dedicated **"Likelihood"** section of the documentation, which covers the basics of parameter estimation and the interface between the likelihood and the chosen sampler.

By utilizing the provided examples and the core API, researchers can adapt Bilby to virtually any Bayesian inference task by defining the mathematical relationship between their model and their data within a new likelihood class.

Principal Samplers Supported in `bilby.core`

The **Bilby** library serves as a modular interface to several sampling algorithms, allowing users to perform Bayesian parameter estimation across different signal models. While the library is designed to interface with multiple external sampling packages, the documentation highlights two primary samplers through dedicated guides and API sections:

- **Dynesty:** A nested sampling package that is widely used within the Bilby framework for evidence calculation and posterior estimation.
- **Bilby MCMC:** The library's own Markov Chain Monte Carlo implementation, which has a specific API (`bilby.bilby_mcmc`) and a comprehensive user guide.

Additionally, Bilby provides tools and examples that allow users to **compare different samplers**, helping them identify the most efficient algorithm for their specific scientific problem, whether it involves gravitational-wave data or general inference.

A.2 The Difference Between `bilby.core` and `bilby.gw` Modules

In the **Bilby** library, the architecture is divided into specialized modules to balance general-purpose statistical utility with domain-specific functionality for gravitational-wave astronomy. The primary distinction between the `bilby.core` and `bilby.gw` modules lies in their scope and the types of problems they are designed to solve:

- `bilby.core`**: General Bayesian Infrastructure** The `bilby.core` module provides the fundamental engine for Bayesian inference. It is designed for **general problems** beyond astrophysics, providing the base classes and logic for:
  - Defining general **prior** distributions.
  - Implementing generic **likelihood** functions, such as those used for curve fitting with x and y errors.
  - Interfacing with various **sampling** algorithms like Dynesty or the internal Bilby MCMC.
- `bilby.gw`**: Specialized Gravitational-Wave Tools** The `bilby.gw` module is specifically tailored for the needs of **gravitational-wave (GW) astronomy**. It extends the core functionality with specialized tools for:
  - Handling **transient gravitational-wave data I/O** and interferometric data from detectors like LIGO and Virgo.
  - Implementing **detector models** and signal models for compact binary coalescences (CBC).
  - Providing specialized **transient gravitational-wave priors** for parameters like mass, spin, and distance.

In summary, while `bilby.core` serves as the universal library for any Bayesian inference task, `bilby.gw` builds upon that foundation to provide the expert-level infrastructure required for analyzing gravitational-wave signals.

A.3 The Role of Dynesty in Bilby Sampling

In the **Bilby** library, **Dynesty** serves as one of the primary sampling algorithms used to perform Bayesian parameter estimation. Its role is to explore the parameter space defined by the user's priors and likelihood function to produce posterior distributions and calculate Bayesian evidence.

According to the documentation, Dynesty's integration into Bilby is characterized by several key features:

- **Dedicated Support:** The library includes a specific **"Dynesty Guide"** to help users navigate the settings and optimization of this sampler.
- **Performance Evaluation:** Bilby provides learning resources, such as the **"Compare samplers"** example, which allows users to benchmark Dynesty against other available algorithms like **Bilby MCMC**.

- **Broad Application:** While Bilby is optimized for gravitational-wave data, the Dynesty interface within `bilby.core` is applicable to general inference problems, such as fitting models to data with complex error structures.

As a nested sampling algorithm, Dynesty is particularly valuable in the context of Bilby for tasks requiring model selection, as it provides a robust estimate of the evidence alongside the parameter posteriors.

A.4 Tools and Examples for Visualizing Results

The sources indicate that **Bilby** is designed with a user-friendly interface that includes specific tools and examples to help users visualize the results of their Bayesian parameter estimation. Within the documentation's "Examples" section, there is a dedicated guide titled **"Visualising the results,"** which provides users with the necessary framework to interpret their findings. Additionally, the documentation contains a section on **"Bilby output,"** which covers how the library handles and presents the data generated during the inference process.

A.5 Visualizing Priors and Distributions

Before and during the sampling process, visualizing the parameter space is crucial. The sources highlight that Bilby provides an example for **"Prior Distributions with Bilby,"** allowing users to visualize and understand their chosen priors before they are applied to the data. Furthermore, for more complex analyses, the library's hierarchical modeling functionality allows for the visualization of population-level characteristics, such as the **shape of the black hole mass distribution** inferred from multiple observations.

A.6 Comparative and Output Tools

To assist in selecting the best approach for a given problem, Bilby includes a **"Compare samplers"** example, which likely involves visual comparisons of posterior distributions produced by different algorithms. The library's output is structured to facilitate these comparisons and to integrate seamlessly with the user's workflow for both gravitational-wave data and general inference problems.

References

Ashton, Gregory et al. "Bilby: A user-friendly Bayesian inference library for gravitational-wave astronomy". In: *Astrophys. J. Suppl.* 241.2 (2019), p. 27. DOI: 10.3847/1538-4365/ab06fc. arXiv: 1811.02042 [astro-ph.IM].

Romero-Shaw, Isobel M. et al. "Bayesian inference for compact binary coalescences with bilby: validation and application to the first LIGO-Virgo catalog". In: *Mon. Not. Roy. Astron. Soc.* 499.3 (2020), pp. 3295–3319. DOI: 10.1093/mnras/staa2850. arXiv: 2006.00714 [astro-ph.IM].