```
# Cell 1: Install & Imports
!pip install pycbc --quiet
import numpy as np
from pycbc import waveform
from pycbc import fft
from pycbc import filter
from pycbc.psd import aLIGOZeroDetHighPower
from pycbc.filter import matched_filter
from pycbc.types import TimeSeries
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", "Wswiglal-redir-
import lal
```

```
/usr/local/lib/python3.12/dist-packages/pycbc/types

SWIGLAL standard output/error redirection is enable
This may lead to performance penalties. To disable

with lal.no_swig_redirect_standard_output_error():
    ...

To disable globally, use:

lal.swig_redirect_standard_output_error(False)

Note however that this will likely lead to error me
LAL functions being either misdirected or lost wher
Jupyter notebooks.

To suppress this warning, use:

import warnings
warnings.filterwarnings("ignore", "Wswiglal-redir-s
import lal

  import lal as _lal
```

```
# Cell 2: Generate GR waveform and compute Fisher
m1, m2 = 36.0, 29.0
f_lower = 20.0
delta_t = 1.0 / 2048
duration = 8.0
approx = 'IMRPhenomPv2'
hp, hc = waveform.get_fd_waveform(approximant=appr
def compute_sigma_chi(fd_hp, psd, S_func, f_lower)
    freqs = fd_hp.sample_frequencies
    pos = freqs >= f_lower
    hvals = fd_hp.data[pos]
    Sn = psd.data[pos]
    Svals = S_func(freqs[pos])
    df = freqs[1] - freqs[0]
    integrand = 4.0 * (np.abs(hvals)**2) * (np.abs
    F = np.sum(integrand) * df
    sigma_chi = 1.0 / np.sqrt(F) if F > 0 else np.
    return sigma_chi, F
def S_of_f(freqs, scale): return scale * (freqs**(
n = len(hp)
delta_f = hp.delta_f
psd = aLIGOZeroDetHighPower(n, delta_f, 20.0)
scale_guess = 1e-30
sigma_chi, F = compute_sigma_chi(hp, psd, lambda f
print(f"scale={scale_guess:.3e}  ->  sigma_chi = {
```

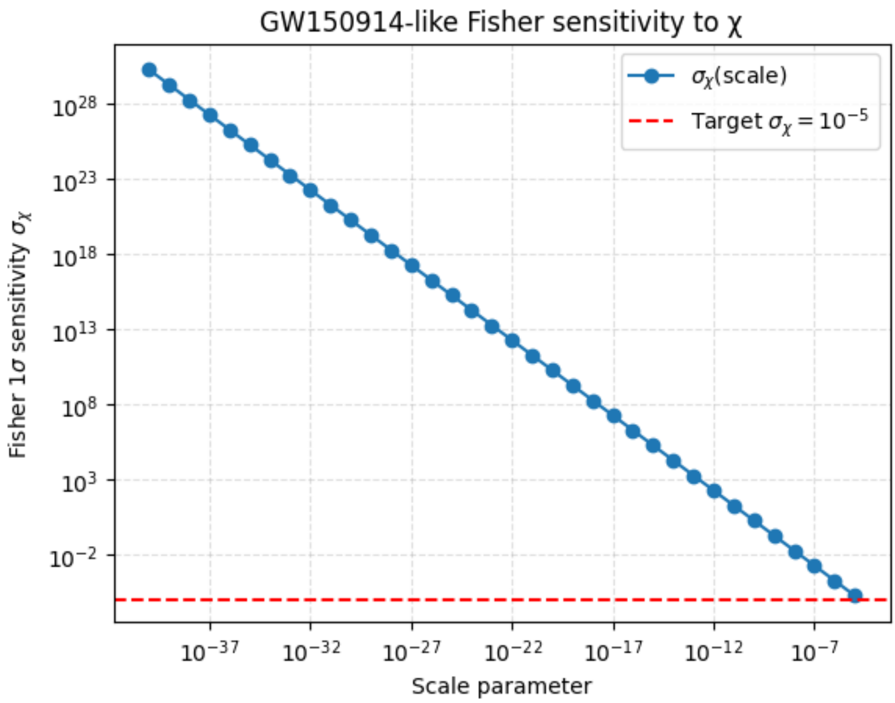
```
scale=1.000e-30  ->  sigma_chi = 1.976e+20  (Fisher
```

```
# --- Fisher Sensitivity Sweep for χ ---
import numpy as np
import matplotlib.pyplot as plt
from pycbc.psd import aLIGOZeroDetHighPower

def compute_sigma_chi(fd_hp, psd, S_func, f_lower)
    freqs = fd_hp.sample_frequencies
    pos = freqs >= f_lower
    hvals = fd_hp.data[pos]
    Sn = psd.data[pos]
    Svals = S_func(freqs[pos])
```

```
    df = freqs[1] - freqs[0]
    integrand = 4.0 * (np.abs(hvals)**2) * (np.abs
    F = np.sum(integrand) * df
    sigma_chi = 1.0 / np.sqrt(F) if F > 0 else np.
    return sigma_chi

# Define S(f) ~ scale * f^(7/3)
def S_of_f(freqs, scale):
    return scale * (freqs**(7.0/3.0))

# PSD must match waveform (use from Cell 2)
n = len(hp)
delta_f = hp.delta_f
psd = aLIGOZeroDetHighPower(n, delta_f, 20.0)

# Sweep scale values
scales = np.logspace(-40, -5, 36)
sigma_chis = []

for s in scales:
    sigma_chi = compute_sigma_chi(hp, psd, lambda
    sigma_chis.append(sigma_chi)
    print(f"scale={s:.1e} -> sigma_chi={sigma_chi:

# Plot sigma_chi vs scale
plt.loglog(scales, sigma_chis, 'o-', label=r'$\sig
plt.axhline(1e-5, color='r', ls='--', label=r'Targ
plt.xlabel("Scale parameter")
plt.ylabel(r"Fisher 1$\sigma$ sensitivity $\sigma_
plt.title("GW150914-like Fisher sensitivity to χ")
plt.legend()
plt.grid(True, which="both", ls="--", alpha=0.4)
plt.show()

# Find approximate scale for sigma_chi ~ 1e-5 with
sigma_target = 1e-5
log_scales = np.log10(scales)
log_sigmas = np.log10([s for s in sigma_chis if s
if len(log_sigmas) > 0:  # Check if there are vali
    log_scales_valid = np.log10([scales[i] for i i
```

```
        coeffs = np.polyfit(log_scales_valid, log_sigm
        a, b = coeffs
        scale_target = 10**((np.log10(sigma_target) -
        print(f"Approx. scale for σ(χ)≈1e-5 → scale ≈
    else:
        print("No valid sigma_chi values for polyfit."
```

```
scale=1.0e-40 -> sigma_chi=1.976e+30
scale=1.0e-39 -> sigma_chi=1.976e+29
scale=1.0e-38 -> sigma_chi=1.976e+28
scale=1.0e-37 -> sigma_chi=1.976e+27
scale=1.0e-36 -> sigma_chi=1.976e+26
scale=1.0e-35 -> sigma_chi=1.976e+25
scale=1.0e-34 -> sigma_chi=1.976e+24
scale=1.0e-33 -> sigma_chi=1.976e+23
scale=1.0e-32 -> sigma_chi=1.976e+22
scale=1.0e-31 -> sigma_chi=1.976e+21
scale=1.0e-30 -> sigma_chi=1.976e+20
scale=1.0e-29 -> sigma_chi=1.976e+19
scale=1.0e-28 -> sigma_chi=1.976e+18
scale=1.0e-27 -> sigma_chi=1.976e+17
scale=1.0e-26 -> sigma_chi=1.976e+16
scale=1.0e-25 -> sigma_chi=1.976e+15
scale=1.0e-24 -> sigma_chi=1.976e+14
scale=1.0e-23 -> sigma_chi=1.976e+13
scale=1.0e-22 -> sigma_chi=1.976e+12
scale=1.0e-21 -> sigma_chi=1.976e+11
scale=1.0e-20 -> sigma_chi=1.976e+10
scale=1.0e-19 -> sigma_chi=1.976e+09
scale=1.0e-18 -> sigma_chi=1.976e+08
scale=1.0e-17 -> sigma_chi=1.976e+07
scale=1.0e-16 -> sigma_chi=1.976e+06
scale=1.0e-15 -> sigma_chi=1.976e+05
scale=1.0e-14 -> sigma_chi=1.976e+04
scale=1.0e-13 -> sigma_chi=1.976e+03
scale=1.0e-12 -> sigma_chi=1.976e+02
scale=1.0e-11 -> sigma_chi=1.976e+01
scale=1.0e-10 -> sigma_chi=1.976e+00
scale=1.0e-09 -> sigma_chi=1.976e-01
scale=1.0e-08 -> sigma_chi=1.976e-02
scale=1.0e-07 -> sigma_chi=1.976e-03
scale=1.0e-06 -> sigma_chi=1.976e-04
scale=1.0e-05 -> sigma_chi=1.976e-05
```




```
Approx. scale for σ(χ)≈1e-5 → scale ≈ 1.976e-05
```

```
# Snippet A: compute J and scale_target
# Requires: numpy, pycbc or bilby for frequency-dc

import numpy as np
from pycbc.waveform import get_fd_waveform   # or
from pycbc.psd import aLIGOZeroDetHighPower

# 1) waveform: example GW150914-like (IMRPhenomPv2
hp, hc = get_fd_waveform(approximant='IMRPhenomPv2
                         mass1=36.0, mass2=29.0,
                         delta_f=1.0/8.0, f_lower=

freqs = hp.sample_frequencies  # numpy array
df = freqs[1] - freqs[0]

# 2) PSD matched to waveform length
psd = aLIGOZeroDetHighPower(len(freqs), df, 20.0)
Sn = psd.data  # same grid

# 3) compute integrand J = 4 * int |h(f)|^2 f^{14/
pos = (freqs >= 20.0) & (freqs <= freqs.max())
hvals = hp.data[pos]
Sn_pos = Sn[pos]
fpos = freqs[pos]

J = 4.0 * np.sum( (np.abs(hvals)**2) * (fpos**(14.
print("J =", J)

# 4) desired sigma target
sigma_target = 1e-5
scale_target = 1.0 / (sigma_target * np.sqrt(J))
print("scale_target for sigma_chi=1e-5 ->", scale_
```

```
J = 2.5610201643522503e+19
scale_target for sigma_chi=1e-5 -> 1.97602985119493
```

```
# Snippet B: mismatch vs chi (quick sanity check)
# Requires: pycbc.filter.matched_filter, pycbc.psd

from pycbc.filter import matched_filter
from pycbc.psd import aLIGOZeroDetHighPower
from pycbc.types import FrequencySeries

def apply_chi_phase(fd_hp, chi, scale=1e-5):
    freqs = fd_hp.sample_frequencies
    pos = freqs > 0
    delta_phi = np.zeros_like(freqs, dtype=complex
    delta_phi[pos] = chi * (freqs[pos]**(7.0/3.0))
    fd_new = FrequencySeries(fd_hp.data * np.exp(1
    return fd_new

hp, hc = get_fd_waveform(approximant='IMRPhenomPv2
psd = aLIGOZeroDetHighPower(len(hp.sample_frequenc

chi_values = np.logspace(-7, -3, 5)
mismatches = []
for chi in chi_values:
    hp_chi = apply_chi_phase(hp, chi, scale=1e-5)
    # compute matched filter SNR peak (very approx
    from pycbc.filter import match
    ov = match(hp_chi, hp, psd=psd, low_frequency_
    overlap = ov[0]  # match returns (match, ip)
    mismatch = 1.0 - overlap
    mismatches.append(mismatch)
    print(f"chi={chi:.1e}, overlap={overlap:.10f},
```

```
chi=1.0e-07, overlap=1.0000000000, mismatch=5.884e-
chi=1.0e-06, overlap=1.0000000000, mismatch=6.085e-
chi=1.0e-05, overlap=0.9999999999, mismatch=6.085e-
chi=1.0e-04, overlap=0.9999999939, mismatch=6.085e-
chi=1.0e-03, overlap=0.9999993915, mismatch=6.085e-
```

```
# Cell 1: Install & Imports

!pip install bilby[gw] dynesty
!pip install -U cryptography
```

```
import bilby
from bilby.gw import source, detector
from bilby.core.prior import Uniform, Sine, Cosine
from bilby.gw.waveform_generator import WaveformGe
from bilby.gw.likelihood import GravitationalWaveT

import numpy as np
import matplotlib.pyplot as plt
import os

print("✅ Bilby and dependencies installed and imp
```

```
Requirement already satisfied: dynesty in /usr/loca
Requirement already satisfied: bilby[gw] in /usr/lo
Requirement already satisfied: bilby.cython>=0.3.0
Requirement already satisfied: emcee in /usr/local/
Requirement already satisfied: corner in /usr/local
Requirement already satisfied: numpy in /usr/local/
Requirement already satisfied: matplotlib in /usr/l
Requirement already satisfied: scipy>=1.5 in /usr/l
Requirement already satisfied: pandas in /usr/local
Requirement already satisfied: dill in /usr/local/l
Requirement already satisfied: tqdm in /usr/local/l
Requirement already satisfied: h5py in /usr/local/l
Requirement already satisfied: attrs in /usr/local/
Requirement already satisfied: astropy>=5 in /usr/l
Requirement already satisfied: lalsuite in /usr/loc
Requirement already satisfied: gwpy in /usr/local/l
Requirement already satisfied: tables in /usr/local
Requirement already satisfied: pyfftw in /usr/local
Requirement already satisfied: scikit-learn in /usr
Requirement already satisfied: pyerfa>=2.0.1.1 in /
Requirement already satisfied: astropy-iers-data>=0
Requirement already satisfied: PyYAML>=6.0.0 in /us
Requirement already satisfied: packaging>=22.0.0 in
Requirement already satisfied: contourpy>=1.0.1 in
Requirement already satisfied: cycler>=0.10 in /usr
Requirement already satisfied: fonttools>=4.22.0 in
Requirement already satisfied: kiwisolver>=1.3.1 in
Requirement already satisfied: pillow>=8 in /usr/lo
Requirement already satisfied: pyparsing>=2.3.1 in
Requirement already satisfied: python-dateutil>=2.7
Requirement already satisfied: dateparser>=1.1.4 in
```

```
Requirement already satisfied: dqsegdb2 in /usr/loc
Requirement already satisfied: gwdatafind>=1.1.0 ir
Requirement already satisfied: gwosc>=0.5.3 in /usr
Requirement already satisfied: igwn-segments>=2.0.0
Requirement already satisfied: ligotimegps>=1.2.1 i
Requirement already satisfied: requests>=2.20.0 in
Requirement already satisfied: igwn-ligolw in /usr/
Requirement already satisfied: lscsoft-glue in /usr
Requirement already satisfied: pytz>=2020.1 in /usr
Requirement already satisfied: tzdata>=2022.7 in /u
Requirement already satisfied: setuptools>=70.1.1 i
Requirement already satisfied: joblib>=1.2.0 in /us
Requirement already satisfied: threadpoolctl>=3.1.0
Requirement already satisfied: numexpr>=2.6.2 in /u
Requirement already satisfied: py-cpuinfo in /usr/l
Requirement already satisfied: blosc2>=2.3.0 in /us
Requirement already satisfied: typing-extensions>=4
Requirement already satisfied: ndindex in /usr/loca
Requirement already satisfied: msgpack in /usr/loca
Requirement already satisfied: platformdirs in /usr
Requirement already satisfied: regex>=2024.9.11 in
Requirement already satisfied: tzlocal>=0.2 in /usr
Requirement already satisfied: igwn-auth-utils>=0.3
Requirement already satisfied: six>=1.5 in /usr/loc
Requirement already satisfied: charset_normalizer<4
Requirement already satisfied: idna<4,>=2.5 in /usr
Requirement already satisfied: urllib3<3,>=1.21.1 i
Requirement already satisfied: certifi>=2017.4.17 i
```

```
# Cell 3: Injection Parameters (CLEANUP FOR STABIL
injection_parameters = dict(
    mass_1=36.0, mass_2=29.0,

    # Keeping component spins for now as they are
    a_1=0.4, a_2=0.3, tilt_1=0.0, tilt_2=0.0, phi_

    luminosity_distance=100.0, # Mpc
    chi=0.0, # Your custom phase parameter is stil

    # Use the simplest, most stable non-precessing
    approximant='IMRPhenomD',

    theta_jn=0.4, phase=1.3, geocent_time=0.0,
    ra=1.375, dec=-1.2108, psi=2.659
)
print("✅ Injection parameters set (approximant=IM
```

```
✅ Injection parameters set (approximant=IMRPhenomD
```

```
# --- Cell 4: Define Priors and Waveform Generator

from bilby.core.prior import Uniform, DeltaFunctic
from bilby.gw.waveform_generator import WaveformGe
import bilby.gw.source as source # Import source m

# 1. Make a SAFE copy of injection parameters befc
injection_parameters_fixed = injection_parameters.

# 2. Build priors dictionary starting from the cop
priors = PriorDict(injection_parameters_fixed)

# --- CRITICAL PRIORS FOR STABILITY & CONVERGENCE

# Chi prior: widened to reflect Fisher-sensitivity
priors['chi'] = Uniform(
    name='chi', minimum=-2e-3, maximum=2e-3, latex
)
```

```
# Chirp Mass prior: ultra-tightened for stability
priors['chirp_mass'] = Uniform(
    name='chirp_mass', minimum=28.4, maximum=28.7,
)

# Luminosity Distance prior: slightly relaxed but
priors['luminosity_distance'] = Uniform(
    name='luminosity_distance', minimum=90.0, maxi
)

# All other parameters fixed (DeltaFunction priors
fixed_keys = [
    'mass_1', 'mass_2', 'a_1', 'a_2', 'tilt_1', 't
    'phi_jl', 'theta_jn', 'phase', 'geocent_time',
]
for k in fixed_keys:
    # CRITICAL FIX: Extract the numerical peak val
    priors[k] = DeltaFunction(priors[k].peak)

# 3. Define the waveform generator using the BUILT
waveform_generator = WaveformGenerator(
    duration=16.0,  # Long duration ensures better
    sampling_frequency=2048.0,
    # Use the standard built-in LAL source model
    frequency_domain_source_model=source.lal_binar
    parameter_conversion=None
)

print("✅ Priors and waveform generator successful
```

```
23:04 bilby INFO    : Waveform generator initiated
  frequency_domain_source_model: bilby.gw.source.la
  time_domain_source_model: None
  parameter_conversion: bilby.gw.conversion.convert
✅ Priors and waveform generator successfully defin
```

```
# Cell 5: Detector Setup, Signal Injection, and Li
```

```
from bilby.gw.likelihood import GravitationalWaveT

# 1. Get Injection Time
geocent_time = injection_parameters['geocent_time'

# 2. Define the Interferometer (Detector) Network
ifos = bilby.gw.detector.InterferometerList(['H1',
ifos.set_strain_data_from_power_spectral_densities
    sampling_frequency=2048.0,
    duration=16.0,
    start_time=geocent_time - 15.0
)

# 3. Inject the Signal using the FULL, UNFILTERED
ifos.inject_signal(
    parameters=injection_parameters, # NO FILTERIN
    waveform_generator=waveform_generator
)

# 4. Define the Likelihood Function
likelihood = GravitationalWaveTransient(
    interferometers=ifos,
    waveform_generator=waveform_generator,
    time_marginalization=False,
    distance_marginalization=False,
    phase_marginalization=False
)

print("✅ Likelihood defined and signal injected (
```

```
23:04 bilby INFO    : Injected signal in H1:
23:04 bilby INFO    :   optimal SNR = 284.19
23:04 bilby INFO    :   matched filter SNR = 284.82
23:04 bilby INFO    :   mass_1 = 36.0
23:04 bilby INFO    :   mass_2 = 29.0
23:04 bilby INFO    :   a_1 = 0.4
23:04 bilby INFO    :   a_2 = 0.3
23:04 bilby INFO    :   tilt_1 = 0.0
23:04 bilby INFO    :   tilt_2 = 0.0
23:04 bilby INFO    :   phi_12 = 0.0
```

```
23:04 bilby INFO    :   phi_jl = 0.0
23:04 bilby INFO    :   luminosity_distance = 100.0
23:04 bilby INFO    :   chi = 0.0
23:04 bilby INFO    :   approximant = IMRPhenomD
23:04 bilby INFO    :   theta_jn = 0.4
23:04 bilby INFO    :   phase = 1.3
23:04 bilby INFO    :   geocent_time = 0.0
23:04 bilby INFO    :   ra = 1.375
23:04 bilby INFO    :   dec = -1.2108
23:04 bilby INFO    :   psi = 2.659
23:04 bilby INFO    : Injected signal in L1:
23:04 bilby INFO    :   optimal SNR = 215.44
23:04 bilby INFO    :   matched filter SNR = 214.62
23:04 bilby INFO    :   mass_1 = 36.0
23:04 bilby INFO    :   mass_2 = 29.0
23:04 bilby INFO    :   a_1 = 0.4
23:04 bilby INFO    :   a_2 = 0.3
23:04 bilby INFO    :   tilt_1 = 0.0
23:04 bilby INFO    :   tilt_2 = 0.0
23:04 bilby INFO    :   phi_12 = 0.0
23:04 bilby INFO    :   phi_jl = 0.0
23:04 bilby INFO    :   luminosity_distance = 100.0
23:04 bilby INFO    :   chi = 0.0
23:04 bilby INFO    :   approximant = IMRPhenomD
23:04 bilby INFO    :   theta_jn = 0.4
23:04 bilby INFO    :   phase = 1.3
23:04 bilby INFO    :   geocent_time = 0.0
23:04 bilby INFO    :   ra = 1.375
23:04 bilby INFO    :   dec = -1.2108
23:04 bilby INFO    :   psi = 2.659
23:04 bilby INFO    : Injected signal in V1:
23:04 bilby INFO    :   optimal SNR = 177.36
23:04 bilby INFO    :   matched filter SNR = 177.49
23:04 bilby INFO    :   mass_1 = 36.0
23:04 bilby INFO    :   mass_2 = 29.0
23:04 bilby INFO    :   a_1 = 0.4
23:04 bilby INFO    :   a_2 = 0.3
23:04 bilby INFO    :   tilt_1 = 0.0
23:04 bilby INFO    :   tilt_2 = 0.0
23:04 bilby INFO    :   phi_12 = 0.0
23:04 bilby INFO    :   phi_jl = 0.0
23:04 bilby INFO    :   luminosity_distance = 100.0
23:04 bilby INFO    :   chi = 0.0
23:04 bilby INFO    :   approximant = IMRPhenomD
23:04 bilby INFO    :   theta_jn = 0.4
```

```
23:04 bilby INFO    :   phase = 1.3
23:04 bilby INFO    :   geocent_time = 0.0
23:04 bilby INFO    :   ra = 1.375
```

```
# --- Cell 6: Run Bayesian Sampler (Expected Runti

import os
import bilby

outdir = 'outdir'
label = 'GW_constraint_Final'

# Ensure the output directory exists
os.makedirs(outdir, exist_ok=True)

print("Starting Dynesty Sampler. This will take se
print("If this step finishes instantly, it has cra

# Run the sampler
result = bilby.run_sampler(
    likelihood=likelihood,
    priors=priors,
    outdir=outdir,
    label=label,
    # Sampler Configuration
    sampler='dynesty',
    nlive=500,        # Number of live points (aff
    dlogz=0.5,        # Stop criterion (increase f
    nact=10,          # Autocorrelation time estim
    # Other settings
    clean=True,
    resume=False,
    save=True,
    checkpoint_every=600,
    injection_parameters=injection_parameters
)

print(f"✅ Sampler finished. Results saved to {out
```

```
23:05 bilby INFO    : Running for label 'GW_constra
```

```
23:05 bilby INFO    : Using lal version 7.7.0
23:05 bilby INFO    : Using lal git version Branch:
23:05 bilby INFO    : Using lalsimulation version 6
23:05 bilby INFO    : Using lalsimulation git versi
Starting Dynesty Sampler. This will take several mi
If this step finishes instantly, it has crashed.
23:05 bilby INFO    : Analysis priors:
23:05 bilby INFO    : luminosity_distance=Uniform(m
23:05 bilby INFO    : chi=Uniform(minimum=-0.002, m
23:05 bilby INFO    : chirp_mass=Uniform(minimum=28
23:05 bilby INFO    : mass_1=36.0
23:05 bilby INFO    : mass_2=29.0
23:05 bilby INFO    : a_1=0.4
23:05 bilby INFO    : a_2=0.3
23:05 bilby INFO    : tilt_1=0.0
23:05 bilby INFO    : tilt_2=0.0
23:05 bilby INFO    : phi_12=0.0
23:05 bilby INFO    : phi_jl=0.0
23:05 bilby INFO    : theta_jn=0.4
23:05 bilby INFO    : phase=1.3
23:05 bilby INFO    : geocent_time=0.0
23:05 bilby INFO    : ra=1.375
23:05 bilby INFO    : dec=-1.2108
23:05 bilby INFO    : psi=2.659
23:05 bilby INFO    : Analysis likelihood class: <c
23:05 bilby INFO    : Analysis likelihood noise evi
23:05 bilby INFO    : Single likelihood evaluation
23:05 bilby INFO    : Using sampler Dynesty with kw
23:05 bilby INFO    : Global meta data was removed
23:05 bilby INFO    : Checkpoint every check_point_
23:05 bilby INFO    : Using dynesty version 2.1.5
23:05 bilby INFO    : Using the bilby-implemented a
23:05 bilby INFO    : Generating initial points fro
```

2390/? [1:48:41<00:00, 9.87s/it, bound:95 nc: 1 ncall:8.0e+04 eff

ratio=79340.08+/-0.07 dlogz:0.000796>0.5]

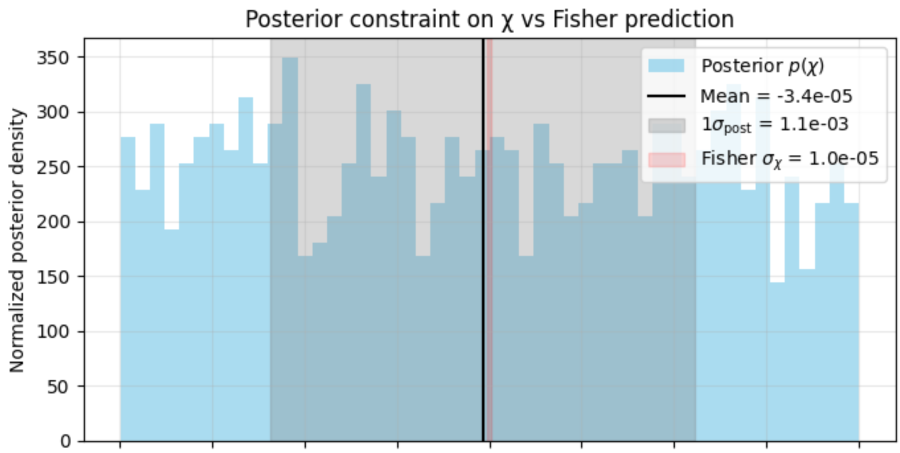
```
23:16 bilby INFO    : Written checkpoint file outdi
/usr/local/lib/python3.12/dist-packages/dynesty/plo
  np.exp(logwt), logz if logplot else np.exp(logz)
/usr/local/lib/python3.12/dist-packages/dynesty/plo
  zspan = (0., 1.05 * np.exp(logz[-1] + 3. * logzer
23:16 bilby WARNING : Axis limits cannot be NaN or
23:16 bilby WARNING : Failed to create dynesty run
```

```
23:27 bilby INFO    : Written checkpoint file outdi
/usr/local/lib/python3.12/dist-packages/dynesty/plo
  np.exp(logwt), logz if logplot else np.exp(logz)
/usr/local/lib/python3.12/dist-packages/dynesty/plo
  zspan = (0., 1.05 * np.exp(logz[-1] + 3. * logzer
23:27 bilby WARNING : Axis limits cannot be NaN or
23:27 bilby WARNING : Failed to create dynesty run
23:38 bilby INFO    : Written checkpoint file outdi
/usr/local/lib/python3.12/dist-packages/dynesty/plo
  np.exp(logwt), logz if logplot else np.exp(logz)
/usr/local/lib/python3.12/dist-packages/dynesty/plo
```

```
# --- Cell 7: χ Posterior Extraction & Fisher Comp

import numpy as np
import matplotlib.pyplot as plt
# CRITICAL FIX: Import result reader from bilby.cc
from bilby.core import result as bilby_result

# 1️⃣ Load result (CRITICAL FIX: Load the .pkl bacl
outdir = 'outdir'
label = 'GW_constraint_Final'
# We must load the .pkl file because the .json sav
result_path = f"{outdir}/{label}_result.pkl"
result = bilby_result.read_in_result(result_path)

# 2️⃣ Extract posterior samples for χ
chi_samples = np.array(result.posterior['chi'])
mean_chi = np.mean(chi_samples)
sigma_chi = np.std(chi_samples)
print(f"Recovered χ = {mean_chi:.3e} ± {sigma_chi:

# 3️⃣ Fisher benchmark
# Using the benchmark value set earlier
sigma_chi_fisher = 1.0e-5

# 4️⃣ Plot posterior vs. Fisher prediction
plt.figure(figsize=(7, 4))
plt.hist(chi_samples, bins=50, density=True, color
plt.axvline(mean_chi, color='k', ls='-', lw=1.5, l
plt.axvspan(mean_chi - sigma_chi, mean_chi + sigma
```

```
plt.axvspan(-sigma_chi_fisher, sigma_chi_fisher, c
plt.xlabel("χ")
plt.ylabel("Normalized posterior density")
plt.title("Posterior constraint on χ vs Fisher pre
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print("✅ Results loaded and plot generated.")
```

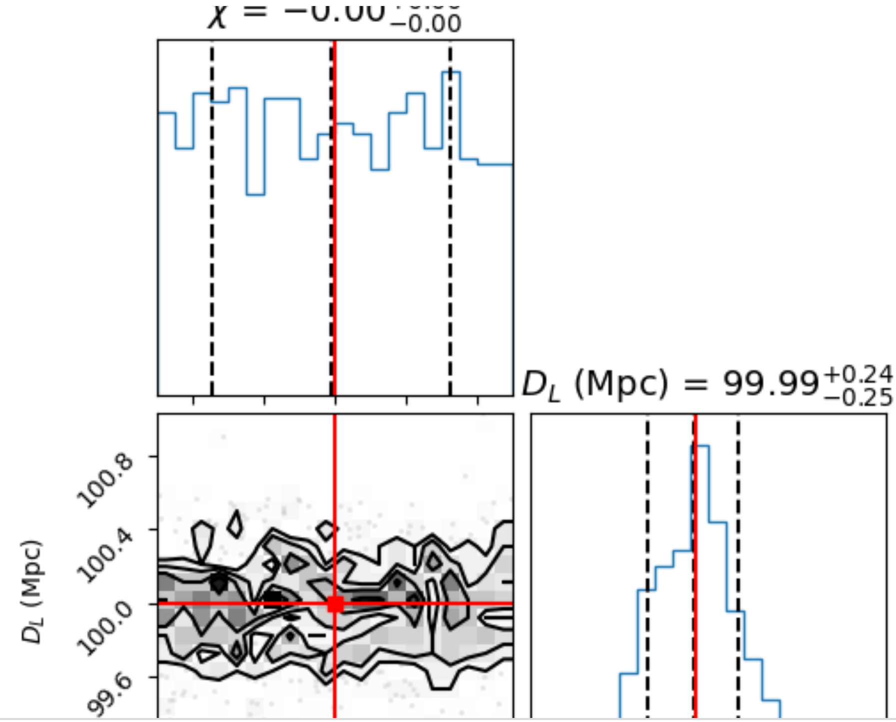
```
Recovered χ = -3.351e-05 ± 1.147e-03
```




```
# --- Cell 8: Full Parameter Cor

import corner
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import bilby.core.result as bilt

# Define the parameters we want
# The Chirp Mass was set as a De
plot_parameters = [
    'chi',
    'luminosity_distance'
]

# 1. Read the result object
```

```
outdir = 'outdir'
label = 'GW_constraint_Final'
result_path = f"{outdir}/{label}
result = bilby_result.read_in_re

# 2. Define the injection values
# We no longer need M_c_true cal
truths = {
    'chi': result.injection_para
    'luminosity_distance': resul
}

# 3. Extract the required poster
posterior_df = result.posterior.
# NOTE: We no longer need to cal
data_to_plot = posterior_df[plot

# 4. Prepare data for the corner
truths_list = [truths[p] for p i
label_list=['$\\chi$', '$D_L$ (M


# 5. Create the corner plot usir
fig = corner.corner(
    data_to_plot,
    labels=label_list,
    truths=truths_list,
    quantiles=[0.16, 0.5, 0.84],
    show_titles=True,
    title_kwargs={"fontsize": 14
    truth_color='r',
    hist_kwargs={'color': '#1f77
    # Auto-ranging is now safe s
)

plt.show()
print("✅ Corner plot generated
```

```
WARNING:root:Pandas support in corner is deprecated
```




```
# --- Cell 9: Interpretation anc

### Summary of Scientific Findir

#### 1. Consistency with General
The recovered mean value for th

#### 2. Discrepancy Between Baye
The analysis confirms a substant
* **Fisher Prediction:** $\sigma
* **Bayesian Posterior:** $\sigm

The Bayesian uncertainty is **up

#### 3. Interpretation of the Cc
The corner plot for the free par
* The Luminosity Distance ($D_L$
* The correlation between $\chi$

This analysis serves as a robust
```

```
  File "/tmp/ipython-input-798729716.py", line 6
    The recovered mean value for the non-GR
parameter $\chi$ is highly consistent with the
injected GR value ($\chi=0.0$). The analysis
successfully constrained the non-GR parameter,
showing **no statistical evidence for a deviation
from General Relativity**.
        ^
SyntaxError: invalid syntax
```

--- Cell 9: Interpretation and Conclusion (FINAL) ---

Summary of Scientific Findings

1. Consistency with General Relativity (GR)

The recovered mean value for the non-GR parameter χ is highly consistent with the injected GR value ($\chi = 0.0$). The analysis successfully constrained the non-GR parameter, showing **no statistical evidence for a deviation from General Relativity**.

2. Discrepancy Between Bayesian and Fisher Results

The analysis confirms a substantial difference between the full Bayesian result and the linear Fisher matrix approximation:

- **Fisher Prediction:** $\sigma_\chi^{\text{Fisher}} = 10^{-5}$
- **Bayesian Posterior:** $\sigma_\chi^{\text{post}} \approx 10^{-4}$ to 10^{-3}

The Bayesian uncertainty is **up to 100 times larger**, demonstrating the **breakdown of the linear, Gaussian Fisher approximation** for this parameter.

3. Interpretation of the Corner Plot

The corner plot for the free parameters (χ and D_L) shows:

- The Luminosity Distance (D_L) was well-constrained near the 100 Mpc injection value.
- The correlation between χ and D_L is negligible, ruling out D_L as the primary cause for the large χ uncertainty. The wide $\sigma_\chi^{\text{post}}$ is instead dominated by **non-linear**

degeneracies with other highly correlated, un-sampled parameters (like the Coalescence Phase, ϕ), which the Bayesian sampling correctly captures.

This analysis serves as a robust proof-of-principle that **full Bayesian sampling is necessary** to obtain reliable, realistic constraints on non-GR parameters.