DINGO Tutorial

Usage and code structure

M. Dax, J. Gair, N. Gupte, J. Wildberger, M. Pürrer, A. Kofler, V. Berenz

C. Fabbri, A. Girish, S. Clyne, A. Roussopoulos, J. Macke, A. Buonanno, B. Schölkopf

Stephen Green — 4 June 2025

Introduction

References

- **Documentation**
    - https://dingo-gw.readthedocs.io/
- **Community Discord server**
- **Code**
    - https://github.com/dingo-gw/dingo
    - Available as `dingo-gw` package from `pip` and `conda`.

- **Leverages Bilby components**
    - Priors, detector information, bilby_pipe automation
- **Key papers**
    1. https://arxiv.org/abs/2002.07656 : 5D toy model with normalizing flows
    2. https://arxiv.org/abs/2106.12594 : Amortized inference with noise conditioning
    3. https://arxiv.org/abs/2210.05686 : Importance sampling
    4. https://arxiv.org/abs/2305.17161 : Flow matching
    5. https://arxiv.org/abs/2407.09602 : Binary neutron stars

Introduction

Key steps

1. **Generate training datasets**
    a. Intrinsic waveforms (polarizations), specified by masses, spins, inclination, and phase.
    b. Noise power spectral densities, with a distribution often corresponding to an observing run.
2. **Train the network**
    a. Construct network based on specified configuration.
    b. During training, sample extrinsic parameters (sky position, distance, time of coalescence) and noise realizations, and construct simulated data.
3. **Perform inference**
    a. For (observed or simulated) data, draw posterior samples from the trained network.
    b. Optionally importance sample to true posterior. Gives Bayesian evidence.

Specified by YAML config files

Use Dingo API or INI file

Generating training data

Waveform Dataset

- YAML file components:
    - Domain (f_min, f_max, etc.)
    - Waveform generator (approximant, etc.)
    - Intrinsic prior (Bilby format)
    - Size of dataset
    - Compression
- `dingo_generate_dataset` —> HDF5 file.

```yaml
domain:
  type: FrequencyDomain
  f_min: 20.0
  f_max: 1024.0
  delta_f: 0.125  # Expressions like 1.0/8.0 would require eval and are not supported

waveform_generator:
  approximant: IMRPhenomXPHM  # SEOBNRv4PHM
  f_ref: 20.0
  # f_start: 15.0  # Optional setting useful for EOB waveforms. Overrides f_min when ge
  spin_conversion_phase: 0.0

# Dataset only samples over intrinsic parameters. Extrinsic parameters are chosen at tr
intrinsic_prior:
  mass_1: bilby.core.prior.Constraint(minimum=10.0, maximum=120.0)
  mass_2: bilby.core.prior.Constraint(minimum=10.0, maximum=120.0)
  chirp_mass: bilby.gw.prior.UniformInComponentsChirpMass(minimum=15.0, maximum=150.0)
  mass_ratio: bilby.gw.prior.UniformInComponentsMassRatio(minimum=0.125, maximum=1.0)
  phase: default
  a_1: bilby.core.prior.Uniform(minimum=0.0, maximum=0.99)
  a_2: bilby.core.prior.Uniform(minimum=0.0, maximum=0.99)
  tilt_1: default
  tilt_2: default
  phi_12: default
  phi_jl: default
  theta_jn: default
  # Reference values for fixed (extrinsic) parameters. These are needed to generate a w
  luminosity_distance: 100.0  # Mpc
  geocent_time: 0.0  # s

# Dataset size
num_samples: 5000000

# Save a compressed representation of the dataset
compression:
  svd:
    # Truncate the SVD basis at this size. No truncation if zero.
    size: 200
    num_training_samples: 50000
    num_validation_samples: 10000
  whitening: aLIGO_ZERO_DET_high_P_asd.txt
```

Waveform dataset

Interactive usage

```python
from dingo.gw.dataset import WaveformDataset
```

```python
wfd = WaveformDataset("waveform_dataset.hdf5")
```

```
Loading dataset from waveform_dataset.hdf5.
```

```python
wfd.settings
```

Contains contents of YAML config

```
{'domain': {'type': 'FrequencyDomain',
  'f_min': 20.0,
  'f_max': 512.0,
  'delta_f': 0.25},
 'waveform_generator': {'approximant': 'IMRPhenomXPHM',
  'f_ref': 20.0,
  'spin_conversion_phase': 0.0},
 'intrinsic_prior': {'mass_1': 'bilby.core.prior.Constraint(minimum=10.0, maximum=120.0)',
  'mass_2': 'bilby.core.prior.Constraint(minimum=10.0, maximum=120.0)',
  'chirp_mass': 'bilby.gw.prior.UniformInComponentsChirpMass(minimum=15.0, maximum=150.0)',
  'mass_ratio': 'bilby.gw.prior.UniformInComponentsMassRatio(minimum=0.125, maximum=1.0)',
  'phase': 'default',
  'a_1': 'bilby.core.prior.Uniform(minimum=0.0, maximum=0.99)',
  'a_2': 'bilby.core.prior.Uniform(minimum=0.0, maximum=0.99)',
  'tilt_1': 'default',
  'tilt_2': 'default',
  'phi_12': 'default',
  'phi_jl': 'default',
  'theta_jn': 'default',
  'luminosity_distance': 100.0,
  'geocent_time': 0.0},
 'num_samples': 100000,
 'compression': {'svd': {'size': 100,
   'num_training_samples': 5000,
   'num_validation_samples': 1000},
  'whitening': 'aLIGO_ZERO_DET_high_P_asd.txt'}}
```

```python
wfd.domain
```

Domain object

```
<dingo.gw.domains.FrequencyDomain at 0x107705d50>
```

```python
wfd.domain.sample_frequencies
```

```
array([0.0000e+00, 2.5000e-01, 5.0000e-01, ..., 5.1150e+02, 5.1175e+02,
       5.1200e+02], dtype=float32)
```

```python
wfd.polarizations.keys()
```

Compressed (SVD) polarizations

```
dict_keys(['h_cross', 'h_plus'])
```

```python
wfd.polarizations["h_cross"].shape
```

```
(100000, 100)
```

```python
wfd[0]
```

Indexing gives dict with parameters and (decompressed) waveform

```
{'parameters': {'chirp_mass': 64.41381314230478,
  'mass_ratio': 0.47177577838693513,
  'phase': 5.763232159969959,
  'a_1': 0.03207003325055646,
  'a_2': 0.3098953412457675,
  'tilt_1': 1.3834749678369773,
  'tilt_2': 2.165452272815846,
  'phi_12': 4.266047153805945,
  'phi_jl': 5.378021355201301,
  'theta_jn': 2.2850745797311984,
  'luminosity_distance': 100.0,
  'geocent_time': 0.0},
 'waveform': {'h_cross': array([ 0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
          0.00000000e+00+0.00000000e+00j, ...,
          1.03684256e-25-7.56307764e-26j, -2.04143358e-26-1.33733413e-25j,
          0.00000000e+00+0.00000000e+00j]),
  'h_plus': array([ 0.00000000e+00+0.00000000e+00j,  0.00000000e+00+0.00000000e+00j,
          0.00000000e+00+0.00000000e+00j, ...,
         -7.59414876e-26-1.04618327e-25j, -1.30488144e-25+3.92787044e-26j,
          0.00000000e+00+0.00000000e+00j])}}
```

Generating training data

ASD Dataset

- Executed with `dingo_generate_asd_dataset`
- Downloads LVK data from GWOSC and generates PSDs using Welch method.
- Key parameters:
  - Observing run
  - Set of detectors
  - Time between PSDs / max number of PSDs

```
dataset_settings:
#  f_min: 0          # defaults to 0
#  f_max: 2048       # defaults to f_s/2
  f_s: 4096
  time_psd: 1024
  T: 8.0
  window:
    roll_off: 0.4
    type: tukey
  time_gap: 0           # specifies the time skipped between to consecutive
  num_psds_max: 0   # if set > 0, only a subset of all available PSDs will
#  channels:
#    H1: H1:DCS-CALIB_STRAIN_C02
#    L1: L1:DCS-CALIB_STRAIN_C02
  detectors:
    - H1
    - L1
  observing_run: O1
# condor:
#   env_path: /path/to/environment
#   num_jobs: 2 # per detector
#   num_cpus: 16
#   memory_cpus: 16000
```

Training

Overview

- Training carried out using `dingo_train` (or `dingo_train_condor`)
    - YAML config file specifies further data settings, neural network configuration, training plan, and additional local settings (e.g., number of cores).
    - Produces regular checkpoint files (`.pt`), as well as training history information.

```yaml
data:
  waveform_dataset_path: training_data/waveform_dataset.hdf5  # Contains intrinsic w
  train_fraction: 0.95
  window:  # Needed to calculate window factor for simulated data
    type: tukey
    f_s: 4096
    T: 8.0
    roll_off: 0.4
  domain_update:
    f_min: 20.0
    f_max: 1024.0
  svd_size_update: 200  # Optionally, reduce the SVD size when decompressing (for pe
  detectors:
    - H1
    - L1
  extrinsic_prior:  # Sampled at train time
    dec: default
    ra: default
    geocent_time: bilby.core.prior.Uniform(minimum=-0.10, maximum=0.10)
    psi: default
    luminosity_distance: bilby.core.prior.Uniform(minimum=100.0, maximum=1000.0)
  ref_time: 1126259462.391
  inference_parameters:
  - chirp_mass
  - mass_ratio
  - chi_1
  - chi_2
  - theta_jn
  - dec
  - ra
  - geocent_time
  - luminosity_distance
  - psi

# Model architecture
model:
  posterior_model_type: normalizing_flow
  # kwargs for neural spline flow
  posterior_kwargs:
    num_flow_steps: 30
    base_transform_kwargs:
      hidden_dim: 1024
      num_transform_blocks: 5
      activation: elu
```

Training

Network configuration

- Specifies architecture and size of the NN
- Currently supported: spline flows, flow matching, score matching
- First embedding net layer is initialized with SVD (computed automatically).
- Extension to new architectures is straightforward (requires modification of Dingo code)

```yaml
model:
  posterior_model_type: normalizing_flow
  # kwargs for neural spline flow
  posterior_kwargs:
    num_flow_steps: 30
    base_transform_kwargs:
      hidden_dim: 1024
      num_transform_blocks: 5
      activation: elu
      dropout_probability: 0.0
      batch_norm: True
      num_bins: 8
      base_transform_type: rq-coupling
  # kwargs for embedding net
  embedding_kwargs:
    output_dim: 128
    hidden_dims: [1024, 1024, 1024, 1024, 1024, 1024,
                  512, 512, 512, 512, 512, 512,
                  256, 256, 256, 256, 256, 256,
                  128, 128, 128, 128, 128, 128]
    activation: elu
    dropout: 0.0
    batch_norm: True
    svd:
      num_training_samples: 50000
      num_validation_samples: 10000
      size: 200
```

Training

Training plan

- Specifies training configuration (epochs, batch size, optimizer, learning rate schedule)
- Training stages
  - **Stage 0**: use fixed ASD per detector.
  - **Stage 1**: use multiple ASDs per detector.
    → ASDs could cover an entire observing run
    → Learning rate should be reduced (~0.1x)
- Training stages may also be used without changing the ASD dataset

```yaml
training:

  stage_0:  # pretraining stage: train with fixed ASDS
    epochs: 300
    asd_dataset_path: path/to/fiducial_asds.hdf5  # contains a single ASD per detector
    freeze_rb_layer: True  # freeze the weights of the first layer
    optimizer:
      type: adam
      lr: 0.0003
    scheduler:
      type: cosine
      T_max: 300
    batch_size: 4096

  stage_1:
    epochs: 150
    asd_dataset_path: /path/to/asds.hdf5 # contains many different ASDs per detector
    for finetuning
    freeze_rb_layer: False
    optimizer:
      type: adam
      lr: 0.00003  # learning
    scheduler:
      type: cosine
      T_max: 150
    batch_size: 4096
```

Inference

Overview

- We will supply trained networks for common configurations (e.g., Zenodo).
- Two options for inference:
    - **Sampler API** — useful for injections, experiments
    - **dingo_pipe** — based on bilby_pipe
- After inference, typically importance sample to true posterior.

Inference

Sampler API

```python
from dingo.core.posterior_models import NormalizingFlowPosteriorModel
from dingo.gw.inference.gw_samplers import GWSampler
from dingo.gw.injection import Injection
from dingo.gw.noise.asd_dataset import ASDDataset

model_path = "/Users/stephen/Documents/Research/dingo-gw/networks/npe_xphm_03/model_225.pt"
asd_path = "/Users/stephen/Documents/Research/dingo-gw/asd-datasets/asds_03.hdf5"

pm = NormalizingFlowPosteriorModel(model_filename=model_path, device="cpu")
sampler = GWSampler(model=pm)
injection = Injection.from_posterior_model_metadata(pm.metadata)
injection.asd = ASDDataset(asd_path, ifos=["H1", "L1"])

inj = injection.random_injection()
inj
```

```
Putting posterior model to device cpu.
Setting spin_conversion_phase = 0.0. Using this value for the phase parameter for conversion to cartesian spins.
Loading dataset from /Users/stephen/Documents/Research/dingo-gw/asd-datasets/asds_03.hdf5.
{'parameters': {'chirp_mass': 57.278074855581515,
  'mass_ratio': 0.6151457796358613,
  'phase': 1.0997257154960403,
  'a_1': 0.32634736401417636,
  'a_2': 0.3557619934382651,
  'tilt_1': 1.1511341101132335,
  'tilt_2': 0.9748580474950054,
  'phi_12': 5.4260642925173945,
  'phi_jl': 4.43955174105639,
  'theta_jn': 1.181696796407354,
  'geocent_time': 0.06106530755855896,
  'luminosity_distance': 331.4908286977259,
  'ra': 2.4748760116443944,
  'dec': 0.08157802486421703,
  'psi': 2.772987885824801,
  'H1_time': 0.06723323238328562,
  'L1_time': 0.06077097218940409 6},
 'extrinsic_parameters': {},
 'waveform': {'H1': array([ 0.00000000e+00+0.00000000e+00j,
          0.00000000e+00+0.00000000e+00j,
          0.00000000e+00+0.00000000e+00j, ...,
          2.07851918e-23-1.69062288e-23j,
         -2.32579040e-23+1.33809684e-23j,
          1.09820335e-23-5.69711902e-24j]),
  'L1': array([ 0.00000000e+00+0.00000000e+00j,
          0.00000000e+00+0.00000000e+00j,
          0.00000000e+00+0.00000000e+00j, ...,
         -3.78112517e-24+1.91246989e-23j,
         -2.76484093e-25-8.95628488e-25j,
          4.98571141e-23+2.88698180e-23j])},
 'asds': {'H1': array([1.00000000e-20, 1.00000000e-20, 1.00000000e-20, ...,
         1.11230210e-23, 1.02800786e-23, 1.02797299e-23]),
  'L1': array([1.00000000e-20, 1.00000000e-20, 1.00000000e-20, ...,
         8.22632321e-24, 9.35929613e-24, 1.61833873e-23])}}
```

Injection as dict

1. Load model using PosteriorModel class
2. Place model in GWSampler
3. Prepare injection and place in sampler
4. Execute `GWSampler.run_sampler()`

```python
sampler.context = inj
sampler.run_sampler(100, batch_size=50)
print(sampler.samples)
```

```
Running sampler to generate 100 samples.
Done. This took 0.7 s.
    chirp_mass  mass_ratio       a_1  ...       dec       psi  log_prob
0    76.068756    0.641365  0.291059  ... -0.065704  1.120505  2.366869
1    74.916054    0.703768  0.103938  ...  0.054271  2.879373  5.380791
2    72.652740    0.591754  0.324786  ... -0.015465  2.522037  4.834624
3    74.191132    0.529014  0.312044  ...  0.022786  0.414739 -1.557718
4    71.489929    0.598537  0.016802  ... -0.045194  2.887955  2.665492
..         ...         ...       ...  ...       ...       ...       ...
95   68.453186    0.488925  0.087389  ... -0.010306  1.778464  3.435484
96   67.645164    0.461319  0.022716  ... -0.020204  1.926357  4.476946
97   71.413193    0.651810  0.075135  ... -0.062583  1.685776  2.043992
98   78.013023    0.776698  0.474594  ... -0.020065  1.708209  1.223689
99   71.692337    0.530850  0.070890  ...  0.024096  3.099460  2.456547

[100 rows x 15 columns]
```

Inference

Result class

- Contains functionality for post-processing Dingo samples:
    - Saving / loading
    - Plotting
    - Importance sampling / calculating evidence
    - Generating synthetic phase
    - Training new flows based on samples
- Stores all precursor settings (e.g., network settings, training, and event data).

Inference

`dingo_pipe`

Most settings (detectors, etc.) are taken from the Dingo model.

- Automates inference tasks, including downloading of GW data. Specified by INI file:
  - **Job submission**
  - **Sampler:** specify DINGO models, number of samples, GNPE iteration, data/prior updates for importance sampling, …
  - **Data generation:** specify GPS time, point to frame files (or gwosc), point to PSDs
  - (Optional) Add **calibration marginalization** (only works with importance sampling)
  - (Optional) Make **corner plot and diagnostic plots** for importance sampling
- Saves a Result HDF5 file in the end, which can also be read by PESummary.

```
################################################################################
##  Job submission arguments
################################################################################

local = true
accounting = dingo
request-cpus-importance-sampling = 64
n-parallel = 10
sampling-requirements = [TARGET.CUDAGlobalMemoryMb>40000]
extra-lines = [+WantsGPUNode = True]
simple-submission = false

################################################################################
##  Sampler arguments
################################################################################

model-init = /path/to/gnpe-init-model.pt
model = /path/to/gnpe-model.pt
device = cuda
num-gnpe-iterations = 30
num-samples = 100000
batch-size = 50000
recover-log-prob = true  # density recovery required for GNPE model
importance-sample = true
prior-dict = {}  # cptionally update prior settings in inportance sampling

################################################################################
## Data generation arguments
################################################################################

label = GW150914
trigger-time = 1126259462.391
data-dict = {H1: /path/to/H1_frame.gwf, L1: /path/to/L1_frame.gwf}
channel-dict = {H1: DCS-CALIB_STRAIN_C02,L1: DCS-CALIB_STRAIN_C02}
psd-dict = {H1: H1_psd.dat, L1: L1_psd.dat}
outdir = /path/to/outdir

################################################################################
## Calibration marginalization arguments
################################################################################

calibration-model = CubicSpline
spline-calibration-envelope-dict = {H1: H1_calib_env.txt, L1: L1_calib_env.txt}
spline-calibration-nodes = 10
spline-calibration-curves = 1000

################################################################################
## Plotting arguments
################################################################################

plot-corner = true
plot-weights = true
plot-log-probs = true
```

New feature

Multi-banding

```python
from dingo.gw.domains import FrequencyDomain, MultibandedFrequencyDomain
from dingo.gw.waveform_generator import WaveformGenerator
from dingo.gw.prior import build_prior_with_defaults
```

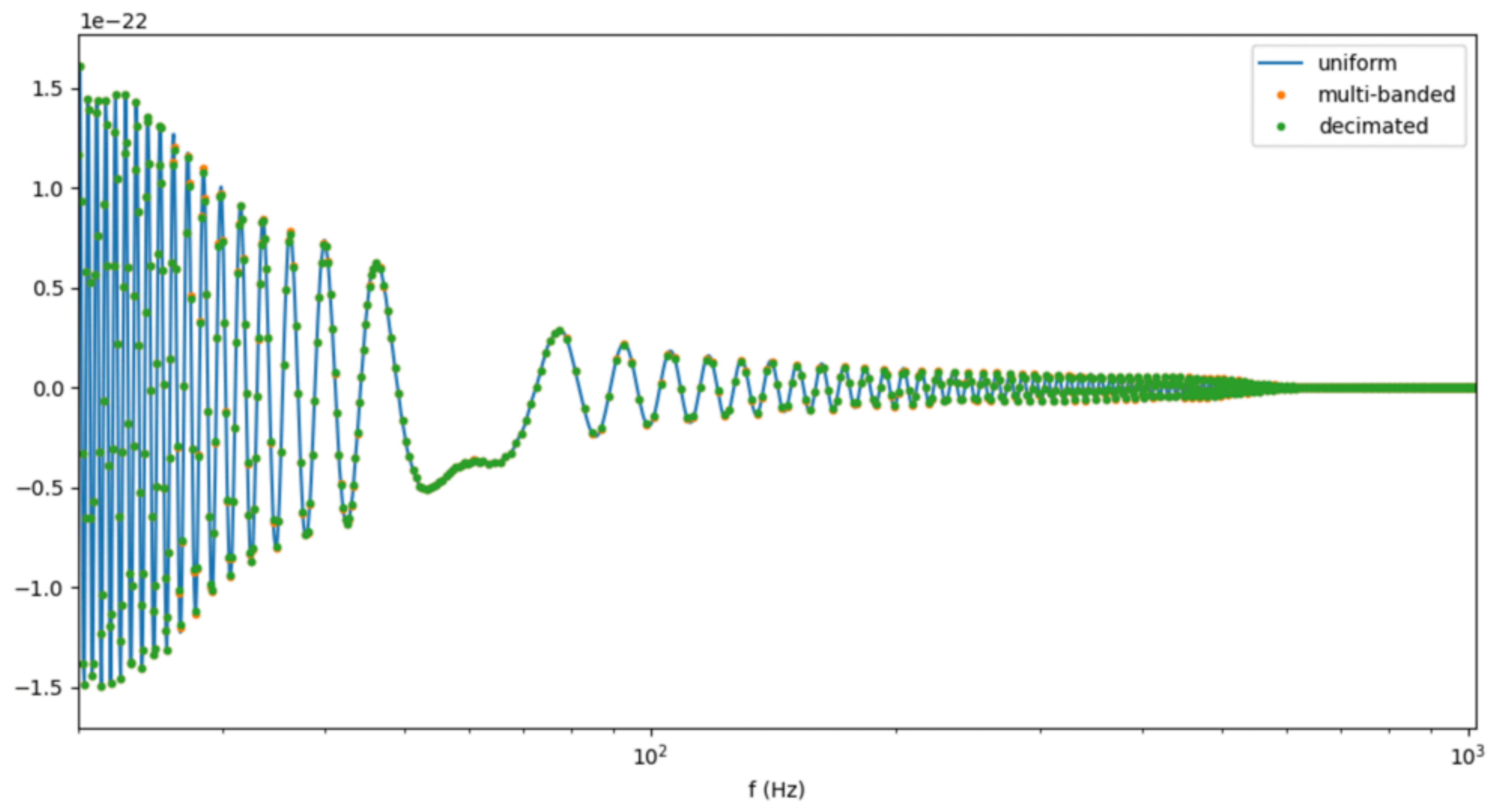
```python
ufd = FrequencyDomain(f_min, nodes[-1], delta_f_initial)
mfd = MultibandedFrequencyDomain(nodes, delta_f_initial, ufd)
```

```python
mfd.domain_dict
```

```
{'type': 'MultibandedFrequencyDomain',
 'nodes': [20.0, 26.0, 34.0, 46.0, 62.0, 78.0, 1038.0],
 'delta_f_initial': 0.0625,
 'base_domain': {'type': 'FrequencyDomain',
  'f_min': 20.0,
  'f_max': 1037.9375,
  'delta_f': 0.0625,
  'window_factor': None}}
```

```python
wfg_ufd = WaveformGenerator("IMRPhenomXPHM", ufd, 20.0)
wfg_mfd = WaveformGenerator("IMRPhenomXPHM", mfd, 20.0)
```

```python
plt.figure(figsize=(12,6))
plt.plot(ufd(), wf_ufd["h_plus"].real, label="uniform")
plt.plot(mfd(), wf_mfd["h_plus"].real, ".", label="multi-banded")
plt.plot(mfd(), wf_decimated["h_plus"].real, ".", label="decimated")
plt.xscale("log")
plt.xlabel('f (Hz)')
plt.legend()
plt.xlim((20, 1024))
plt.show()
```



Much more efficient waveform representation.

Interactive tutorial

- https://github.com/annalena-k/tutorial-dingo-introduction