# Machine Learning for Gravitational Waves: `jax`

Thibeau Wouters

December 15, 2023

Utrecht University

# Table of Contents

# Table of Contents

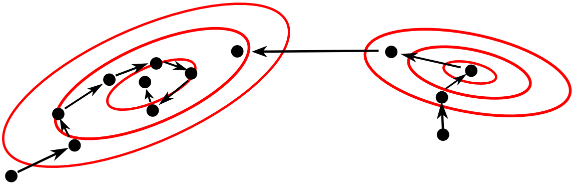
# Parameter estimation

- Parameter estimation (PE): get posterior of EM/GW parameters $\theta$

$$p(\theta|d) = \frac{p(d|\theta)p(\theta)}{p(d)}$$

- Sampling via Markov Chain Monte Carlo (MCMC) [1]

**Goal**: Improve algorithms with a `jax` implementation



# Why `jax`?

What are the benefits of `jax` for MCMC?

1. Automatic differentiation (AD)
2. Just-in-time (JIT) compilation
3. GPU acceleration
4. Parallelization
5. Shown to speed up PE for GWs [2], cosmology [3],...

# What is my experience with `jax`?

1. TaylorF2, NRTidalv2 GW in `ripple` [4]

2. Worked with `flowMC` [5, 6], a gradient-based, normalizing flow-enhanced MCMC sampler

3. Worked on `jim` [2], a PE pipeline for GWs (see appendix of this presentation for more information)

4. Implemented kilonova surrogate models in `flax` [7] for `NMMA` [8]

What are some take-aways for getting started with `jax`?

# Table of Contents

# #1 – Don't be scared, just try it out!

- `jax` is easy to get started with: `numpy` $\rightarrow$ `jax.numpy` works well!
- Read some tutorials, and make sure to read the common gotchas (demo)
  - Arrays are immutable
  - Functions must be pure for jitting
  - Random numbers are a bit annoying

# #2 – Functions must be pure: example

Example: `lalsuite` vs `jax` for NRTidalv2

```c
else if ( 0. <= lambda2bar && lambda2bar  < 1. ) {
    /* Extension of the fit in the range lambda2bar=[0,1.] so that
    the BH limit is enforced, lambda2bar->0 gives quadparam->1. and
    the junction with the universal relation is smooth, of class C2  */
    return 1. + lambda2bar*(0.427688866723244 + lambda2bar*(-0.324336526985068 + lambda2bar*0.1107439432180572));
}
else {
    lnx = log( lambda2bar );
}
REAL8 lny = XLALSimUniversalRelation( lnx, coeffs );
return exp(lny);
```

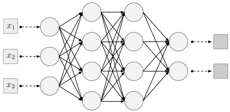
`lalsuite`

```python
def get_quadparam_octparam(lambda_: float) -> tuple[float, float]:

    # Check if lambda is low or not, and choose right subroutine
    is_low_lambda = lambda_ < 1
    return jax.lax.cond(is_low_lambda, _get_quadparam_octparam_low, _get_quadparam_octparam_high, lambda_)

def _get_quadparam_octparam_low(lambda_: float) -> tuple[float, float]:
            You, now • Uncommitted changes
    # Coefficients of universal relation
    oct_coeffs = [0.003131, 2.071, -0.7152, 0.2458, -0.03309]

    # Extension of the fit in the range lambda2 = [0,1.] so that the BH limit is enforced, lambda2bar->0 gives quadpa
    quadparam = 1. + lambda_ * (0.427688866723244 + lambda_ * (-0.324336526985068 + lambda_ * 0.1107439432180572))
    log_quadparam = jnp.log(quadparam)

    # Compute octparam:
    log_octparam = universal_relation(oct_coeffs, log_quadparam)
    octparam = jnp.exp(log_octparam)

    return quadparam, octparam

def _get_quadparam_octparam_high(lambda_: float) -> tuple[float, float]:
```

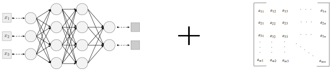
`jax`
(code)

Neural networks in `flax`:

- Most similar to `PyTorch`, more work than `tensorflow`
- `flax` has a different mindset: compare tensorflow to flax




`nn.train()`



architecture + weights/state

`state=apply_gradients(grads)`

# Table of Contents

# Conclusion

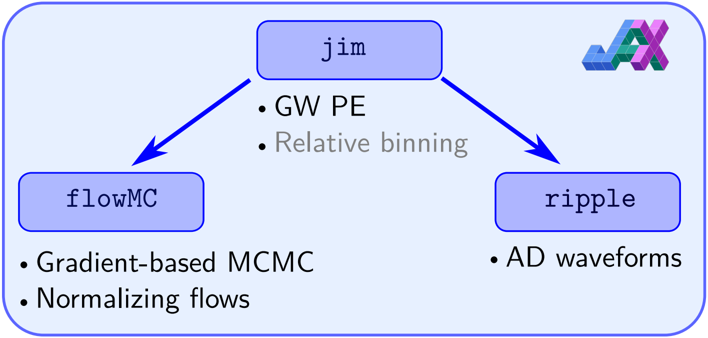
- `jax` is a great tool

# Table of Contents

# Overview

We extend `jim` [2], based on `jax` [9], with building blocks:

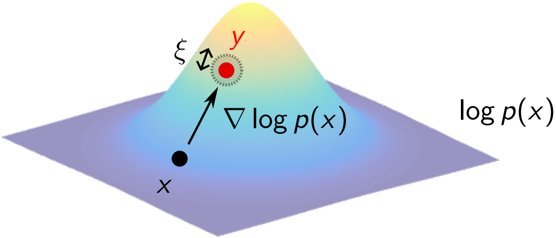
1. Normalizing flow-enhanced, gradient-based MCMC (`flowMC` [5, 6])
2. Automatically-differentiable (AD) GW (`ripple` [4])
3. Relative binning likelihood [10]



1. **Local sampling**: MALA (Metropolis-adjusted Langevin algorithm)
   - Proposal $y$: Langevin diffusion

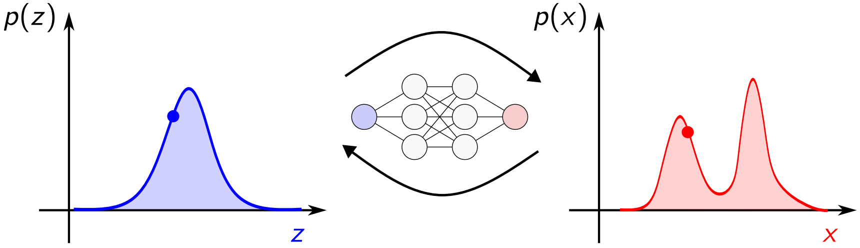
$$y = x + \frac{\epsilon^2}{2} \nabla \log p(x) + \epsilon \xi$$

   - Metropolis-Hastings acceptance step
   - Motivates the need for automatic differentiation



Normalizing flows (NF):

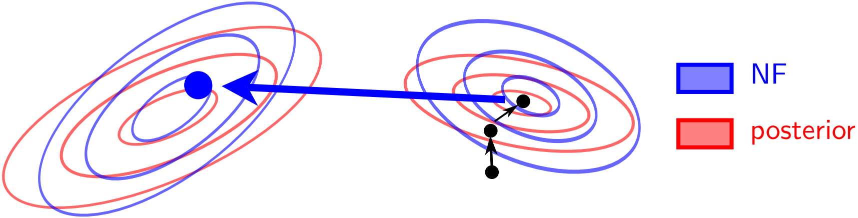
- Latent space: easy to sample (e.g. Gaussian)
- Data space: distribution learned from samples
- Enable approximate sampling from complicated distributions



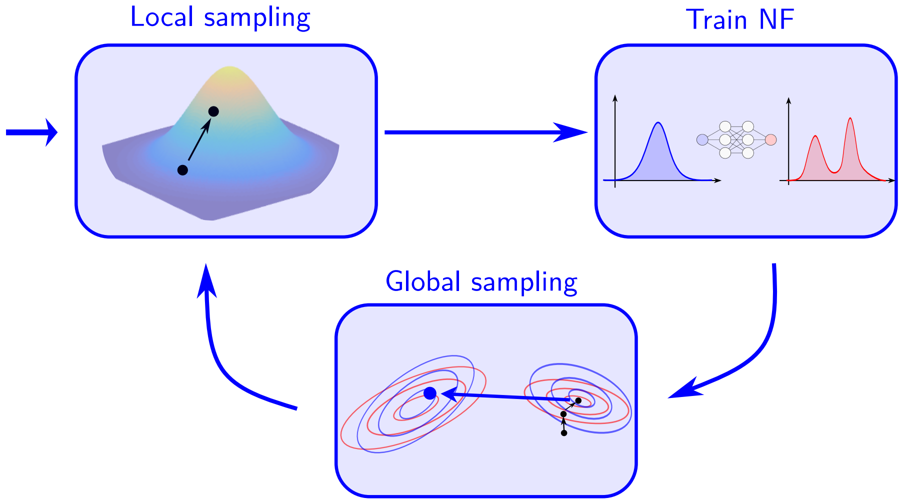
# flowMC – global sampling

❷ **Global sampling**

- Global proposal by sampling from NF
- Metropolis-Hastings acceptance step

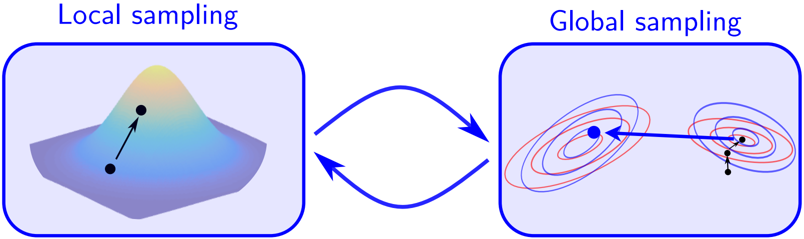


**Training loop** & Production loop



# flowMC – complete algorithm

Training loop & **Production loop**



# Results

- TaylorF2 in `ripple`
- IMRPhenomD_NRTidalv2 in `ripple` (ongoing)
- Reproduced PE for GW170817 & GW190425 with TaylorF2
- $\sim$ 30 mins training, $\sim$ 1 min sampling