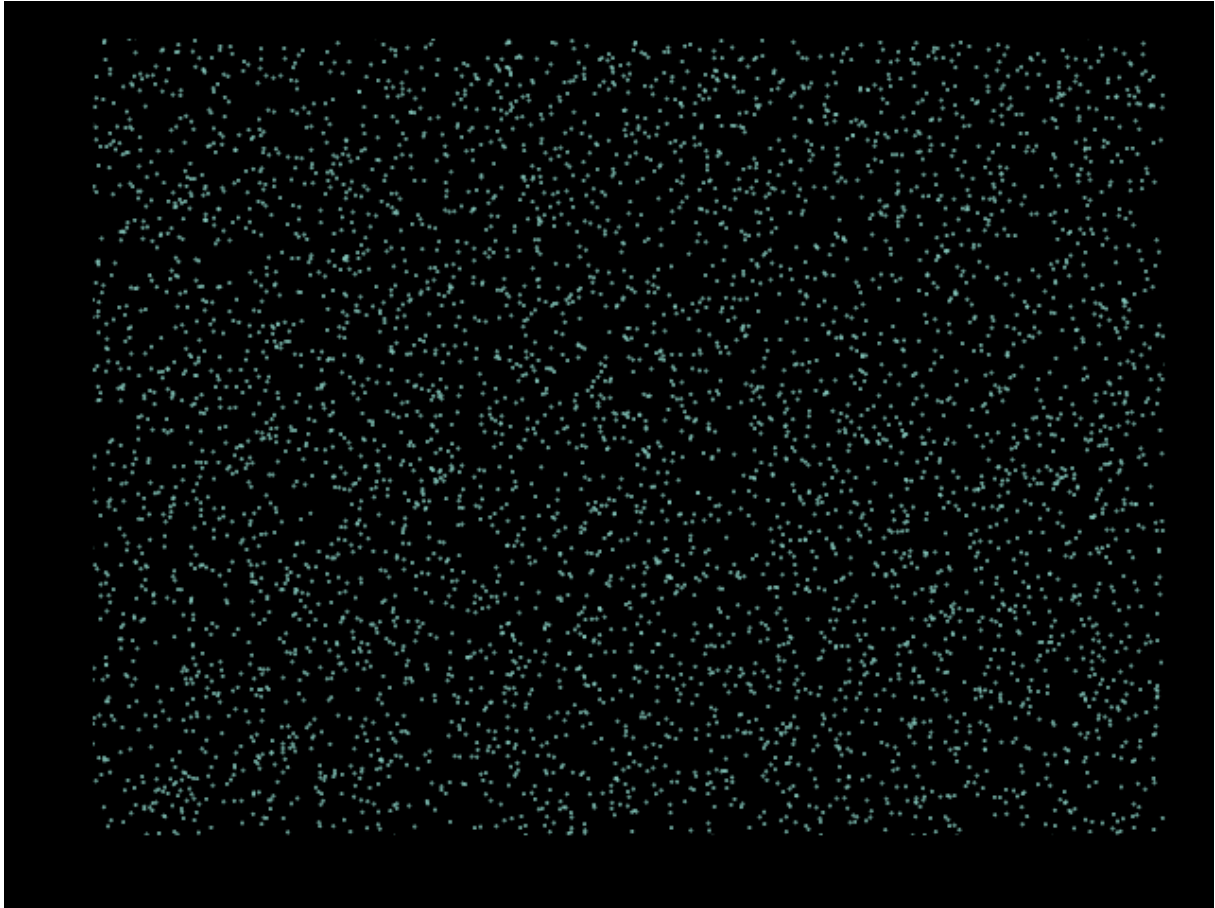

BlackJAX



What is BlackJAX?

BlackJAX is a library of samplers for JAX that works on CPU as well as GPU.

It is *not* a probabilistic programming library. However it integrates really well with PPLs as long as they can provide a (potentially unnormalized) log-probability density function compatible with JAX.

Who should use BlackJAX?

BlackJAX should appeal to those who: - Have a logpdf and just need a sampler; - Need more than a general-purpose sampler; - Want to sample on GPU; - Want to build upon robust elementary blocks for their research; - Are building a probabilistic programming language; - Want to learn how sampling algorithms work.

Quickstart

Installation

You can install BlackJAX using `pip`:

```
1 pip install blackjax
```

or via conda-forge:

```
1 conda install -c conda-forge blackjax
```

Nightly builds (bleeding edge) of Blackjax can also be installed using `pip`:

```
1 pip install blackjax-nightly
```

BlackJAX is written in pure Python but depends on XLA via JAX. By default, the version of JAX that will be installed along with BlackJAX will make your code run on CPU only. **If you want to use BlackJAX on GPU/TPU** we recommend you follow these instructions to install JAX with the relevant hardware acceleration support.

Example

Let us look at a simple self-contained example sampling with NUTS:

```
1 import jax
2 import jax.numpy as jnp
3 import jax.scipy.stats as stats
4 import numpy as np
5
6 import blackjax
7
8 observed = np.random.normal(10, 20, size=1_000)
9 def logdensity_fn(x):
10     logpdf = stats.norm.logpdf(observed, x["loc"], x["scale"])
11     return jnp.sum(logpdf)
12
13 # Build the kernel
14 step_size = 1e-3
15 inverse_mass_matrix = jnp.array([1., 1.])
16 nuts = blackjax.nuts(logdensity_fn, step_size, inverse_mass_matrix)
17
18 # Initialize the state
19 initial_position = {"loc": 1., "scale": 2.}
20 state = nuts.init(initial_position)
21
22 # Iterate
```

```
23 rng_key = jax.random.key(0)
24 for step in range(100):
25     nuts_key = jax.random.fold_in(rng_key, step)
26     state, _ = nuts.step(nuts_key, state)
```

See the documentation for more examples of how to use the library: how to write inference loops for one or several chains, how to use the Stan warmup, etc.

Philosophy

What is BlackJAX?

BlackJAX bridges the gap between “one liner” frameworks and modular, customizable libraries.

Users can import the library and interact with robust, well-tested and performant samplers with a few lines of code. These samplers are aimed at PPL developers, or people who have a logpdf and just need a sampler that works.

But the true strength of BlackJAX lies in its internals and how they can be used to experiment quickly on existing or new sampling schemes. This lower level exposes the building blocks of inference algorithms: integrators, proposal, momentum generators, etc and makes it easy to combine them to build new algorithms. It provides an opportunity to accelerate research on sampling algorithms by providing robust, performant and reusable code.

Why BlackJAX?

Sampling algorithms are too often integrated into PPLs and not decoupled from the rest of the framework, making them hard to use for people who do not need the modeling language to build their logpdf. Their implementation is most of the time monolithic and it is impossible to reuse parts of the algorithm to build custom kernels. BlackJAX solves both problems.

How does it work?

BlackJAX allows to build arbitrarily complex algorithms because it is built around a very general pattern. Everything that takes a state and returns a state is a transition kernel, and is implemented as:

```
1 new_state, info = kernel(rng_key, state)
```

kernels are stateless functions and all follow the same API; state and information related to the transition are returned separately. They can thus be easily composed and exchanged. We specialize these kernels by closure instead of passing parameters.

Contributions

Please follow our short guide.

Citing Blackjax

To cite this repository:

```
1 @misc{cabezas2024blackjax,  
2   title={BlackJAX: Composable {B}ayesian inference in {JAX}},  
3   author={Alberto Cabezas and Adrien Corenflos and Junpeng Lao and  
4     Rémi Louf},  
5   year={2024},  
6   eprint={2402.10797},  
7   archivePrefix={arXiv},  
8   primaryClass={cs.MS}  
}
```

In the above bibtex entry, names are in alphabetical order, the version number should be the last tag on the `main` branch.

Acknowledgements

Some details of the NUTS implementation were largely inspired by NumPyro's.