

# TUTORIAL I

Liesel Devs

# WELCOME

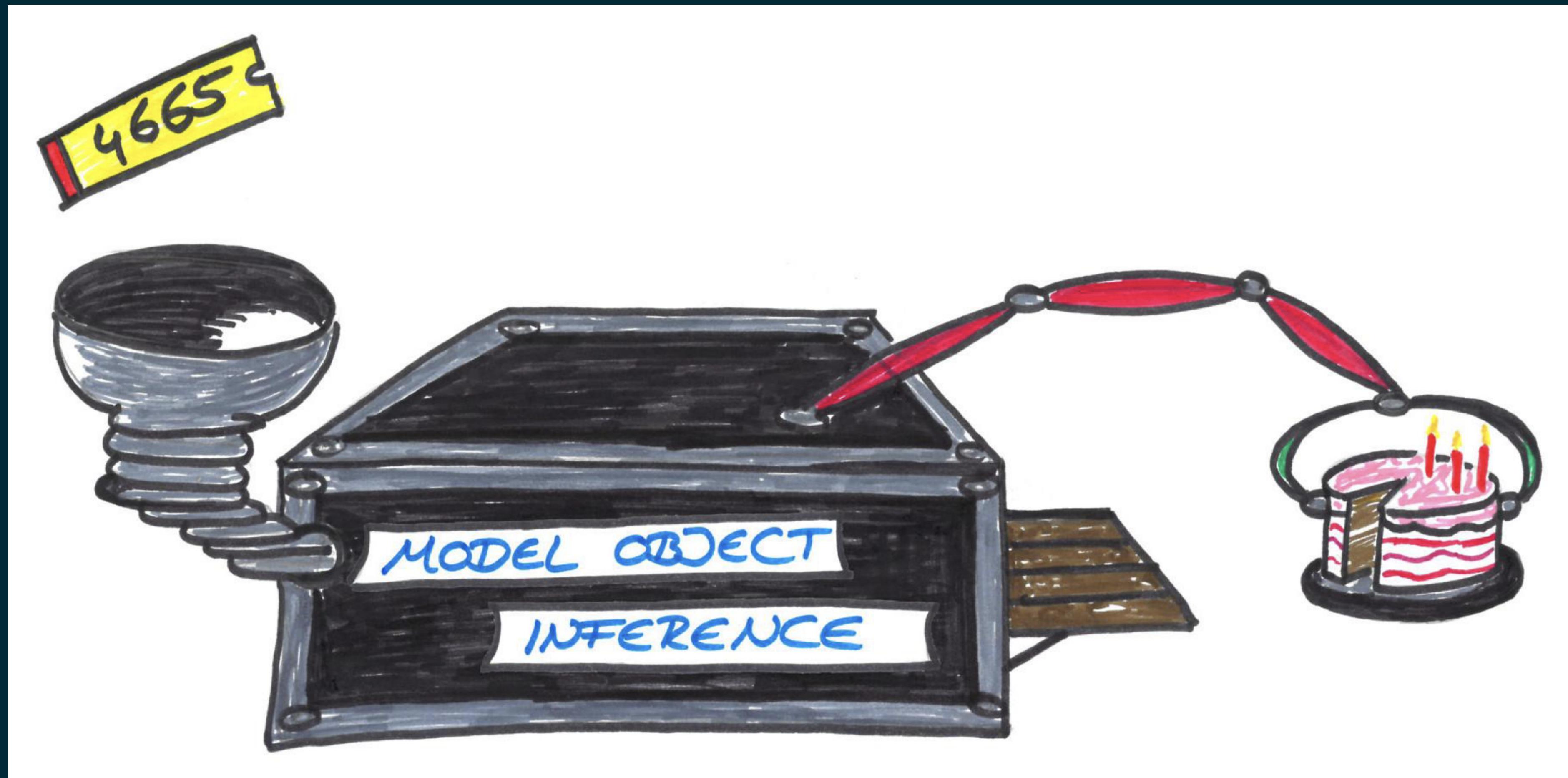
# AGENDA

- In the tutorial, you will gain some hand-on experience working on Bayesian models with Liesel
- We start simple and will make our way to Bayesian distributional regression
- Some tutorials have small lecture-like parts where we introduce some software / concepts
- In all tutorials, we will give you exercise sheets with tasks you can work on in the class
- We will actively support you

# WHAT IS LIESEL?

Wait for it.

A LOT OF STATISTICAL SOFTWARE WORKS LIKE THIS...



blackbox

...BUT LIESEL TRIES TO BE LIKE THIS



lieselbox

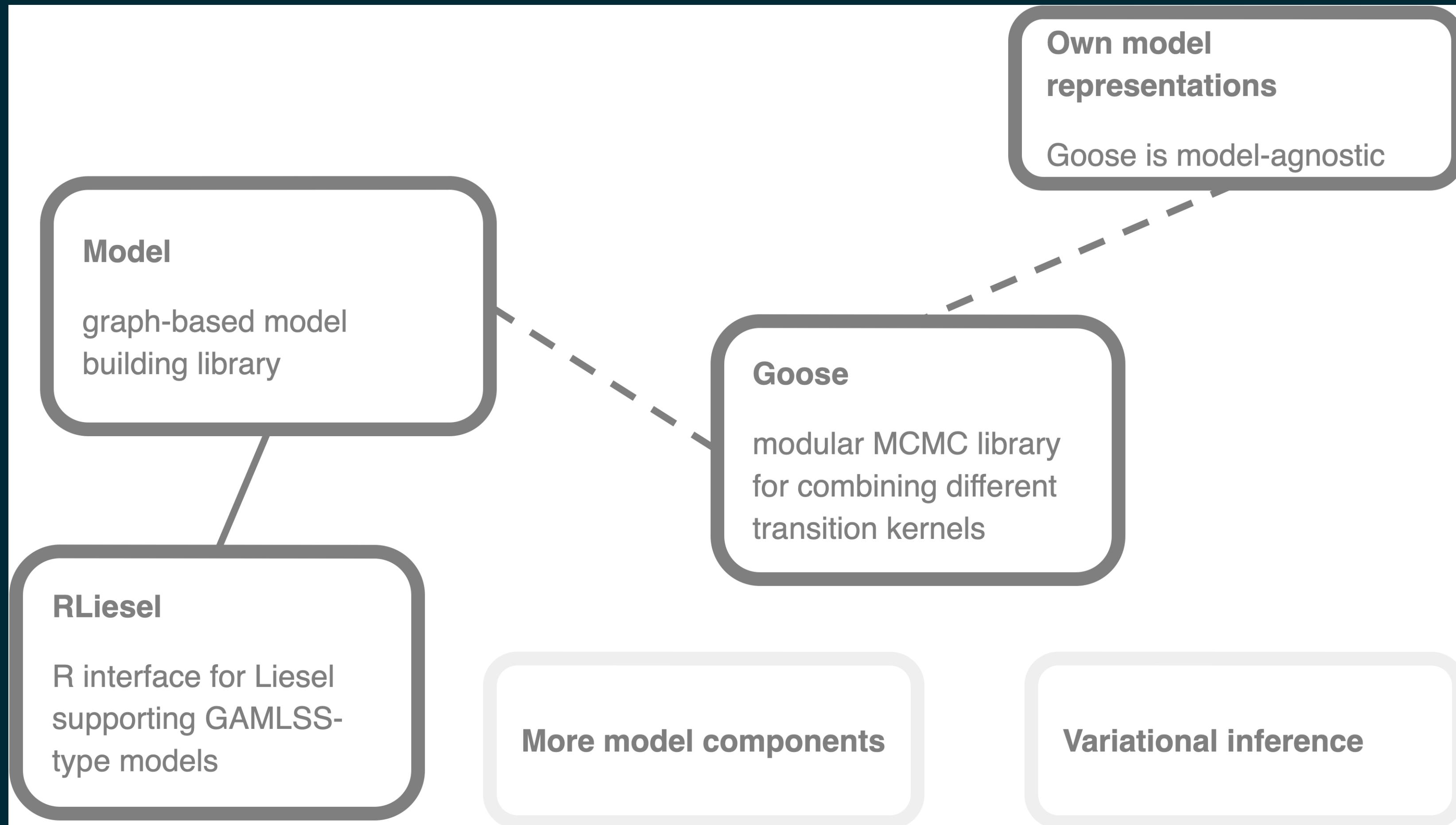
# WHAT IS LIESEL?

- Liesel is a software framework for research on semi-parametric regression models and Bayesian inference.
- Liesel aims to provide
  - easy modeling with full programmatic access to manipulate the model
  - an easy way to set up an MCMC sampler and the building blocks to tailor the inference algorithm for your problem
  - extensible
  - well-tested
- Liesel builds on good software.
- Liesel uses modern tools.
- Liesel is partially funded by the German Research Foundation (DFG) since April 2022 ❤️



# WHAT IS LIESEL?

components



# WHAT IS LIESEL?

- a Python package with an R interface
- based on the JAX machine learning library...
  - automatic differentiation
  - vectorization & parallelization
  - just-in-time compilation
  - CUDA support
- ...and BlackJAX
  - Goose wraps their HMC & NUTS algorithms (kernels)

# WHY NOT X?

X does not mean Twitter!

## STAN

- great probabilistic programming language which allows the user to specify their models
- models are compiled, cannot be examined with a REPL
- inference not as flexible, strong focus on MCMC with NUTS
  - models sometimes need to be re-parameterized to be sampled efficiently
  - no discrete parameters / no Gibbs sampling
- use them if they fit your problem

## GAMLSS / BAMLSS / MGCV

# DEVELOPERS



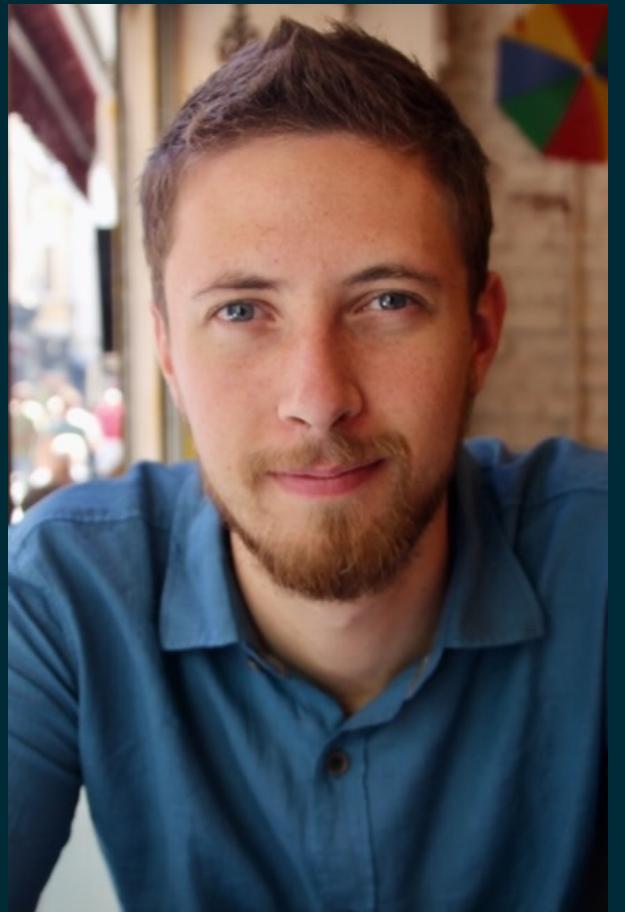
Hannes Riebl  
University of Göttingen



Paul Wiemann  
University of Wisconsin-Madison



Gianmarco Callegher  
University of Göttingen



Johannes Brachem  
University of Göttingen

# JAX

# INTRODUCTION TO JAX

- a library for numerical computing in Python
- the design follows numpy very closely
  - JAX uses the same concept of multi-dimensional arrays
  - JAX provides high-level mathematical functions Operating on those arrays
  - many functions have the same names and arguments as in numpy
  - sometimes there are subtle differences
- focused on performance and automatic differentiation
- developed by Google for machine learning

```
1 import jax
2 import jax.numpy as jnp
```

# KEY CONCEPTS

- functional programming paradigm
- transformable functions
- auto-vectorization and jit compilation
- automatic differentiation

# FUNCTIONAL PROGRAMMING PARADIGM

- functions need to be pure
- the output depends only on the input
- does not change the state of the world

## EXAMPLE

```
1 def pure(a, b):  
2     return a + b  
3  
4  
5 b = 2  
6 def non_pure(a):  
7     launch_rockets() # changes state of the world  
8     return a + b    # depends on state of the world
```

...

## ADVANTAGE

- pure functions are easier to reason about; for humans and compilers.
- enables jit, automatic differentiation, and more

# AUTOMATIC DIFFERENTIATION

A technique to algorithmically determine the partial derivatives of a function .

$$f : \mathbb{R}^M \rightarrow \mathbb{R}^N$$

- two types:
  - forward mode. More effective when  $M < N$
  - reverse mode. More effective when  $M > N$
  - in stats and ML, we usually have , e.g., loss function, log-likelihood, log unnormalized posterior and a potentially large  $N = 1$
  - for , reverse mode can evaluate in the same time as (up to a constant) but needs more memory
- `jax.grad` takes a scalar-valued function  $f$  as argument and calculates  $\nabla f$  using the reverse mode.

```

1 def ll_poisson(lambda_, x):
2     t0 = x * jnp.log(lambda_) - lambda_
3     t1 = -jax.scipy.special.gammaln(x + 1) # \Gamma(x) = (x - 1) !
4     return t0 + t1
5
6 dll_poisson = jax.grad(ll_poisson, argnums = 0)
7 dll_poisson(1.0, 2)
```

`Array(1., dtype=float32, weak_type=True)`

# SECOND ORDER DERIVATIVES

We can chain the transformations, for example, to get the second order derivative.

```
1 ddll_poisson = jax.grad(ll_poisson, argnums = 0)
2 ddll_poisson(1.0, 2)
```

```
Array(-2., dtype=float32, weak_type=True)
```

or we use the function `jax.hessian` (recommended) to calculate the Hessian matrix of .

```
1 hll_poisson = jax.hessian(ll_poisson, argnums = 0)
2 hll_poisson(1.0, 2)
```

```
Array(-2., dtype=float32, weak_type=True)
```

For details and pretty advanced details you might want to refer to the documentation:

- Taking derivatives with `grad`
- Advanced Automatic Differentiation in JAX
- The Autodiff Cookbook
- API Reference `jax.grad`

# JIT - JUST IN TIME COMPIRATION

- JAX can compile function such that recurrent calls are faster.
- this is done with the function transformation `jax.jit`

```
1 jll_poisson = jax.jit(ll_poisson)
2 jdll_poisson = jax.jit(dll_poisson)
3
4 @jax.jit
5 def my_func(a):
6     return a + 1.0
7
8 jll_poisson(1.0, 2)
```

```
Array(-1.6931474, dtype=float32, weak_type=True)
```

# MORE TRANSFORMATIONS

## VMAP

- use vmap to vectorize a function
- 
- $f : \mathbb{R} \rightarrow \mathbb{R}$  with  $g : \mathbb{R}^N \rightarrow \mathbb{R}^N$
- instead of looping over, vmap pushes the vectorization to the primitive operations for performance reasons
- no need to carry a batch dimension on all operations

## PMAP

SPMD - Single Programm Multiple Data

Lifts a jax programm to accelerators like GPUs which can enable faster computation

# JAX HAS SHARP BITS

- there are some sharp bits - most of them are (easily) avoidable
- however, this requires adaptation

## MOST NOTABLY

- constraints on python's control flow statements in jitted functions
  - for example, `if` does not work
  - however, there are alternatives `jax.lax.cond(cond, f_true, f_false)`
  - this is also true for loops, `for, while` have counterparts in `jax.lax`
- by default, single precision instead of double precision
- random numbers work differently
  - JAX uses pure functions, thus there is no global state
  - The state of the Pseudo random number generator (PRNG) needs to be passed
- refer → to the JAX' documentation [The Sharp Bits](#)

# EXTENDED EXAMPLE

One parameter Poisson model

## EXTENDED EXAMPLE

Consider iid observations of  $y_i$ . We place a prior on  $\lambda$ , namely  $\text{Gamma}(\lambda, \alpha)$ . The goal is to estimate the posterior mode of  $\lambda$ . The posterior mode is defined as the parameter value that maximises the posterior density.

### STEPS:

1. Determine the unnormalized posterior density.
2. Use Newton's method to find the posterior mode.
  - Newton's method requires that the parameter space is defined on the real line, but  $\lambda > 0$
  - Thus, we reparametrize the model with  $\psi = \log \lambda$
  - Change of variable theorem requires us to adjust the density

$$p(\psi | y) \propto \prod_{i=1}^5 p(y_i | \exp(\psi)) p(\exp(\psi)) \exp(\psi)$$



# IMPLEMENTATION STEPS

1. implement the log-likelihood for one observation
2. use vmap to transform the log likelihood into a vectorized function
3. implement the log prior
4. implement the log unnormalized posterior
5. implement the optimization loop

# IMPLEMENT THE LOG LIKELIHOOD

The probability mass function of a Poisson distribution random variable with rate parameter is

$$\Pr(Y = y|\lambda) = \frac{\lambda^y e^{-\lambda}}{Y!}$$

```

1 def ll_poisson(x, lambda_):
2     t0 = x * jnp.log(lambda_) - lambda_
3     t1 = -jax.scipy.special.gammaln(x + 1) # \Gamma(x) = (x - 1) !
4     return t0 + t1
5
6 ll_vec = jax.vmap(ll_poisson, (0, None)) # map along the first axis of the first argument, do not map over the second argument

```

Comment: Many jax functions are already vectorized. We would not need to use `vmap` here.

## ALTERNATIVE

There are different libraries that implement jax compatible distributions. Tensorflow probability is one of them.

```

1 import tensorflow_probability.substrates.jax as tfp
2
3 def ll_vec2(x, lambda_):
4     dist = tfp.distributions.Poisson(lambda_)
5     lps = dist.log_prob(x)
6     return lps
7
8 print(ll_vec(jnp.array([1, 2]), 1.5))
[-1.0945349 -1.3822172]
1 print(ll_vec2(jnp.array([1, 2]), 1.5))
[-1.0945349 -1.3822172]

```

# IMPLEMENT THE LOG PRIOR

```
1 import tensorflow_probability.substrates.jax.distributions as tfd  
2  
3 def log_prior(lambda_):  
4     dist = tfd.Gamma(1, 1)  
5     return dist.log_prob(lambda_)
```

and now in terms of

```
1 def log_prior_psi(psi):  
2     lambda_ = jnp.exp(psi)  
3     lp = log_prior(lambda_)  
4     cov_cor = psi  
5     return lp + cov_cor
```

# IMPLEMENT THE LOG POSTERIOR (UNNORMALIZED)

```
1 @jax.jit
2 def log_upost(ys, psi):
3     lambda_ = jnp.exp(psi)
4     ll = ll_vec(ys, lambda_).sum()
5     lprior = log_prior_psi(psi)
6
7     return ll + lprior
```

# NEWTON'S METHOD

Algorithm:

- Choose  $\psi^0$
- For do (or until convergence)  
 $t = 1, \dots, 10$

$$\psi^{t+1} = \psi^t - \frac{p'(\psi|y)}{p''(\psi|y)}$$

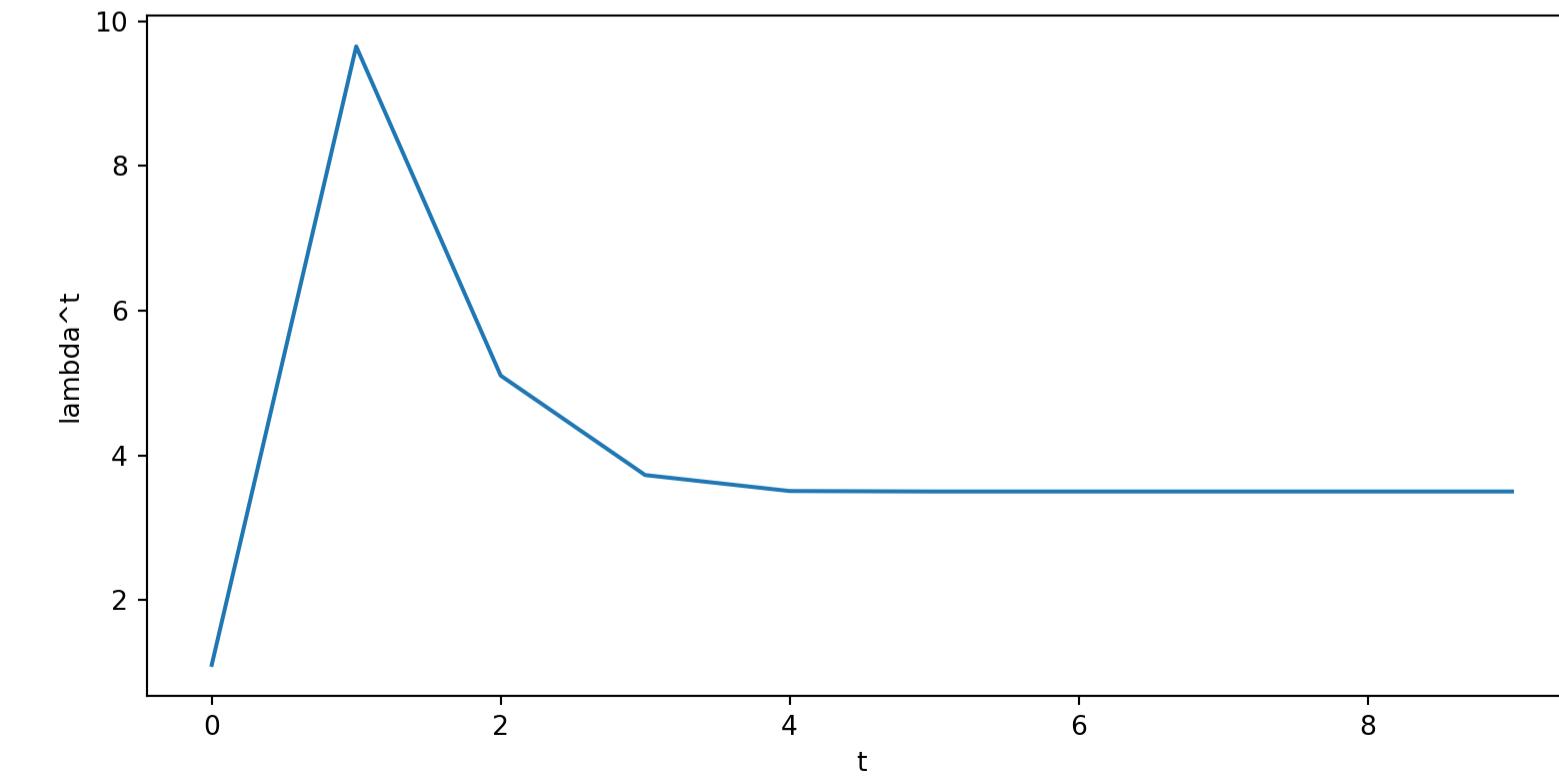
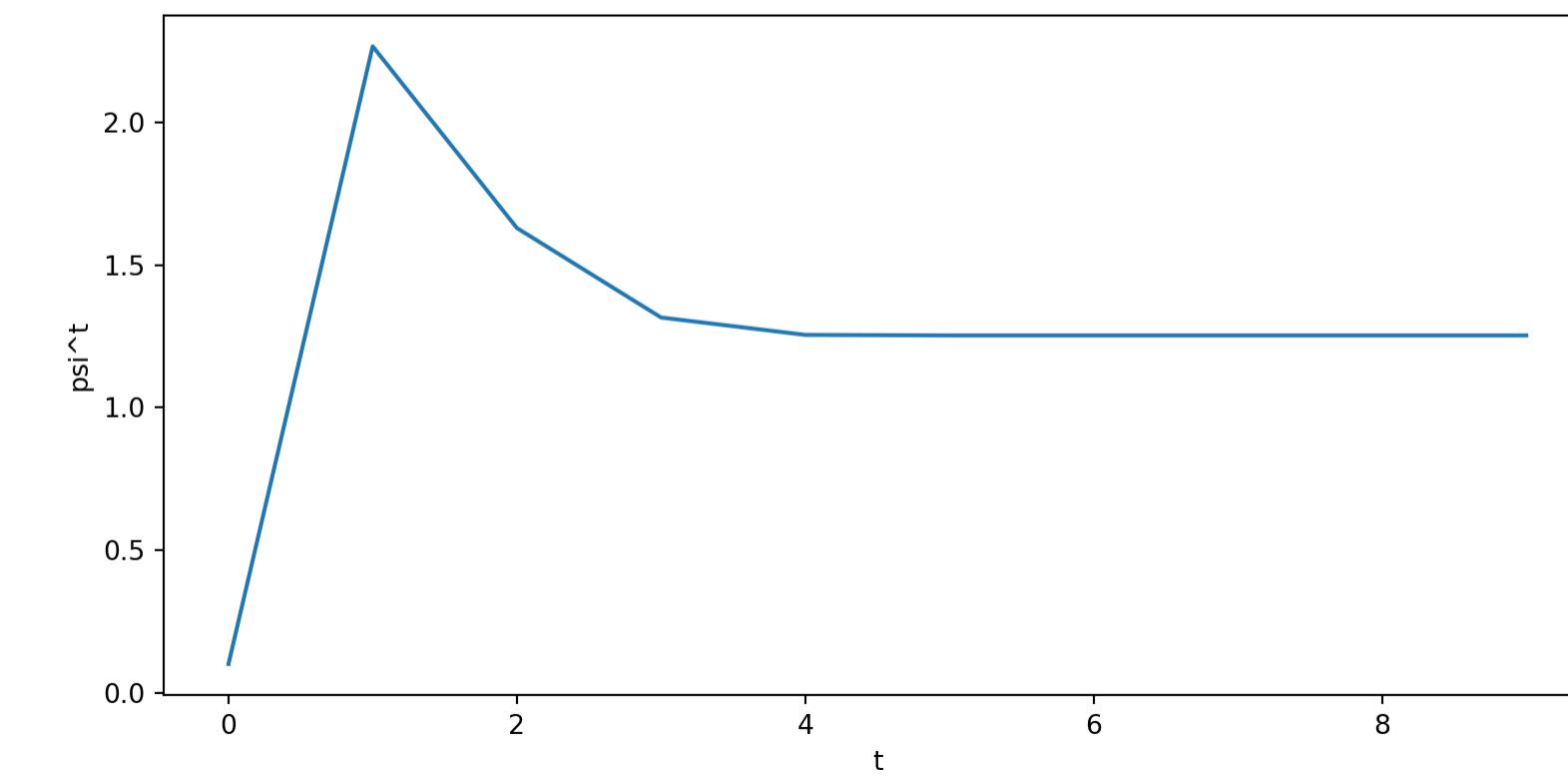
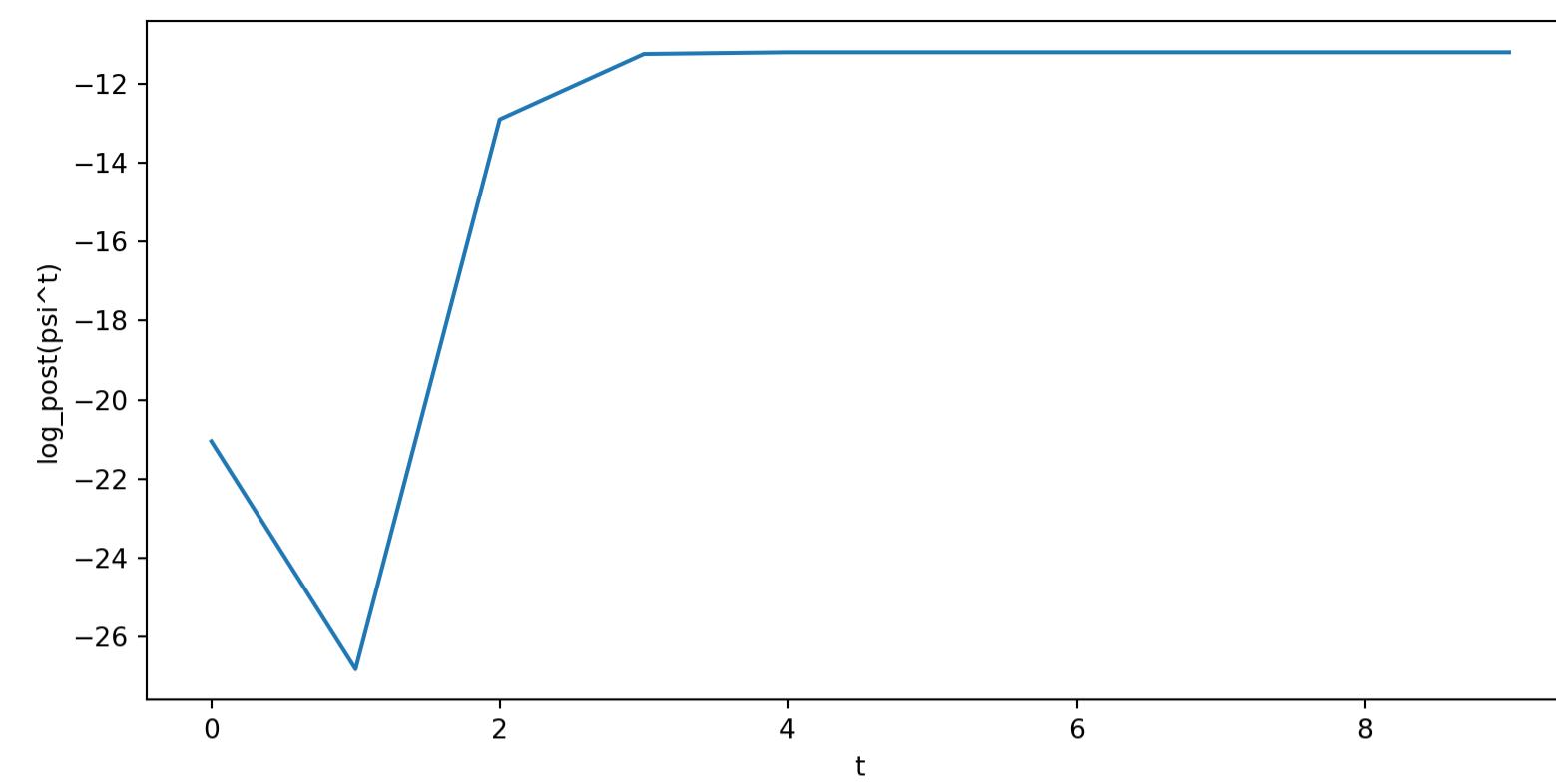
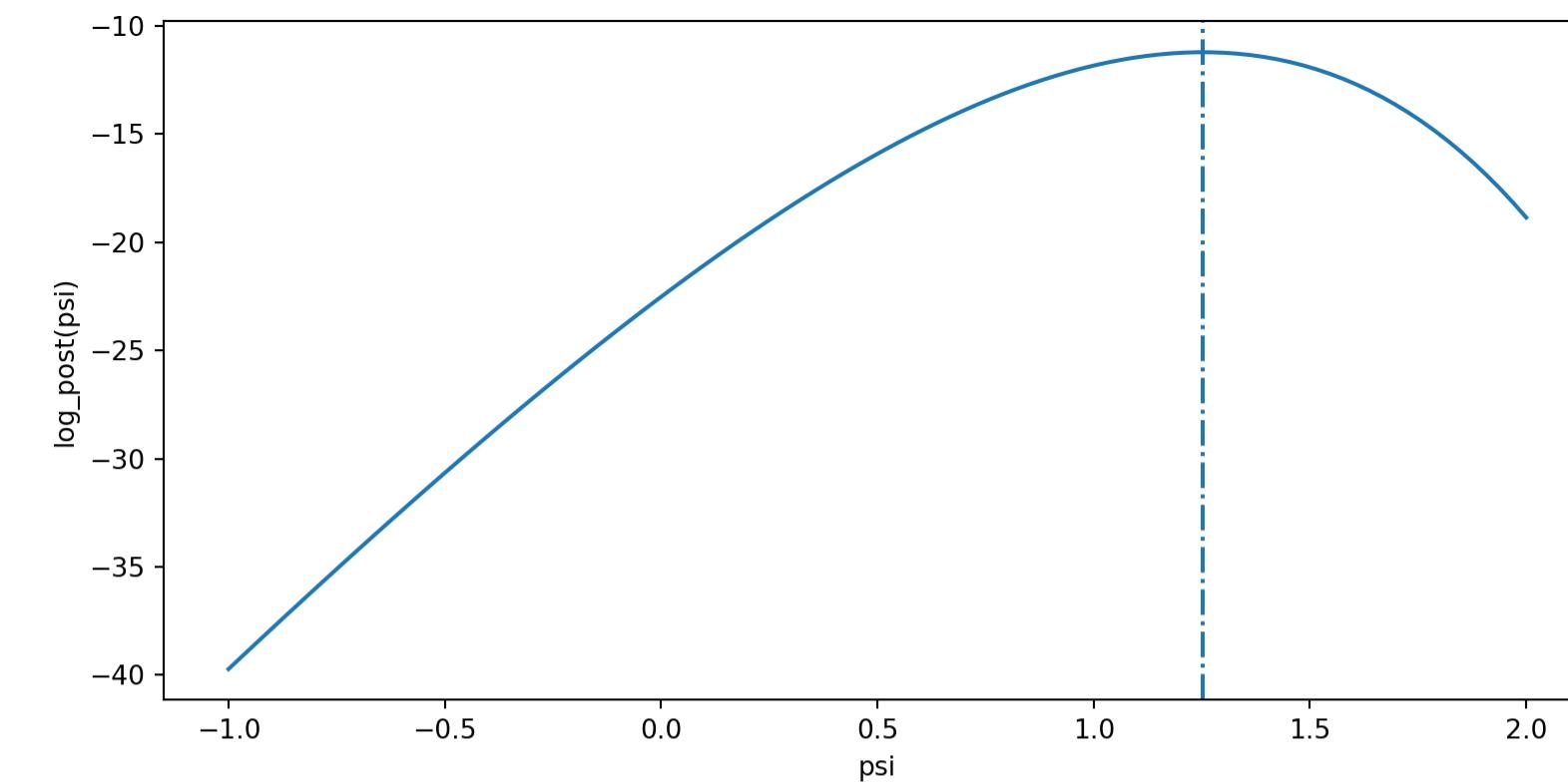
## IMPLEMENTATION

```

1  ys = jnp.array([3, 4, 4, 6, 3])
2  dlp = jax.grad(log_upost, argnums=1)
3  ddlp = jax.hessian(log_upost, argnums=1)
4
5  jdlp = jax.jit(dlp)
6  jddlp = jax.jit(ddlp)
7
8  psis = []
9  lambdas = []
10 log_uposts = []
11
12 psi = 0.1
13 for i in range(10):
14     psis.append(psi)
15     lambdas.append(jnp.exp(psi))
16     log_uposts.append(log_upost(ys, psi))
17     psi = psi - jdlp(ys, psi) / jddlp(ys, psi)

```

# PLOT



# CODE TO CREATE THE PLOTS

```
1 import matplotlib.pyplot as plt
2
3 psi_grid = jnp.linspace(-1, 2, 100)
4 l_upost_grid = []
5 for psi in psi_grid:
6     l_upost_grid.append(log_upost(ys, psi))
7
8 fig, axs = plt.subplots(2, 2)
9 axs[0][0].plot(psi_grid, l_upost_grid)
10 axs[0][0].set_xlabel("psi")
11 axs[0][0].set_ylabel("log_post(psi)")
12 axs[0][0].axvline(psis[-1], linestyle='-.')
13
14 axs[0][1].plot(log_uposts)
15 axs[0][1].set_xlabel("t")
16 axs[0][1].set_ylabel("log_post(psi^t)")
17
18 axs[1][0].plot(psis)
19
```

# EXERCISE SHEET

Please start working on the exercise sheet

