Optimizing the cost of gradient computation with JAX for Experience with *Tunax*

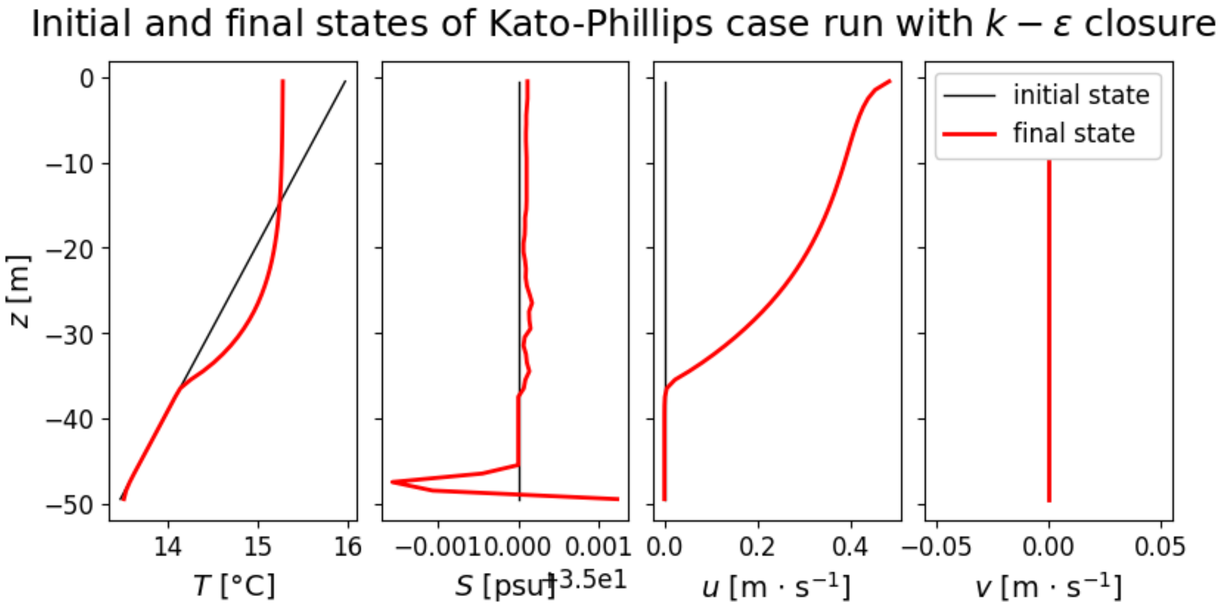
Gabriel Mouttapa

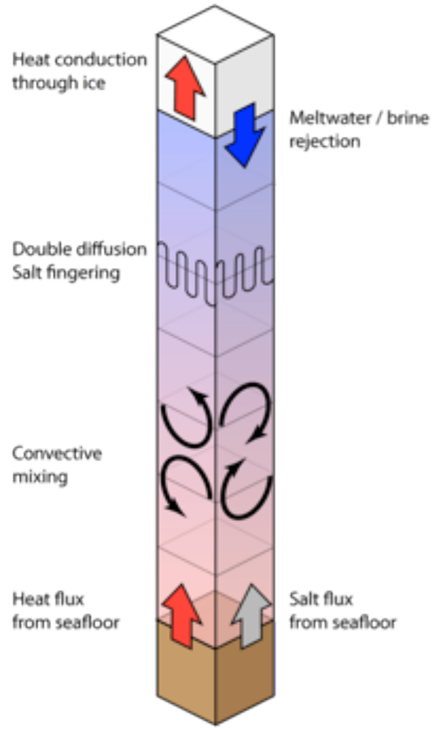
Jaxathon March 12th 2025

Tunax

Ocean Single Column Model

- 1D vertical
- **variables** : T, S, b, u, v
- heat, wind, freshwater **forcings**
- turbulent kinetic diffusion -> physical **closure**
  - TKE, $k - \varepsilon$, KPP, …
  - physical parameters to calibrate $\theta \in \mathbb{R}^d$

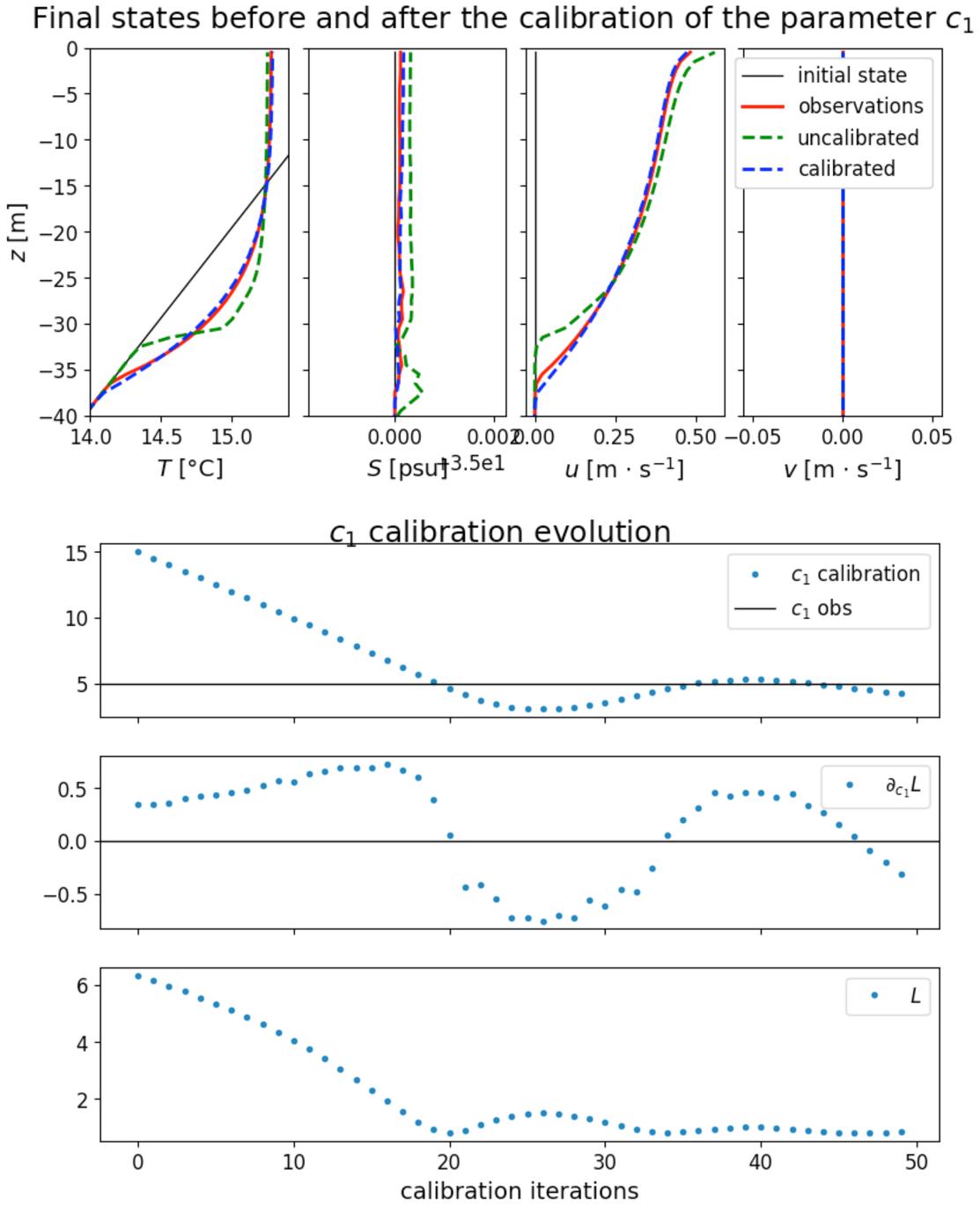




Tunax

Calibration module

- best value of $\theta \in \mathbb{R}^d$ to fit a **database**
- **differentiable** calibration -> **JAX**
- use of `jax.grad` on the whole SCM
- databases
  - perfect model
  - observations
  - **Large Eddy Simulations** (LES)



Tunax

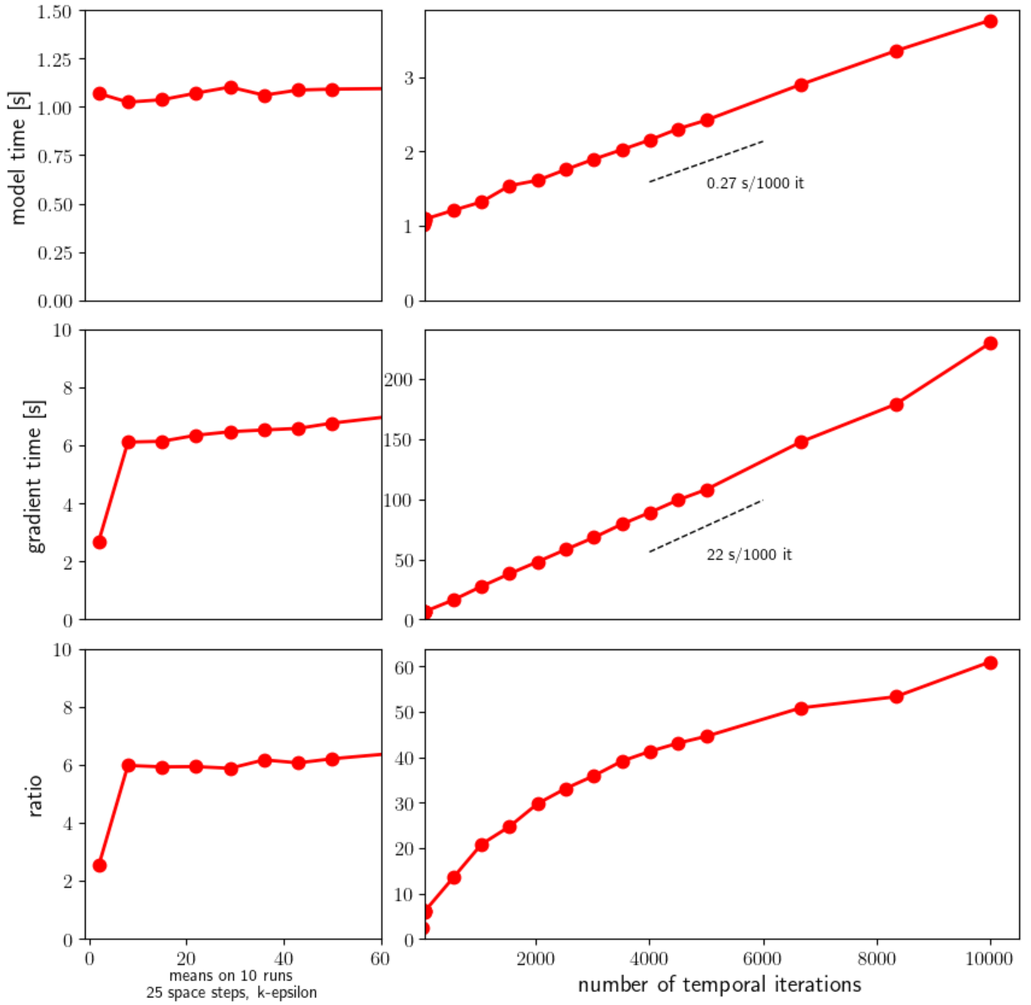
Calculation cost issue

- for non-trivial databases : explosion of calibration

  cost

- issue : gradient on an **iterative** function
  - $U_\theta^n = \mathscr{M}_\theta(U^0) = \mathcal{S}_\theta^n(U^0)$

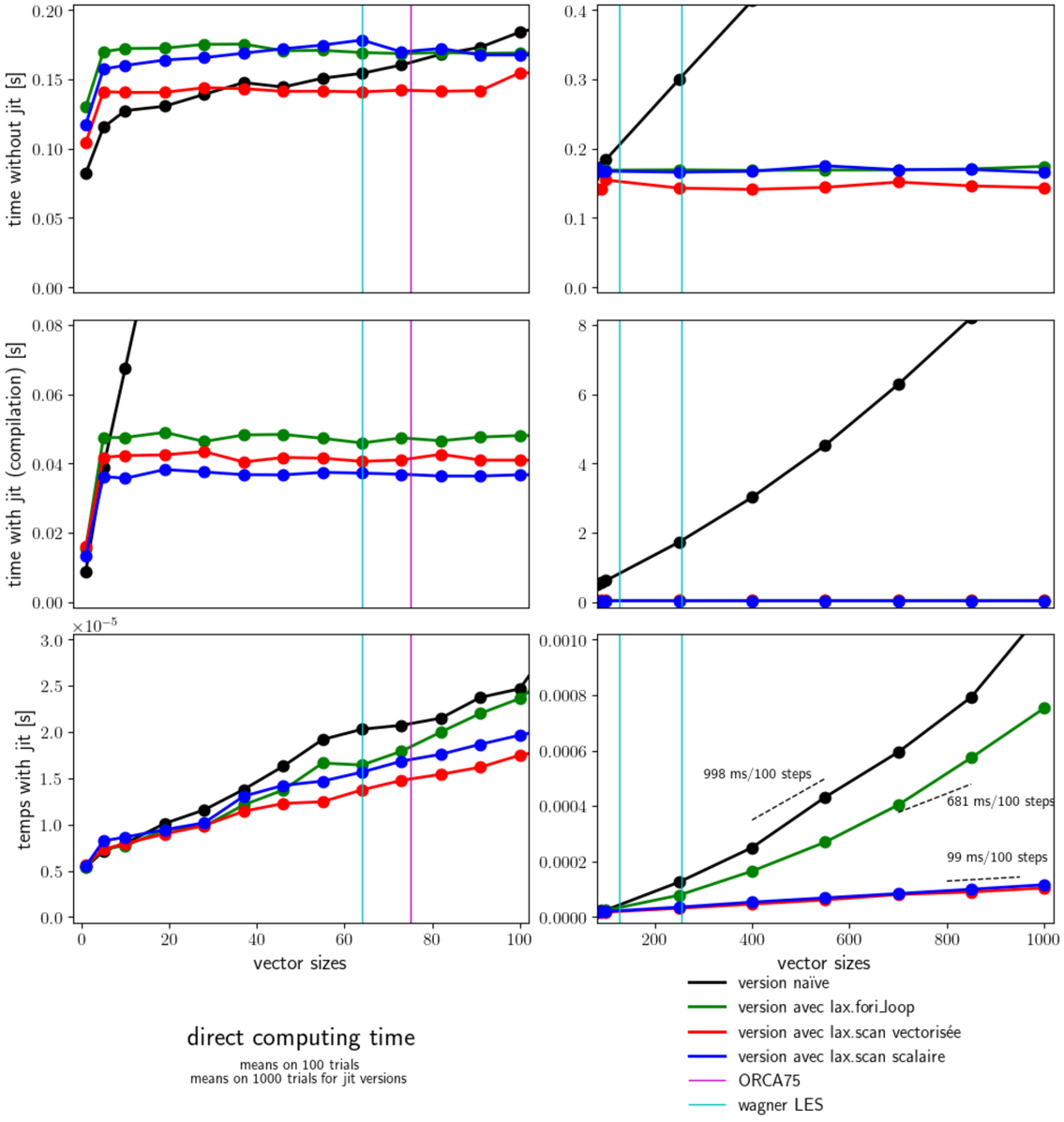
- **ratio** between model and gradient $\simeq$ 30 instead of 3



Optimizing gradient cost

Using `jax.lax`

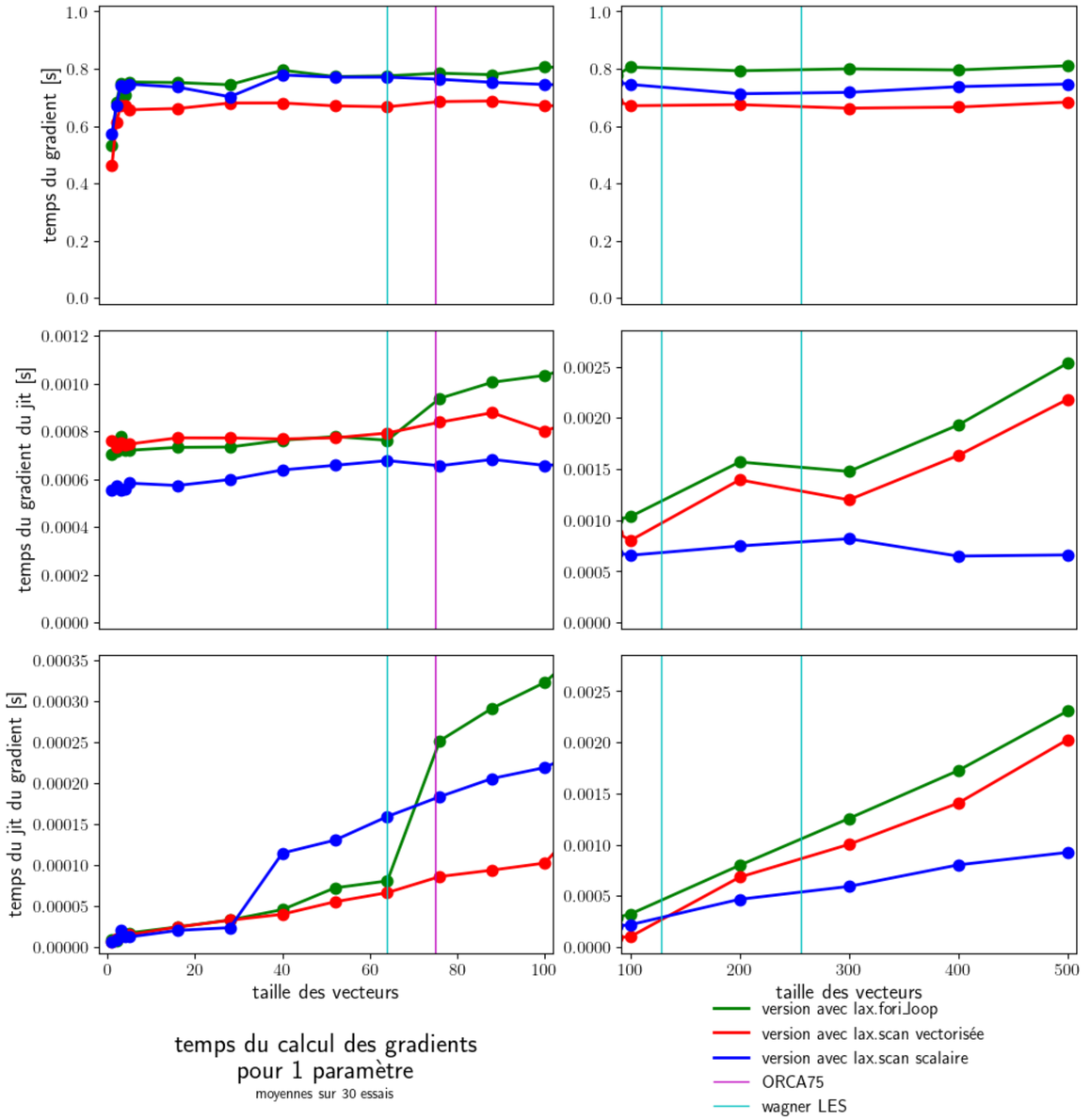
- work on inner **vectorized functions**
  - `jax.lax.scan` > `jax.lax.fori_loop`
- work on **integration loop**
  - hard to implement but efficient (ratio 3)



Optimizing gradient cost

jit∘grad vs. grad∘jit

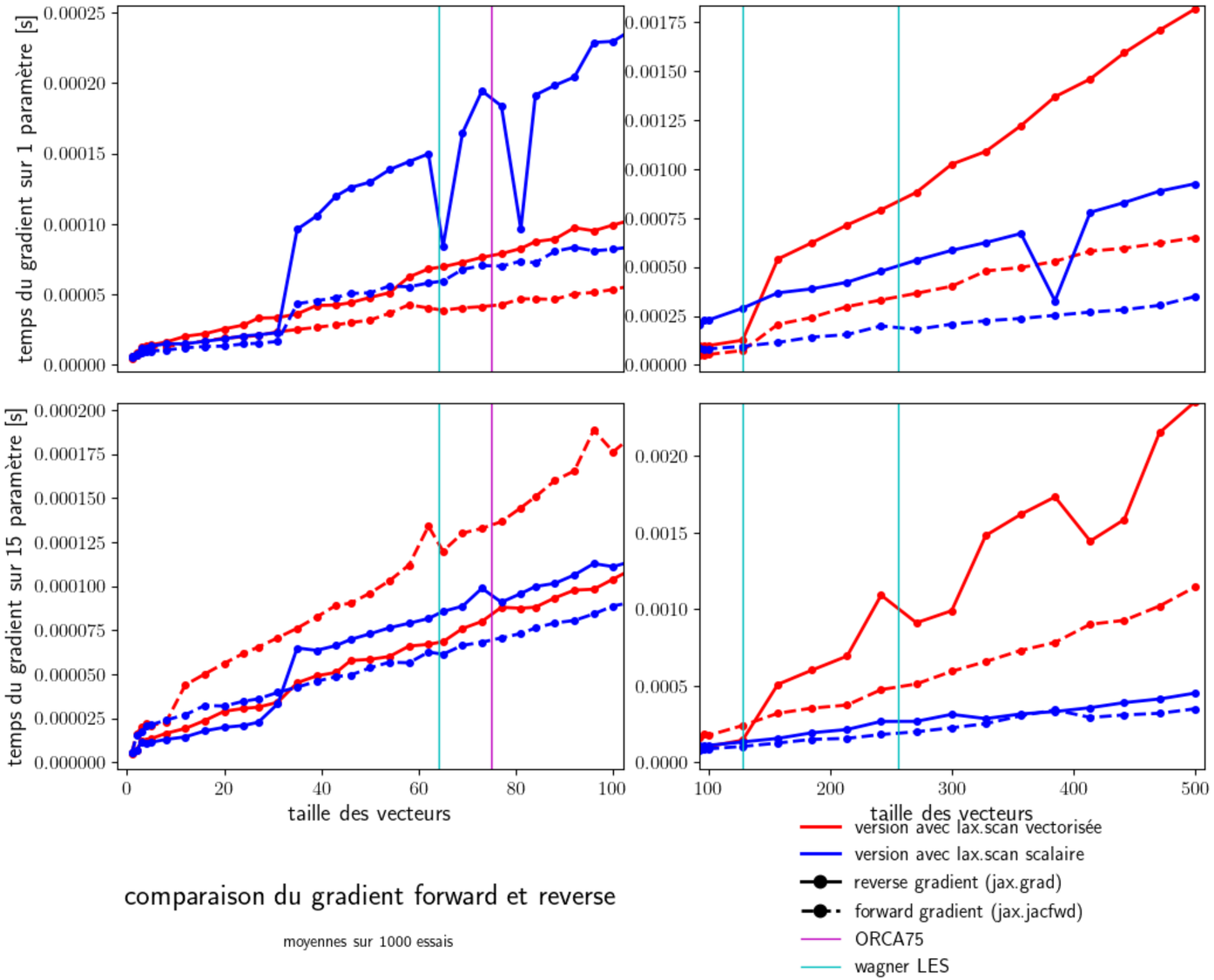
- **no jit**
  - usually too long
- **grad∘jit**
  - user friendly (same function for direct model and gradient)
  - faster compilation
  - good for small functions
- **jit∘grad**
  - faster -> best choice



Optimizing gradient cost

Reverse vs. forward

- **reverse gradient**
  - use of memory for long and complex functions -> potential bottleneck
- **forward gradient** (`jax.jacfwd`)
  - faster when more outputs than inputs



Optimizing gradient cost

Results

- results on surrogate model with
  - forward gradient
  - `jax.lax.scan` on vectorized functions and integration loop
  - jitification ouside gradient
- **ratio model/gradient 30 -> 3**
- **Next steps**
  - study of memory cost
  - checkpointing for reverse gradient