Differentiable programming for experiment design optimization in precision physics experiments

Mateusz Bawaj, Greta Tosti, Francesco De Marco

mateusz.bawaj@unipg.it

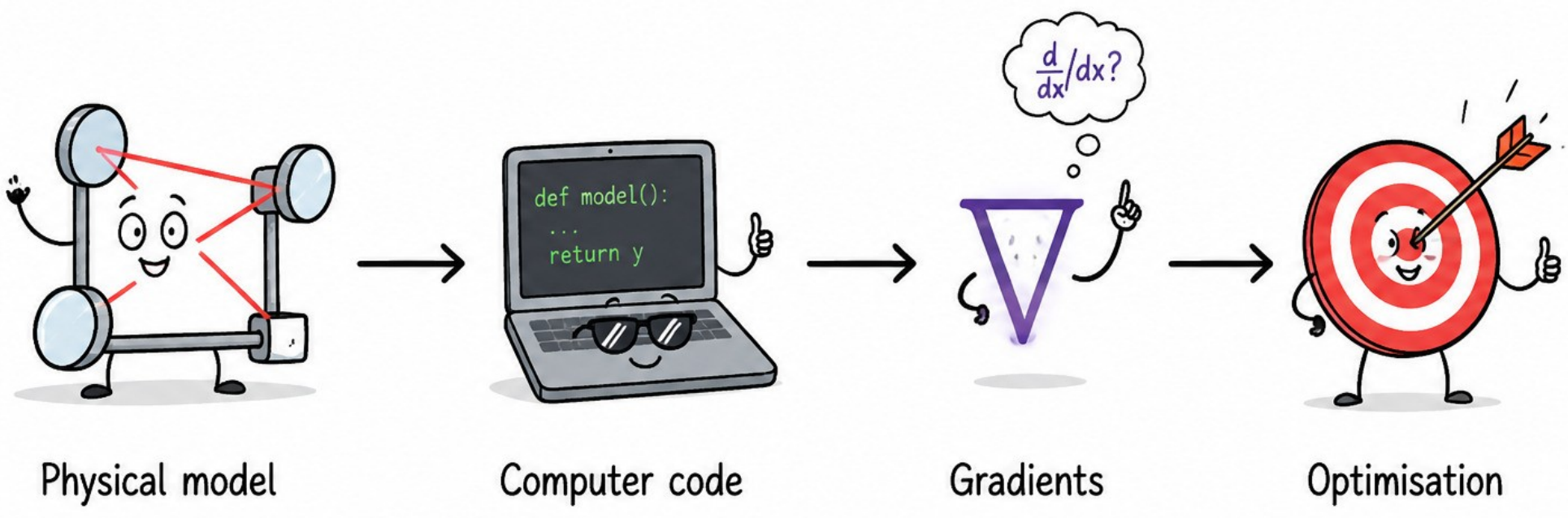
Perugia 16/05/2026

KAGRA International Workshop

A.D. 1308
unipg
UNIVERSITÀ DEGLI STUDI
DI PERUGIA

INFN
Sez. di Perugia

How does it work



Differentiable programming

- can be applied to algebraic functions and to the computer programs;
- it is based on the simultaneous computation of a value and a related derivative;
- it does not provide explicit formula of the derivative.

```
In [1]: def f(x):
            return 64 *(1-x) *(1-2*x)^2 *(1-8*x+8*x*x)^2
```

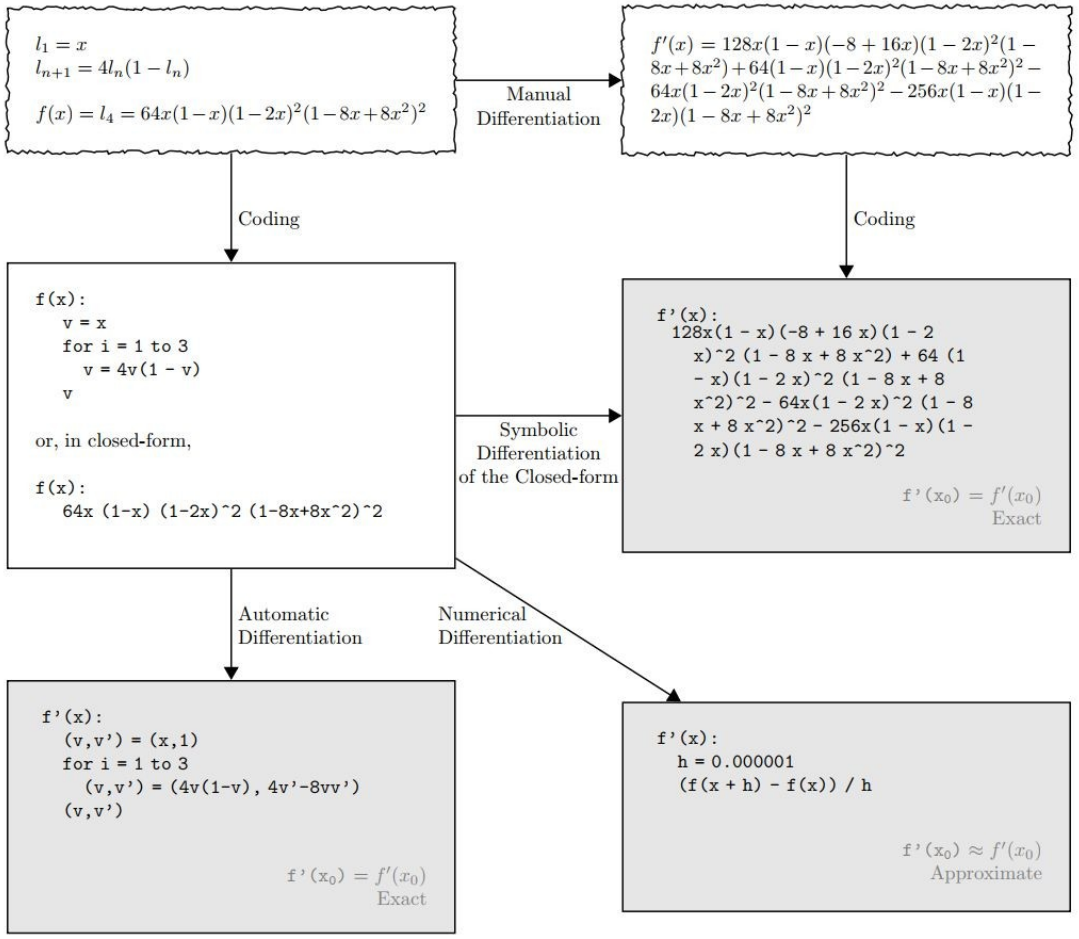
but also

```
In [2]: def f(x,n):
            if n == 1:
                return x
            else:
                v = x
                for i in range(1,n):
                    v = 4*v*(1-v)
                return v
```

Differentiable programming

- automatic differentiation
- algorithmic differentiation
- autodiff
- algodiff
- autograd
- AD

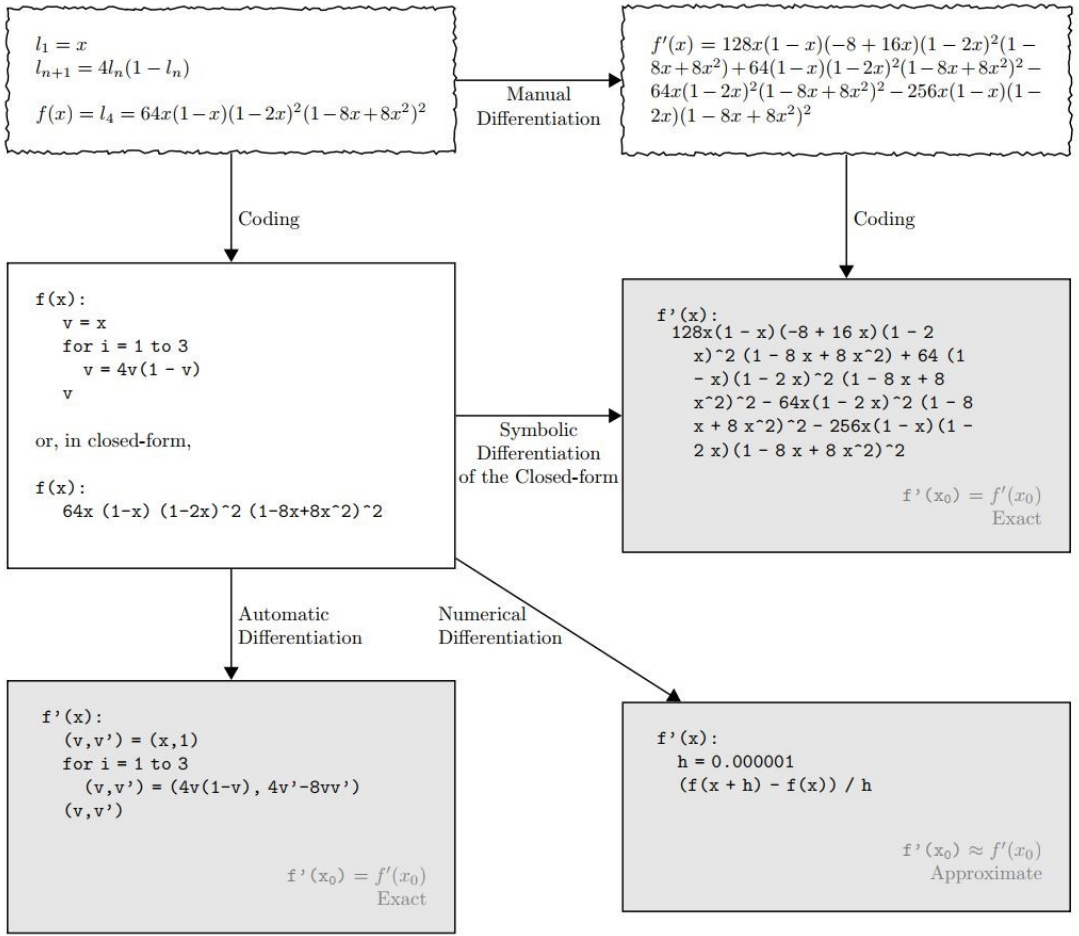
various names come from
different communities as the idea is old.



Baydin, Pearlmutter, Radul, Siskind. 2018. “Automatic Differentiation in Machine Learning: a Survey.” Journal of Machine Learning Research (JMLR)

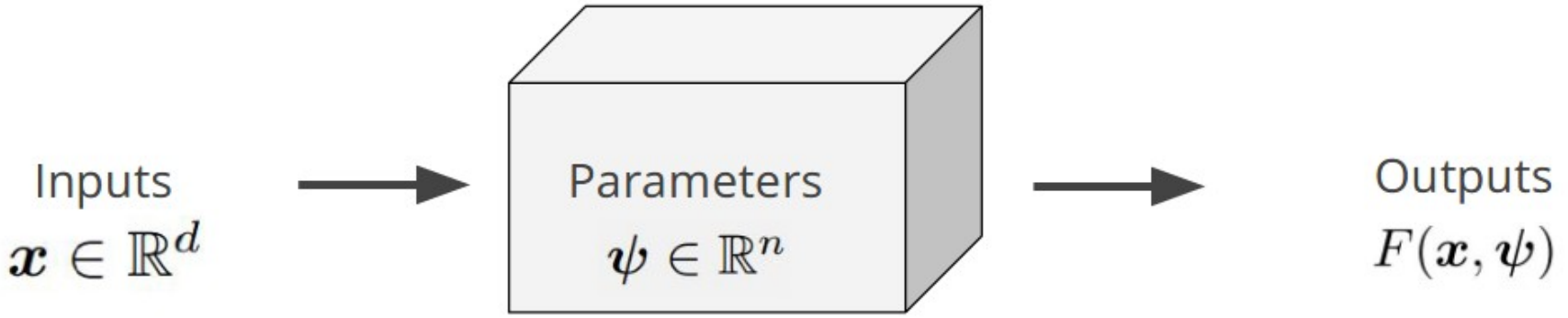
Differentiable programming

- manual
- symbolic differentiation
- numerical differentiation
- automatic differentiation



Baydin, Pearlmutter, Radul, Siskind. 2018. “Automatic Differentiation in Machine Learning: a Survey.” Journal of Machine Learning Research (JMLR)

Design optimization



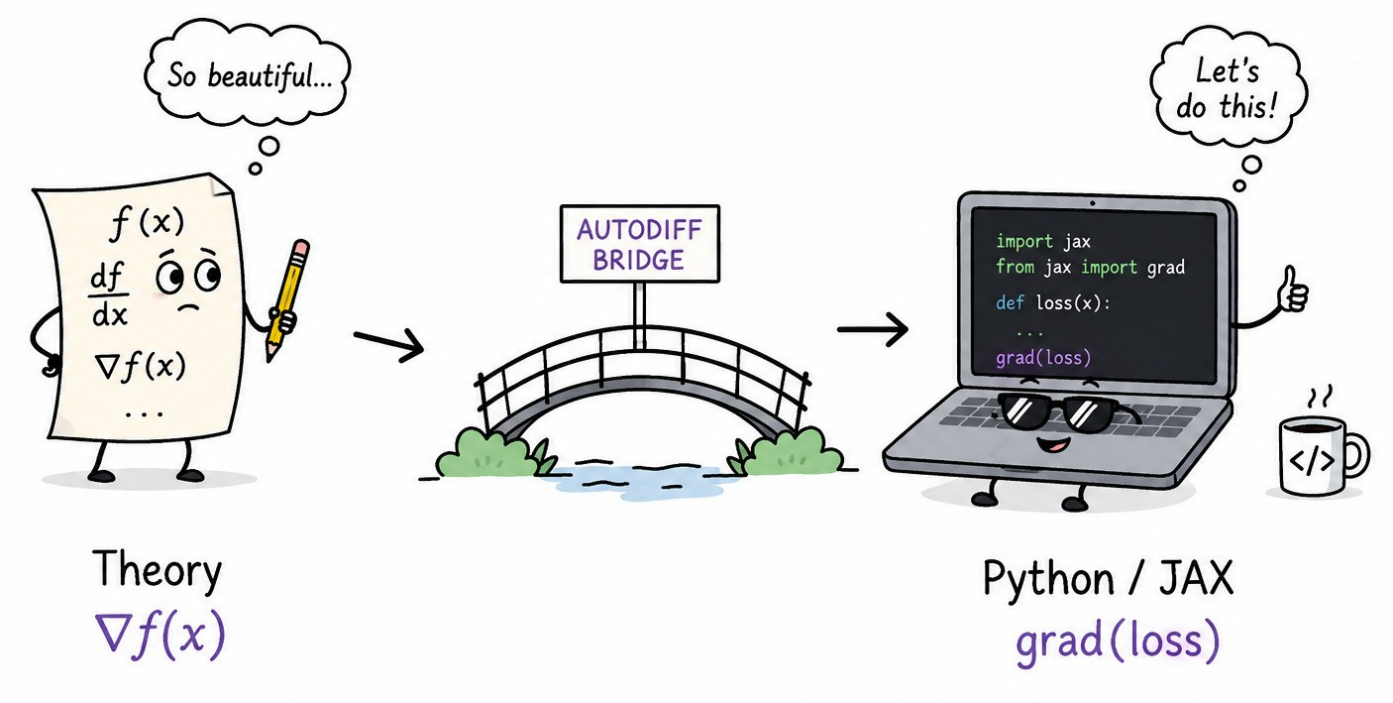
Simulator (model) of the system

$$\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}} \sum_{\boldsymbol{x}} \mathcal{R}(F(\boldsymbol{x}, \boldsymbol{\psi}))$$

Can be efficiently found by gradient-based optimization if $\nabla_{\boldsymbol{\psi}} \mathcal{R}$ is available

from Atılım Güneş Baydin (University of Oxford)

In practice


So beautiful...
$f(x)$
$\frac{df}{dx}$
$\nabla f(x)$
...
AUTODIFF BRIDGE
Let's do this!
import jax
from jax import grad
def loss(x):
...
grad(loss)
</>
Theory
$\nabla f(x)$
Python / JAX
grad(loss)

```python
def objective(v):
    """A smooth non-convex 2D objective with minimum at (1, 1)."""
    x, y = v
    return 20 * (1.0 - x) ** 2 + 100.0 * (y - x**2) ** 2
```

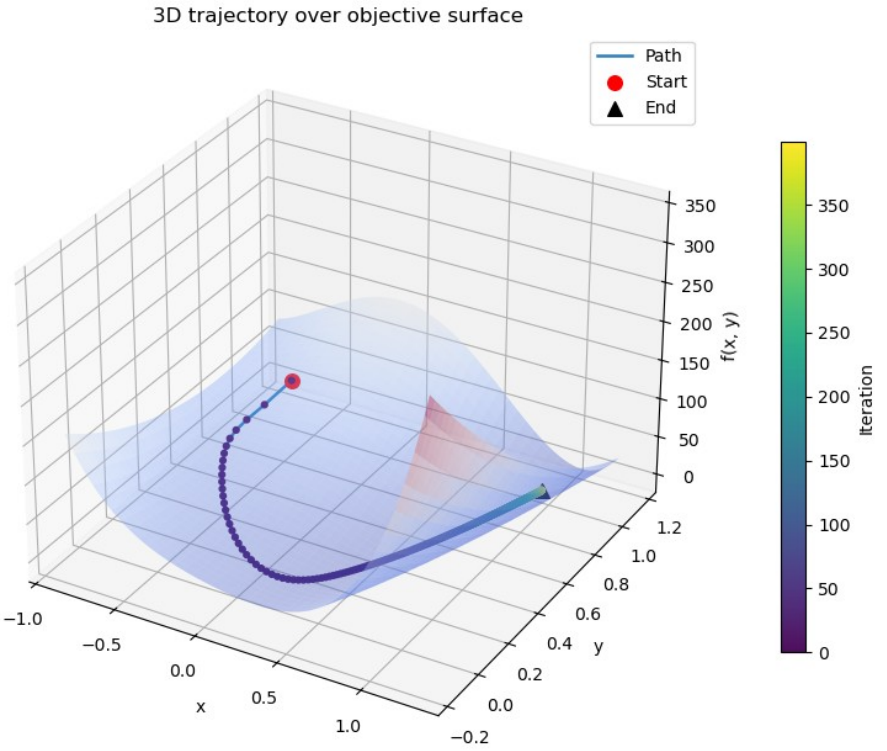
a function to search for its minimum

```python
value_and_grad = jax.value_and_grad(objective)

# lax.scan requires a step function with signature (carry, x_t) -> (carry, y_t).
# Here:
# - carry is the current parameter vector x_curr
# - x_t is unused (we pass xs=None, so scan just repeats 'length=steps' times)
# - y_t stores diagnostics per iteration: objective value and updated parameters
def _step(x_curr, _):
    value, grad = value_and_grad(x_curr)
    x_next = x_curr - lr * grad
    return x_next, (value, x_next)

# We wrap scan in a tiny lambda so x_init remains the runtime input, while
# 'steps' stays captured as a static Python integer for compilation.
run_optimization = jax.jit(
    lambda x_init: jax.lax.scan(_step, x_init, xs=None, length=steps)
)
x_opt, (history, trajectory) = run_optimization(x)
```

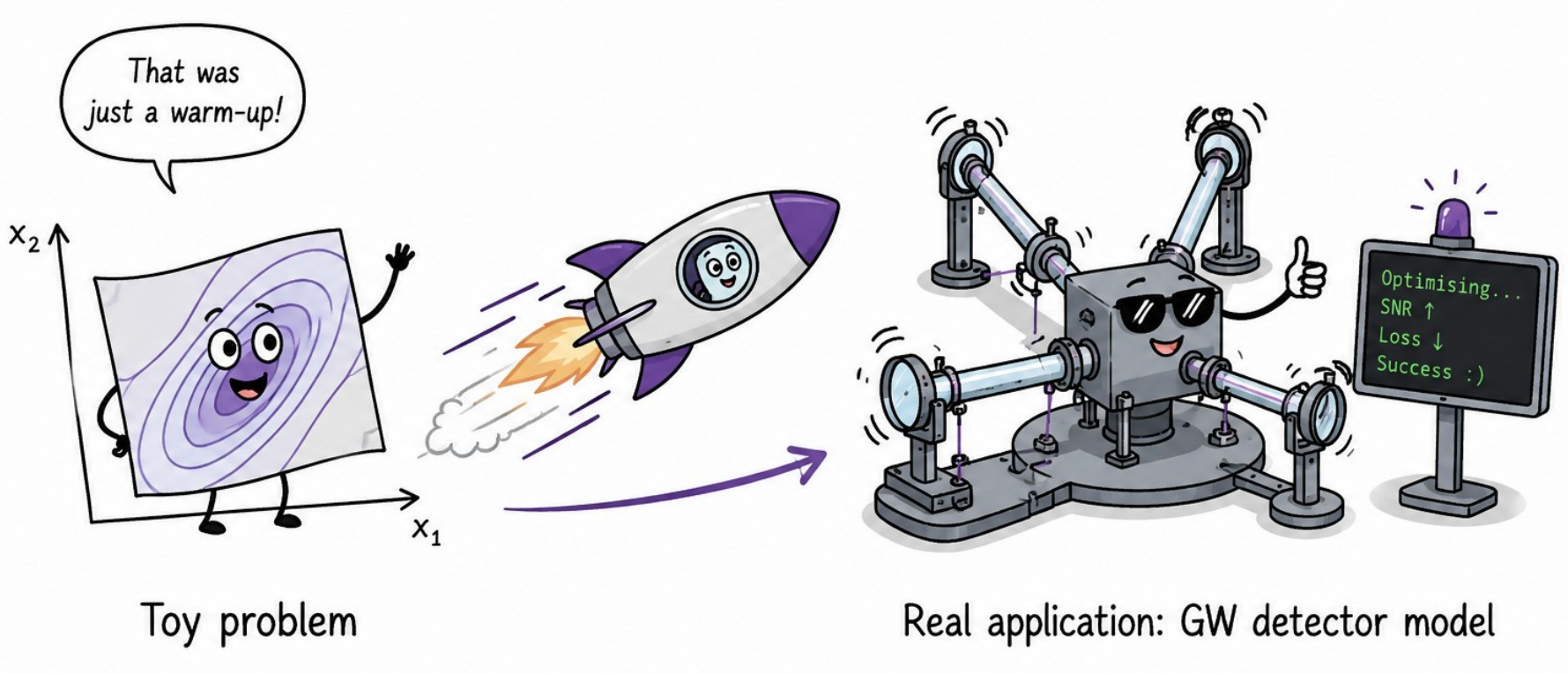
the search algorithm

Interactive

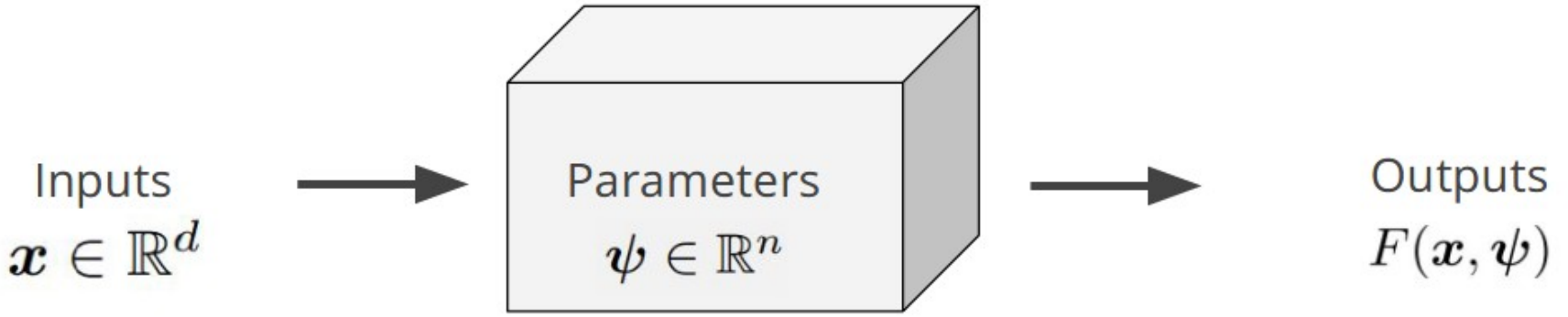
```
$ python back_propagation_examples/jax_backprop_example.py
```



More interesting example



Optimisation on a real model

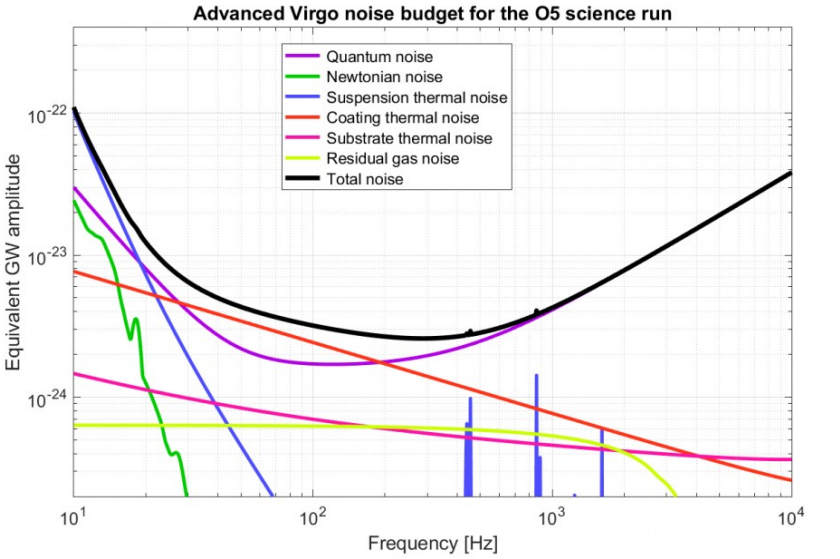


Numerical modeling of quantum noise in gravitational wave detectors *Greta Tosti*

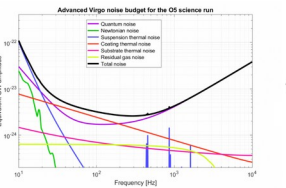
09:20 - 09:40

why not to try?

Model + loss function



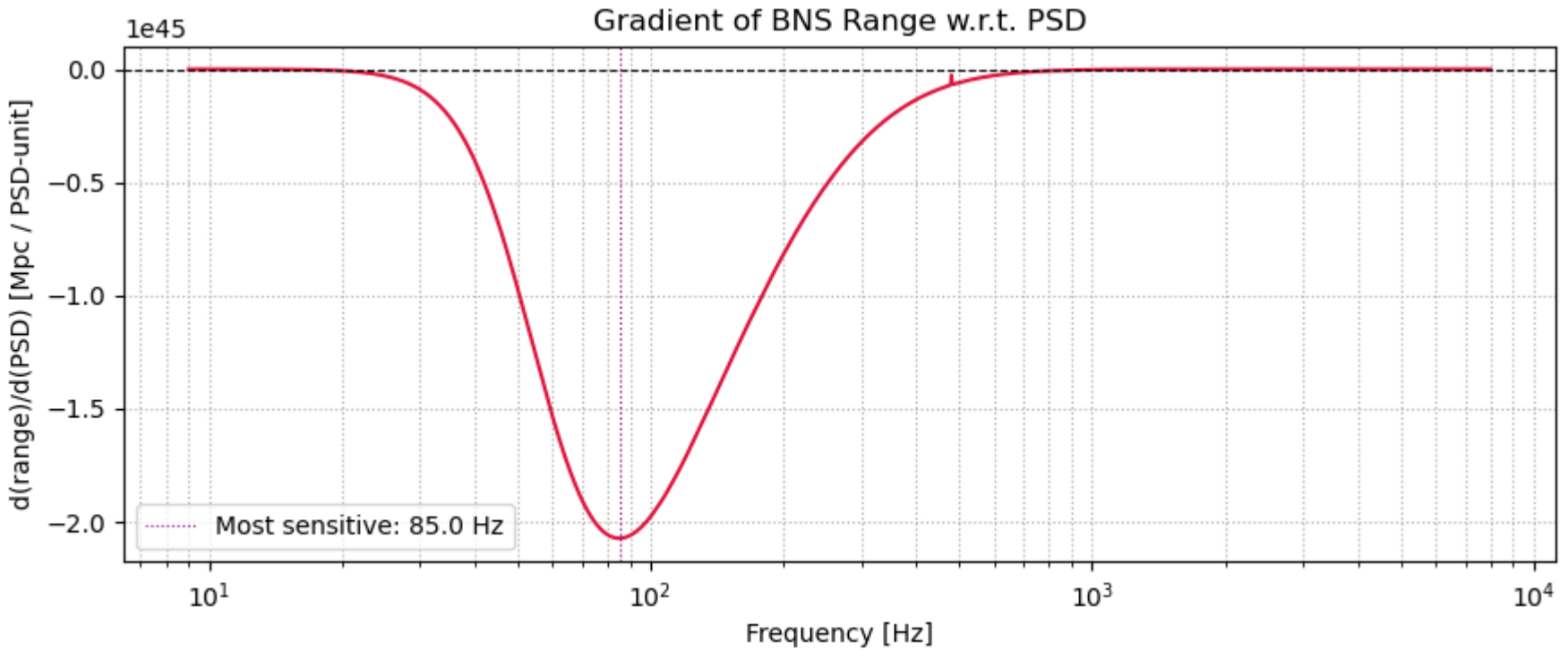

```
+ inspiral_range.range(PSD)
= a number
```

if only we could compute ∇_{ψ} ( `+ inspiral_range.range(PSD)`)

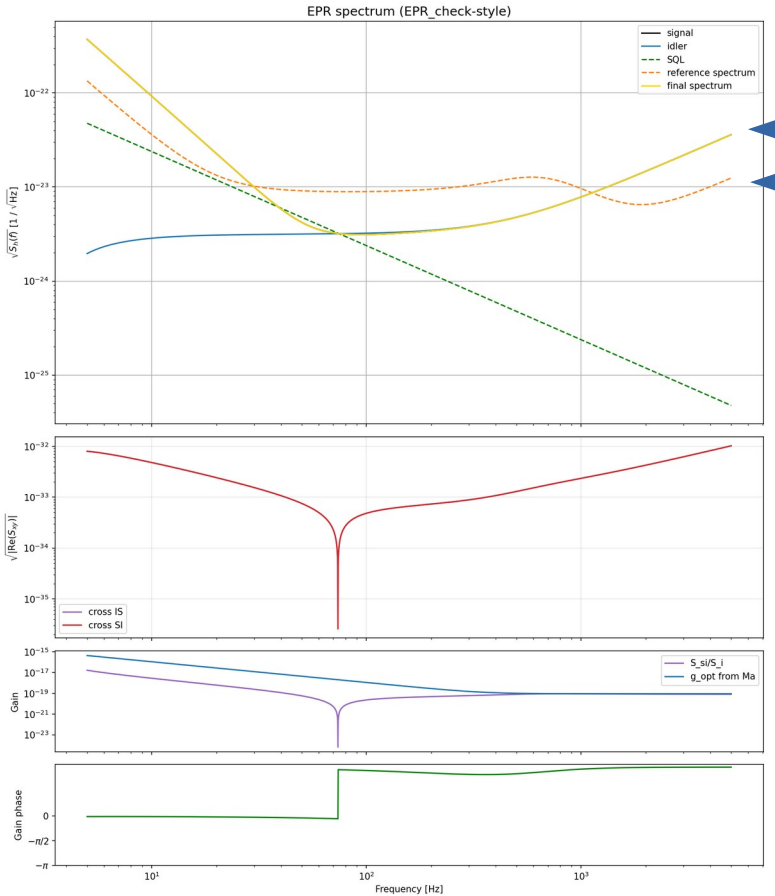
Auto-diff first outcome

know your loss function better

```
grad_psd_fn = jax.jit(jax.grad(lambda p: bns_range(freq, p)))
grad_psd = grad_psd_fn(psd)
plt.loglog(freq, grad_psd, color="crimson")
```

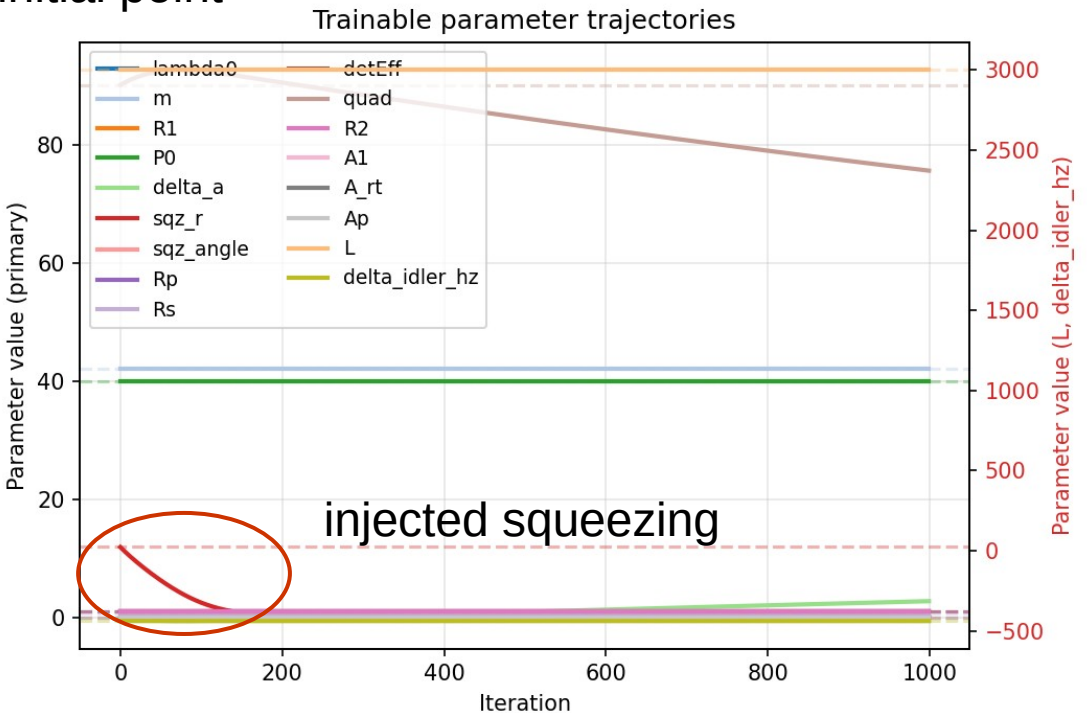


Optimisation



optimised point

initial point



The start and the end

Initial point:

```
lambda0 = 1.064 *
      m = 42 *
     R1 = 0.986 *
      L = 2999.8 *
     P0 = 40 *
delta_a = 0 *
  sqz_r = 12 *
sqz_angle = 0 *
     Rp = 0.95 *
     Rs = 0.6 *
   Lrec = 11.958
 detEff = 1 *
   quad = 90 *
     R2 = 1 *
     A1 = 0 *
   A_rt = 0 *
     Ap = 0 *
delta_idler_hz = -438.28 *
```

Optimized point:

```
lambda0 = 1.064 *
      m = 42 *
     R1 = 0.986 *
      L = 2999.8 *
     P0 = 40 *
delta_a = 2.716026 *
  sqz_r = 3.9508612e-19 *
sqz_angle = 0 *
     Rp = 0.95 *
     Rs = 0.6 *
   Lrec = 11.958
 detEff = 1 *
   quad = 75.589533 *
     R2 = 1 *
     A1 = 0 *
   A_rt = 0 *
     Ap = 0 *
delta_idler_hz = -438.28 *
```

More information

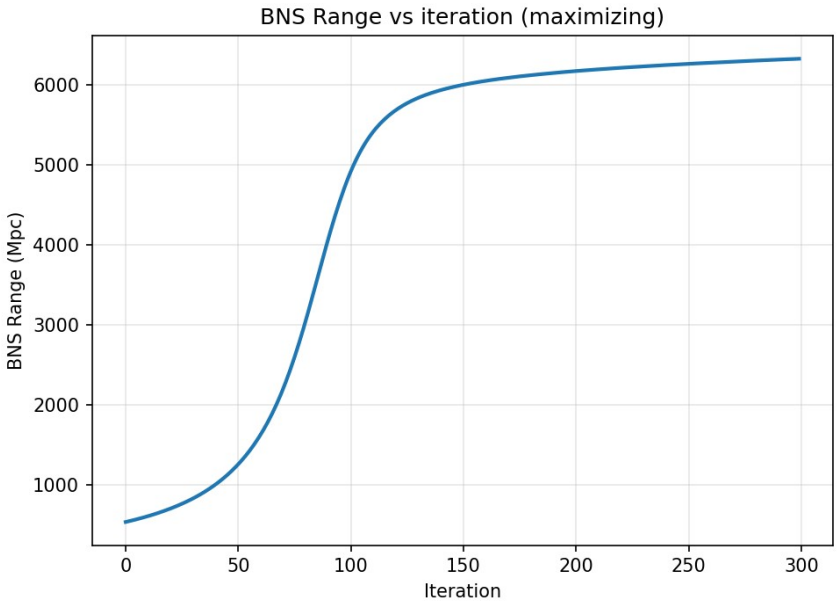
- much more details in the lecture: *Differentiable programming and design optimization* by Atılım Güneş Baydin (University of Oxford)

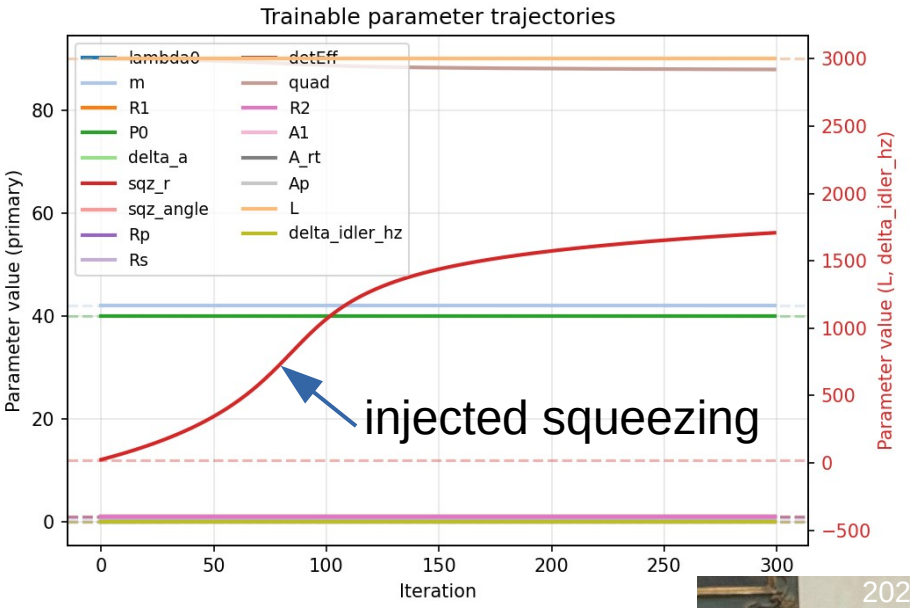
  pay attention to the multi-dimensional optimization aspect!

- entry level implementation explanation - Saupin Guillaume Blog

Thank you for your attention

Plot-twist: “double-check your model”





still do not trust it! ;-)

The start and the end

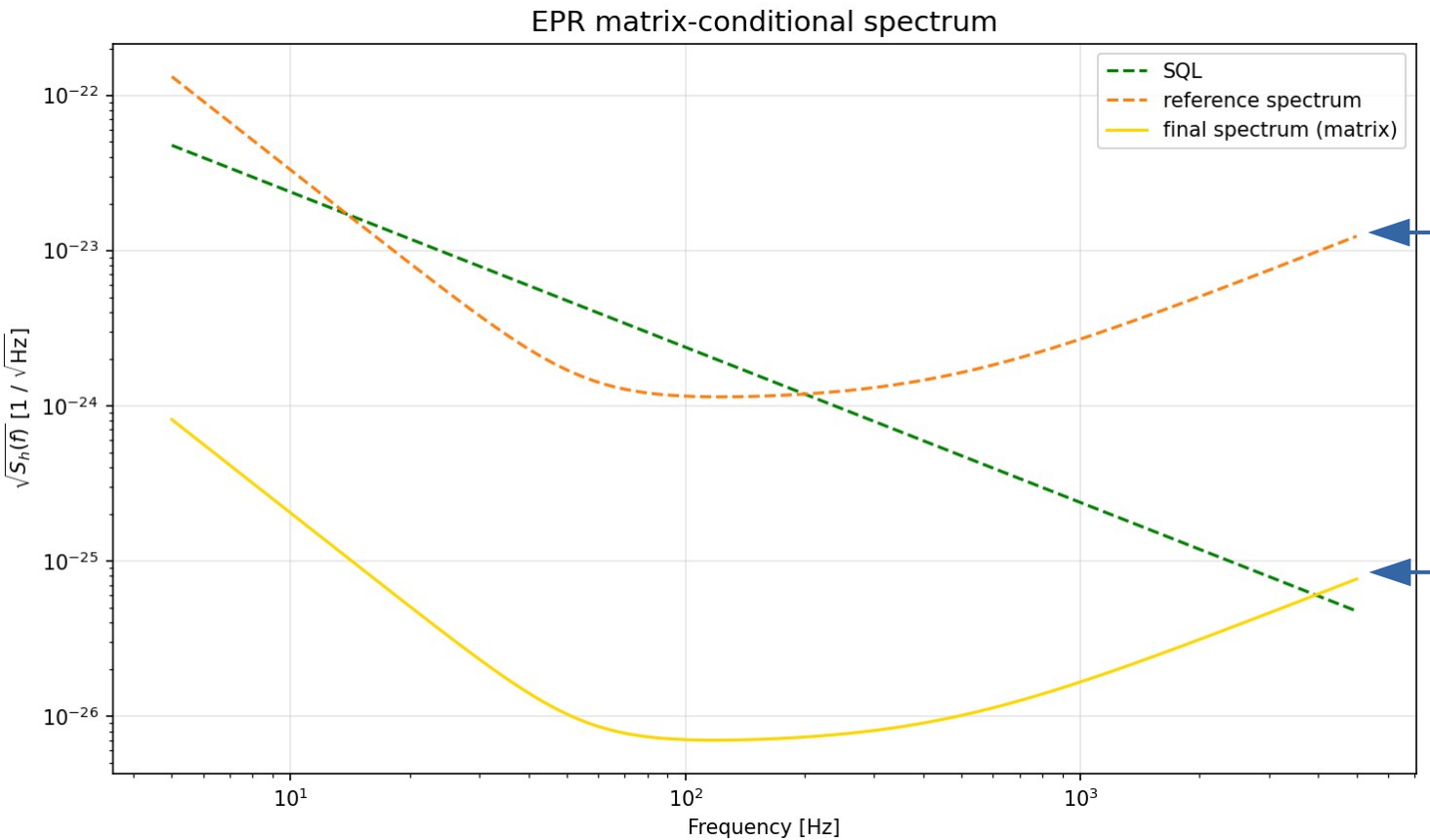
Initial point:

```
lambda0 = 1.064 *
      m = 42 *
     R1 = 0.986 *
      L = 2999.8 *
     P0 = 40 *
delta_a = 0 *
  sqz_r = 12 *
sqz_angle = 0 *
     Rp = 0.95 *
     Rs = 0.6 *
   Lrec = 11.958
 detEff = 1 *
   quad = 90 *
     R2 = 1 *
     A1 = 0 *
   A_rt = 0 *
     Ap = 0 *
delta_idler_hz = -438.28 *
```

Optimized point:

```
lambda0 = 1.064 *
      m = 42 *
     R1 = 0.986 *
      L = 2999.8 *
     P0 = 40 *
delta_a = 0.39445022 *
  sqz_r = 56.188894 *
sqz_angle = 0 *
     Rp = 0.95 *
     Rs = 0.6 *
   Lrec = 11.958
 detEff = 1 *
   quad = 87.925427 *
     R2 = 1 *
     A1 = 0 *
   A_rt = 0 *
     Ap = 0 *
delta_idler_hz = -438.28 *
```

The start and the end



initial point

optimised point

Thank you for your attention

Gradient history