
iFAST

Numerical Optimization Strategies for Optics Matching

Bernardo Abreu Figueiredo[1,2]

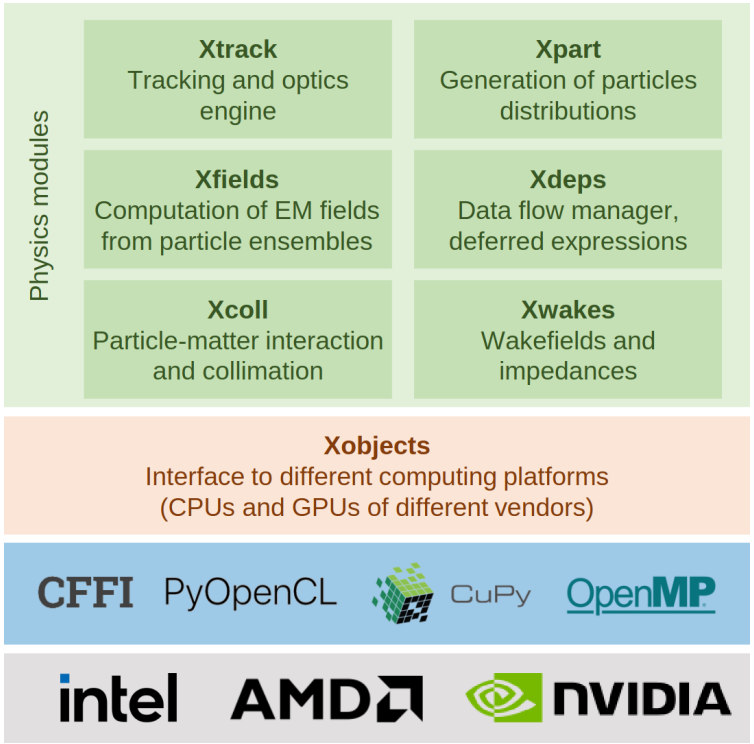
Acknowledgements: L. Deniau[1], J. Gray[1, 3], G. Iadarola[1], S. Łopaciuk[1], R. de Maria[1]

[1] CERN, Geneva, Switzerland
[2] RWTH Aachen University, Germany
[3] Nikhef National Institute for subatomic physics, Amsterdam, Netherlands

Introduction

- Control beam optics in modern accelerators
  - Twiss parameters, tunes, dispersions, chromaticities
- Closed analytical form not feasible
- Instead: Find solution numerically
- Continuous process, performance becomes limiting factor

Xsuite [1]



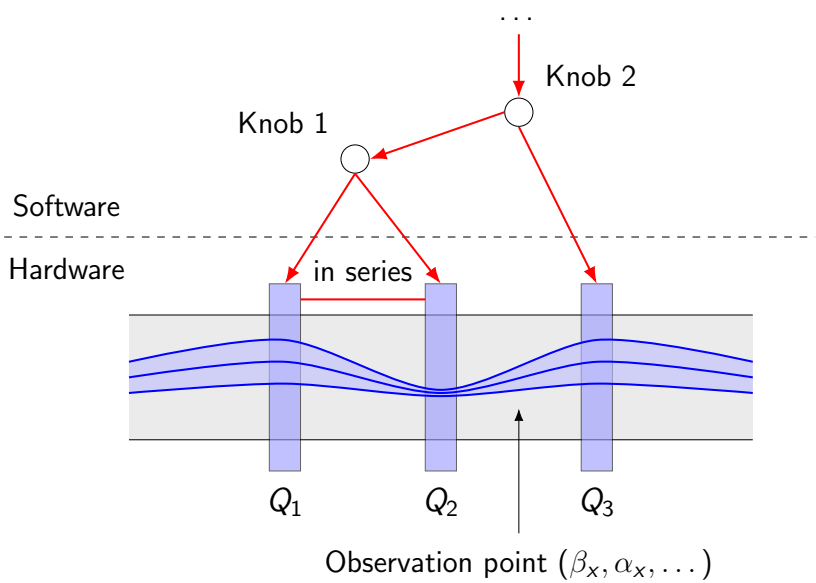
- Beam dynamics simulation toolkit
- Written in Python, some parts C++ and CUDA
- Includes matching routines, e.g. for optics
- More info: `https://xsuite.web.cern.ch`

[1] More information in S. Łopaciuk's talk about Xsuite developments

Subpackages

- Xtrack
  - Physical modeling of accelerator components
  - Tracking particle simulations
- Xdeps
  - Modeling accelerator by control system
  - Knobs that control magnet circuits
  - Software representation of dependencies
- Other packages for other use cases (Xobjects, Xpart, . . .)

Control System




```python
import xtrack as xt

env = xt.Environment()

env['knob2'] = 1.0
env['knob1'] = 'knob2*1.0'

line = env.new_line(components=[
    env.new('d0', xt.Drift, length=1.0),
    env.new('Q1', xt.Quadrupole, length=0.5, k1='knob1*1.0'),
    env.new('d1', xt.Drift, length=1.0),
    env.new('Q2', xt.Quadrupole, length=0.5, k1='knob1*(-1.0)'),
    env.new('d2', xt.Drift, length=1.0),
    env.new('Q3', xt.Quadrupole, length=0.5, k1='knob2*1.0'),
    env.new('d3', xt.Drift, length=1.0),
])
```

- Blue modeled by Xtrack, red by Xdeps
- Simple beamline definition in code

Matching Capabilities in Xsuite

Optics Matching

- Defined as continuous box-constrained optimization problems
- Varying parameter and target dimensions
  - Commonly more than 20 parameters and targets
- Minimize penalty function f w.r.t. desired targets $\vec{t}$
- Knobs are parameters of optimization problem

How to Define it?

- Match tunes for quadrupole strengths
  - 2 Variables, 2 Targets

```
opt = line.match(solve=False,
    vary=xt.VaryList(['knl_f', 'knl_d'], step=1e-3),
    targets=[xt.TargetSet(qx=.18, qy=.16)])
```

- Solve optimization: `opt.solve()`
- Get merit function: `opt.get_merit_function()`
  - Usable for custom optimizers
  - Can provide jacobian with `get_jacobian()`

Implemented Algorithms

- Supports following algorithms

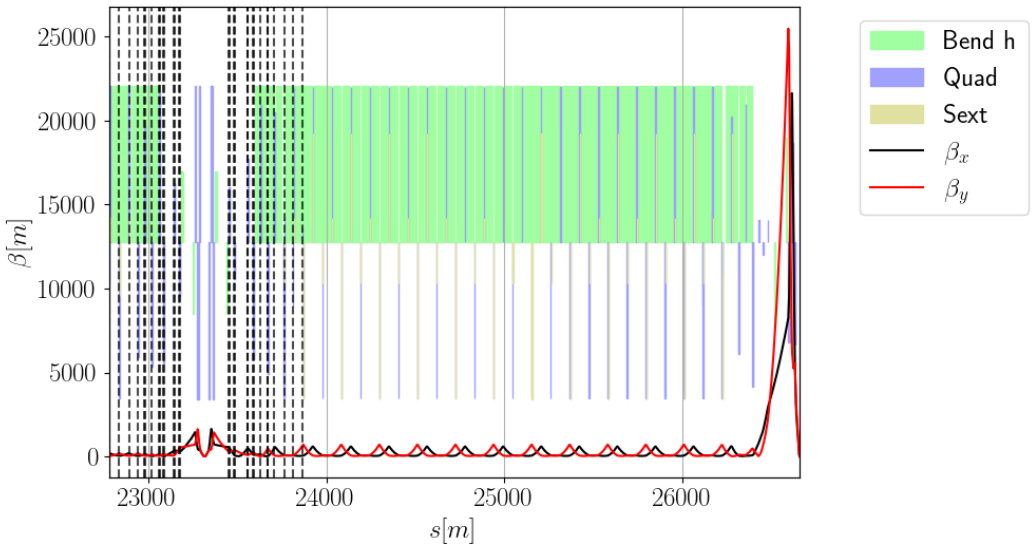
Method	Dimension	Use in Xsuite
Jacobian [2] (default)	vectorial	`opt.run_jacobian()`
Trust region reflective [3]	vectorial	`opt.run_ls_trf()`
Dogbox [3]	vectorial	`opt.run_ls_dogbox()`
L-BFGS-B [3]	scalar	`opt.run_l_bfgs_b()`
BFGS [3]	scalar	`opt.run_bfgs()`
Nelder-Mead [3]	scalar	`opt.run_nelder_mead()`
DiRECT [3]	scalar	`opt.run_direct()`

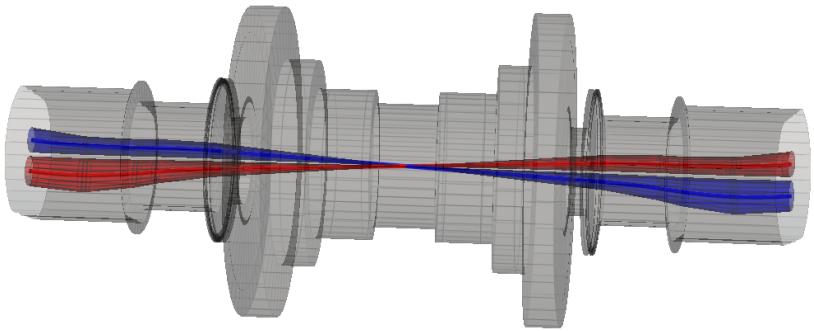
[2] Developed inhouse

[3] Implemented in SciPy

Case Study: LHC Optics Matching

- Reach $\beta_x = 0.15m$, $\beta_y = 0.1m$ at collision point
- Find good strengths k_1 for 30 quadrupoles within 4 km around IP1
- 2146 elements from starting point to IP1





How to define the Optimization Problem

```
opt = line.match(
    default_tol={None: 1e-8, 'betx': 1e-6, 'bety': 1e-6, 'alfx': 1e-6, 'alfy': 1e-6},
    start='s.ds.l8.b1', end='ip1',
    init=tw0, init_at=xt.START,
    vary=[
        xt.VaryList(['kq6.l8b1', 'kq7.l8b1', 'kq8.l8b1', 'kq9.l8b1', 'kq10.l8b1',
            'kqtl11.l8b1', 'kqt12.l8b1', 'kqt13.l8b1',
            'kq4.l8b1', 'kq5.l8b1', 'kq4.r8b1', 'kq5.r8b1',
            'kq6.r8b1', 'kq7.r8b1', 'kq8.r8b1', 'kq9.r8b1',
            'kq10.r8b1', 'kqtl11.r8b1', 'kqt12.r8b1', 'kqt13.r8b1'])],
    targets=[
        xt.TargetSet(at='ip8', tars=('betx', 'bety', 'alfx', 'alfy', 'dx', 'dpx'), value=tw0),
        xt.TargetSet(at='ip1', betx=0.15, bety=0.1, alfx=0, alfy=0, dx=0, dpx=0),
        xt.TargetRelPhaseAdvance('mux', value = tw0['mux', 'ip1.l1'] - tw0['mux', 's.ds.l8.b1'],
                                 start = 's.ds.l8.b1', end = 'ip1.l1'),
        xt.TargetRelPhaseAdvance('muy', value = tw0['muy', 'ip1.l1'] - tw0['muy', 's.ds.l8.b1'],
                                 start = 's.ds.l8.b1', end = 'ip1.l1'),
    ])
```

- Defines tolerances, start/end point, initial conditions, 20 knobs and 14 targets

Initial Status Case Study

Method	Runtime	No. Calls	No. Iterations
Jacobian	2.9s $\pm$ 0.1s	117	6

Can we do better?

Initial Status

Method	Runtime	No. Calls	No. Iterations
Jacobian	2.9s $\pm$ 0.1s	117	6

Can we do better?

- Fairly speaking, this is already quite good
- Iterative process that has to be repeated many times

Other Algorithms?

- Scalar algorithms (L-BFGS-B, BFGS, Nelder-Mead) did not converge for this case
- Vectorial algorithms converge (TRF, Dogbox) but slower

Method	Runtime	No. Calls	No. Iterations
JACOBIAN	$2.9\text{s} \pm 0.1\text{s}$	117	6
TRF	$15\text{s} \pm 1\text{s}$	463-597	20-25
Dogbox	$7.8\text{s} \pm 0.16\text{s}$	265	12-15
L-BFGS-B	no convergence	17450	1200
BFGS	no convergence	2690	600
Nelder-Mead	no convergence	≈ 5000	10000

JACOBIAN Algorithm [4]

- Main algorithm used for matching in Xsuite, based on Newton-Raphson
- Calculate Jacobian J, e.g. with finite differences
- Step $\vec{p} = J^{+}\vec{y}$
  - J^{+} is the pseudo-inverse of jacobian J obtained through SVD
  - $\vec{y}$ is the weighted error vector
- Set $\alpha = -1$, evaluate $f(\vec{x} - 2^{-\alpha}\vec{p})$ until error reduces while incrementing α

[4] R. De Maria, F. Schmidt, and P. K. Skowronski, *"Advances in matching with MAD-X."* in Proc. ICAP'06, Chamonix, Switzerland, Oct. 2006, pp. 213–215.

Challenges

- JACOBIAN algorithm is robust local optimizer, works if starting point is “close” to solution.
- Designing optics is an iterative process $\rightarrow$ smaller runtime has stronger impact
- Bottleneck: Calculating Jacobian matrix (one twiss call per variable)

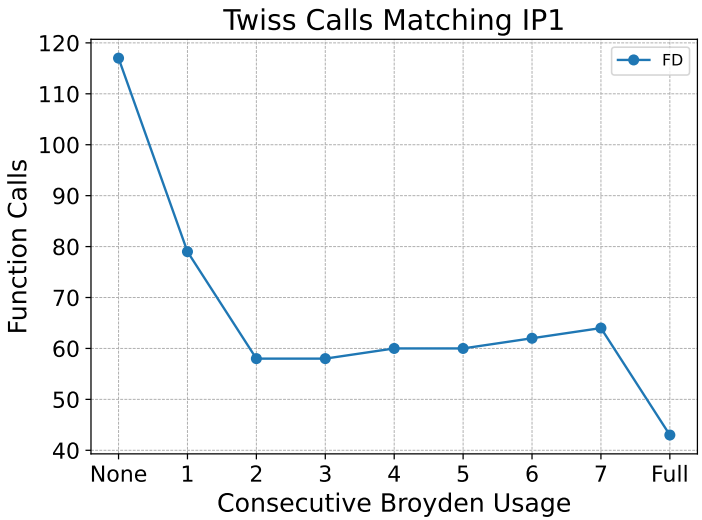
Total Twiss calls	Twiss calls for Jacobian
117	100

Focus on improving derivative computation!

Performance Improvements

Broyden's Method [5]

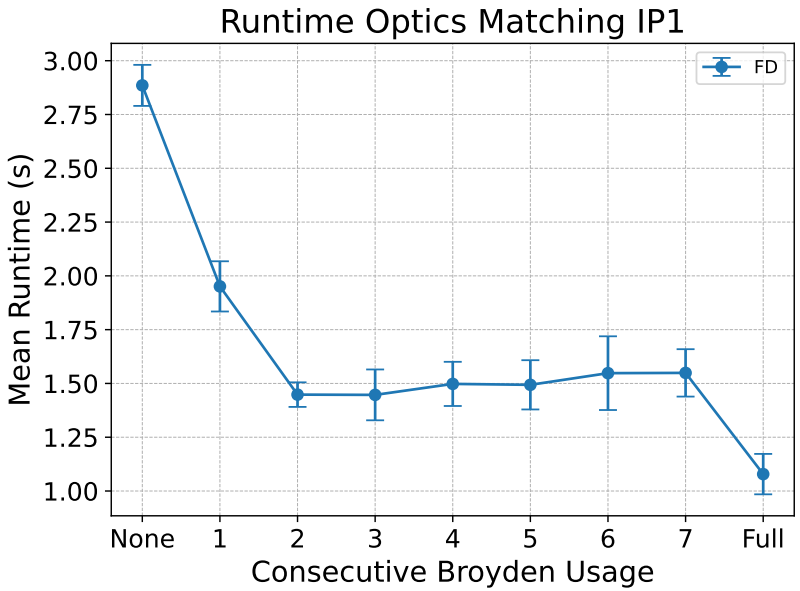
- Broyden's method to approximate Jacobian matrix
  - Compute
    $$J_{n+1} = J_n + \frac{(\Delta\vec{y}_n - J_n\Delta\vec{x}_n)\Delta\vec{x}_n^\top}{\Delta\vec{x}_n^\top\Delta\vec{x}_n}$$
  - $\Delta\vec{x}_n = \vec{x}_n - \vec{x}_{n-1}$, $\Delta\vec{y}_n = \vec{y}_n - \vec{y}_{n-1}$
  - Finite differences for derivatives

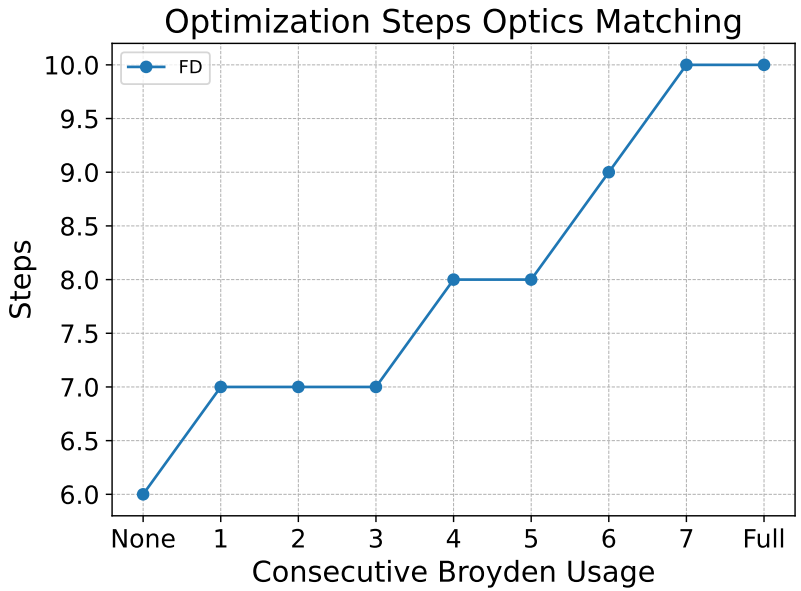


- Simple calculation $\rightarrow$ improves performance
- Works if Jacobian's are similar between iterations

[5] Broyden, C. G. (1965). "A Class of Methods for Solving Nonlinear Simultaneous Equations". Mathematics of Computation. 19 (92). American Mathematical Society: 577–593.

Broyden Results





Method	Runtime	No. Calls	No. Iterations
JACOBIAN (no Broyden)	2.9s $\pm$ 0.1s	117	6
JACOBIAN (only Broyden)	1.08s $\pm$0.05s	43	10

- Better runtime, but more optimization steps!

Automatic Differentiation

- Use automatic differentiation (AD) to calculate Jacobian instead of finite differences (FD)
- AD in beam dynamics successfully demonstrated with Cheetah [6] in tracking
- This approach uses linear optics to calculate quantities
- Reduces use of evaluation function

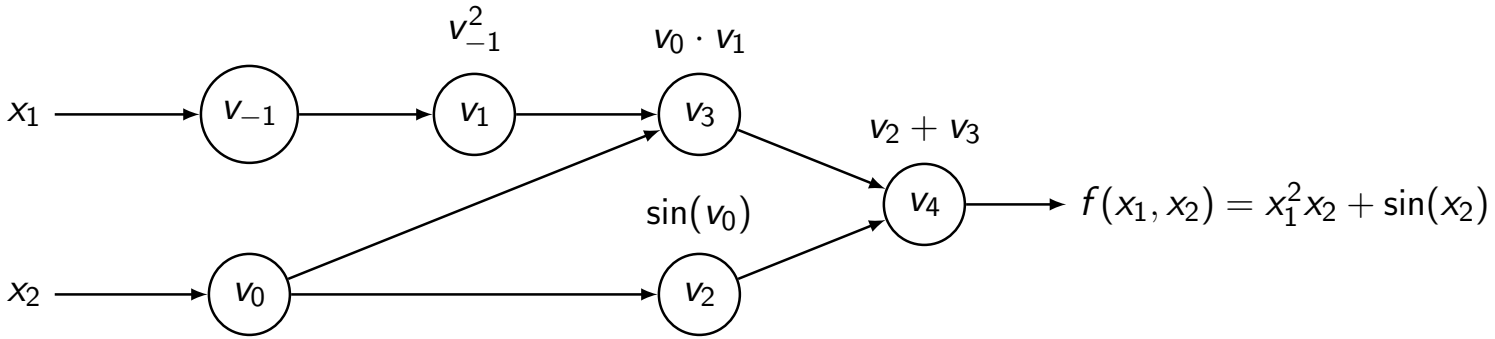
[6] O. Stein, J. Kaiser, A. Eichler - *"Accelerating Linear Beam Dynamics Simulations for Machine Learning Applications"* in Proceedings of the 13th International Particle Accelerator Conference, 2022

Automatic Differentiation

- From a transfer map $R \in \mathbb{R}^{6\times 6}$, subsequent optics can be calculated
- e.g. $\beta_{x2} = \frac{1}{\beta_{x1}}((R_{11}(k_{Q_1}, \dots)\beta_{x1} - R_{12}(k_{Q_1}, \dots)\alpha_{x1})^2 + R_{12}(k_{Q_1}, \dots)^2)$
- Chaining possible to calculate optics from start to end

JAX

- ML/AI hype boosted efforts in tools for fast differentiation
- JAX[7] is a library to calculate derivatives via AD, developed by Google
- Creates computational graph and uses chain rule



[7] Min Lin. *"Automatic Functional Differentiation in JAX"*. In: International Conference on Representation Learning. Ed. by B. Kim et al. Vol. 2024.

Automatic Differentiation

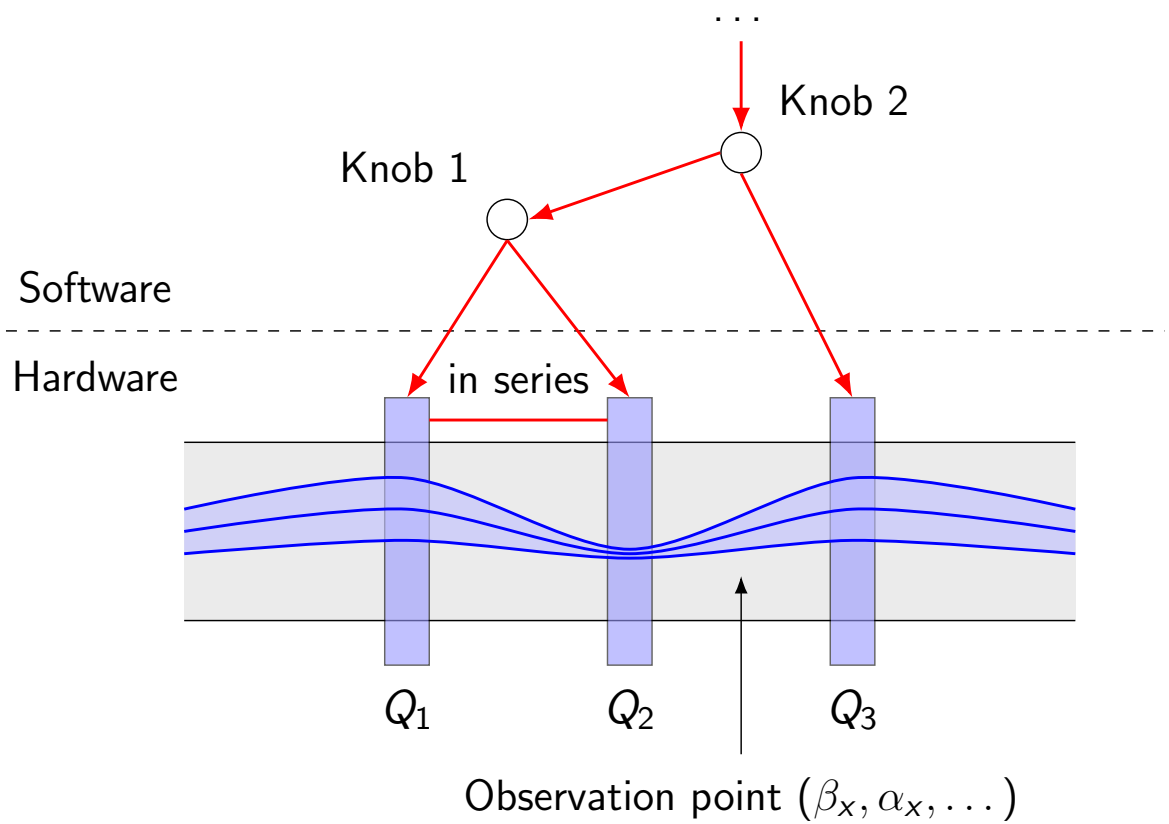
- Use JAX to create computational graph of transfer maps
- Advantages:
  - Sequence and matrices don't change
  - Can be JIT-compiled $\rightarrow$ increases performance
- Yields derivatives of optics parameters with respect to physical quadrupoles

Not the Full Story...

- AD yields e.g. $\frac{\partial \beta_x}{\partial k_1}$, but we need $\frac{\partial \beta_x}{\partial knob_1}$!
- Through chain rule, it is $\frac{\partial \beta_x}{\partial knob_n} = \frac{\partial \beta_x}{\partial k_1} \frac{\partial k_1}{\partial knob_1} \cdots \frac{\partial knob_i}{\partial knob_n}$
- To obtain these, one has to traverse the control system hierarchy

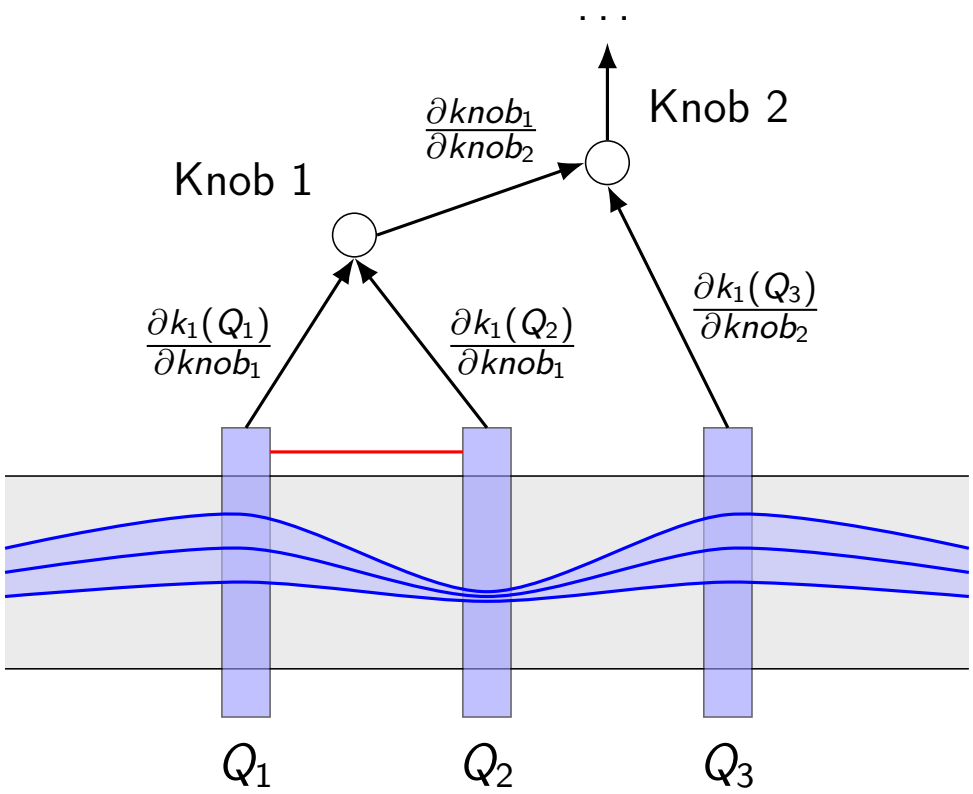
Differentiating Control System Model

$$\frac{\partial \beta_x}{\partial knob_n} = \frac{\partial \beta_x}{\partial k_1} \frac{\partial k_1}{\partial knob_1} \cdots \frac{\partial knob_i}{\partial knob_n}$$



Differentiating Control System Model

$$\frac{\partial \beta_x}{\partial knob_n} = \frac{\partial \beta_x}{\partial k_1} \frac{\partial k_1}{\partial knob_1} \cdots \frac{\partial knob_i}{\partial knob_n}$$



Differentiating Control System Model

- Traverse the graph for each knob to find affected quadrupoles
- Symbolically differentiate the expressions affiliated to the knobs
- SymPy [8] used for symbolic differentiation
- Advantages:
  - Expressions are small, so JIT would cause more overhead
  - Only needed once (except if lattice/knobs change)

[8] Aaron Meurer et al. *"SymPy: symbolic computing in Python"*. In: PeerJ Computer Science 3. 2017

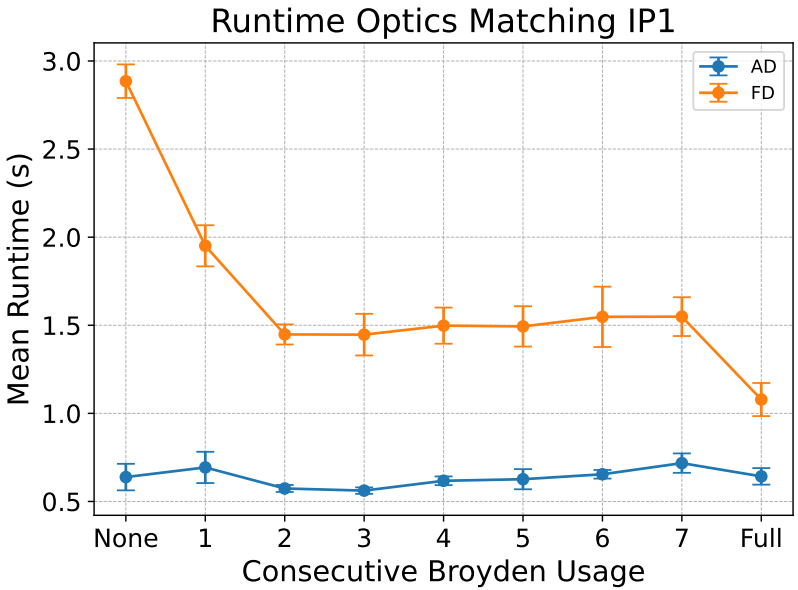
Results

Benchmark Setup

- Benchmarked on IP1 optics matching example
- Executed on Ubuntu machine with Intel(R) Core^{TM} i7-14700T
- Two configurations
  - Xsuite with Finite Differences (default)
  - Xsuite with Automatic Differentiation
- Analyze for different frequency of Broyden use
- 10 runs

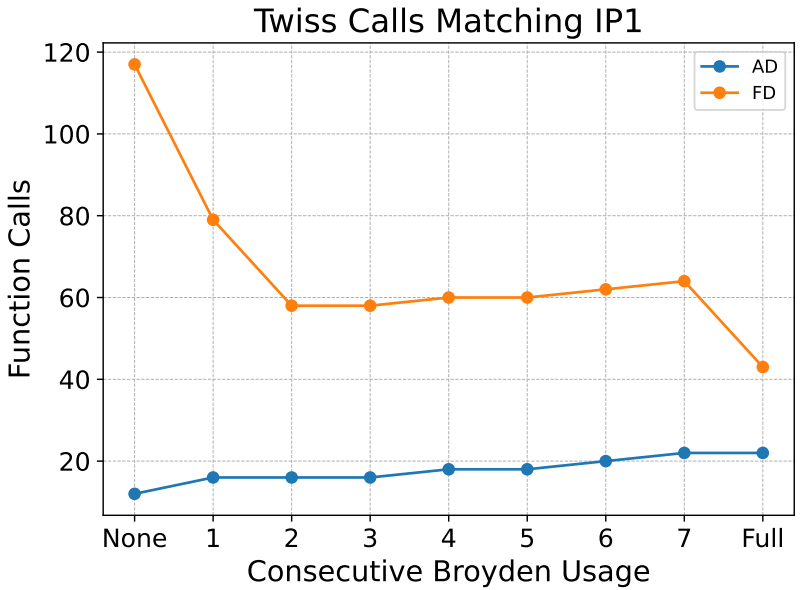
Results Runtime

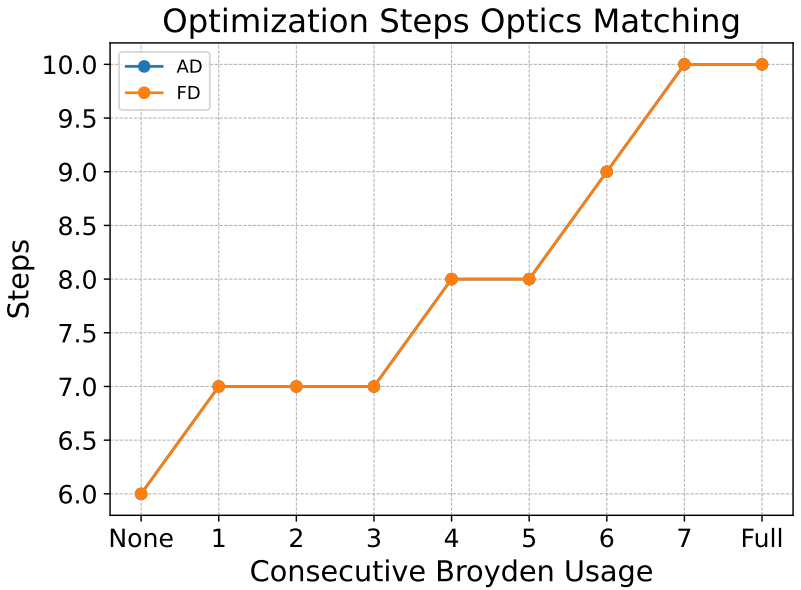
- AD substantially faster than FD approach
- Broyden increases performance for FD, not notable for AD
  - Increases number of iterations, thus also number of calls



Method	Runtime	No. Calls	No. Iterations
FD (no Broyden)	2.9s ± 0.1s	117	6
FD (only Broyden)	1.08s ± 0.05s	43	10
AD (no Broyden)	0.63s ± 0.08s	12	6
AD (only Broyden)	0.64s ± 0.05s	22	10

Broyden vs. Optimization Steps





Method	Runtime	No. Calls	No. Iterations
FD (no Broyden)	2.9s ± 0.1s	117	6
FD (only Broyden)	1.08s ± 0.05s	43	10
AD (no Broyden)	0.63s ± 0.08s	12	6
AD (only Broyden)	0.64s ± 0.05s	22	10

Conclusion

Conclusion

- Two main ways explored to improve optimization performance
- Fast way: Broyden's method
  - Simple implementation and integration
  - Flexible, can be used for different problem definitions
- Limited, but faster way: Automatic Differentiation
  - Very fast calculation
  - Limited to optics calculation, for now

Outlook

- Limitations of AD
  - So far only works for optical functions
  - Only for quadrupole strengths as variables
  - Matching Chromaticities requires further effort
- Full nonlinear dynamics with AD is developed in MAD-NG [9], developed inhouse
  - Improve its integration with Xsuite, especially for fast Jacobian computation
  - Compare performance to JAX

[9] L. Deniau, *"MAD-NG, a standalone multiplatform tool for linear and non-linear optics design and optimisation"*, 2025, arXiv:2412.16006


iFAST