Efficient Derivative Computation for Accelerator Beam Optics Calculations

Bernardo Abreu Figueiredo

28th EuroAD Workshop

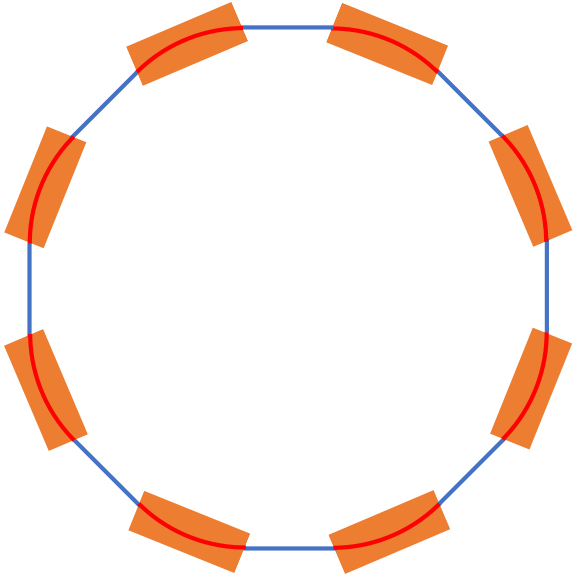
Supervised by: G. Iadarola, Sz. Łopaciuk, U. Naumann

Acknowledgements: L. Deniau, J. Gray, R. De Maria

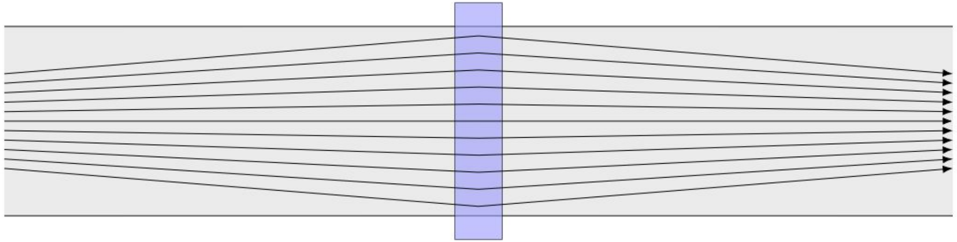
Challenges in Beam Optics

- Control particle beam
  - Ensure stable orbit
  - Focus beam in collision points
  - Minimize beam loss
- Goal: Find good optics (governed by magnets)

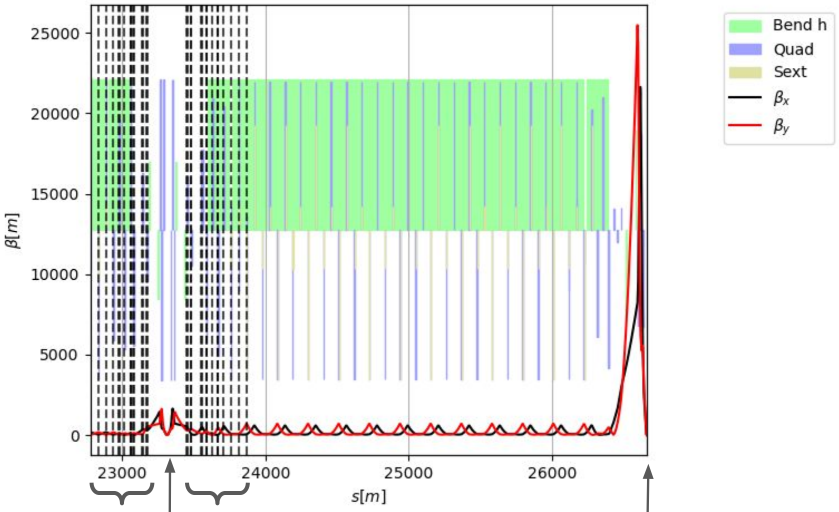
Bending beam with Dipole magnets



Focusing beam with Quadrupole magnets



Challenges in Beam Optics

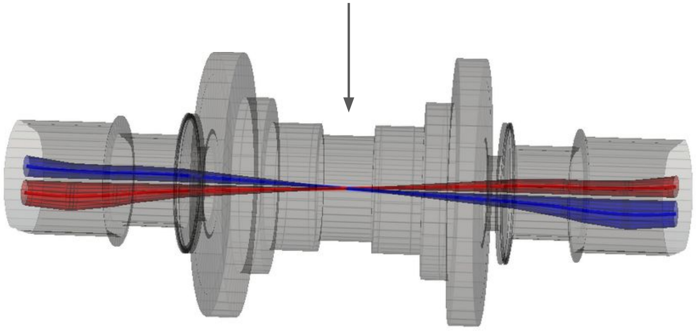


Quadrupole Positions

Collision Point (IP8)

Collision point (IP1)

Collision point (IP1)



Courtesy of Sz. Łopaciuk

Goal: Find quadrupole strengths to get compact beam (small β)

Outline

- Why AD in Accelerator Physics?
- Minimal Introduction to Beam Optics
- Baseline: Finite Differences
- AD with Linear Optics
- TPSA with Nonlinear Optics
- Conclusion

Why AD in Accelerator Physics?

- Beam optics matching involves solving nonlinear continuous optimisation problems for magnet strengths.
- Expensive evaluation function: Calculates several beam optics quantities
- Bottleneck: Computing Jacobian of the target error function.
  - Traditionally done with finite differences → expensive, noisy, scales with number of variables
- Optics Design is an iterative process with many configurations, for many parts of the machine
- **Goal**: Can AD methods accelerate the Jacobian calculation?

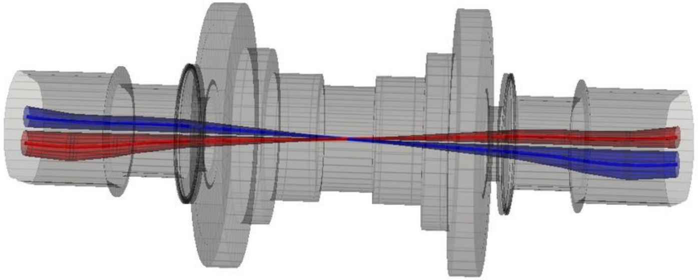
Minimal Introduction to Beam Optics

- Particle motion described via 6D phase-space vector
- Each element (dipole, quadrupole) applies map to this vector
  - Linear optics: 6x6 Matrix
  - Nonlinear optics: Polynomials with higher-order terms
- Key targets for optimization
  - Twiss functions (β, α, γ) → Describe size and shape of particle beam
  - Phase advance (μ) → How far beam oscillation progresses between points
  - Dispersion (D) → Describes how orbit shifts with momentum deviation (δ)
  - Higher-order chromatic functions (dβ/dδ, dμ/dδ) → Sensitivity of optics to energy spread
- **Optics Matching: Find magnet strengths such that optics matches target desired targets**

$$\vec{z} = \begin{pmatrix} x \\ p_x \\ y \\ p_y \\ \zeta \\ \delta \end{pmatrix}$$

Case Study: Small Beam at Collision Point in LHC

- We're interested in making as many collisions as possible in collision points
- Small and dense beam is needed
- We want:
  - Small β functions (beam size)
  - α = 0 (α is derivative of β) → minimal β
  - No dispersion
  - Keep phase advance as before

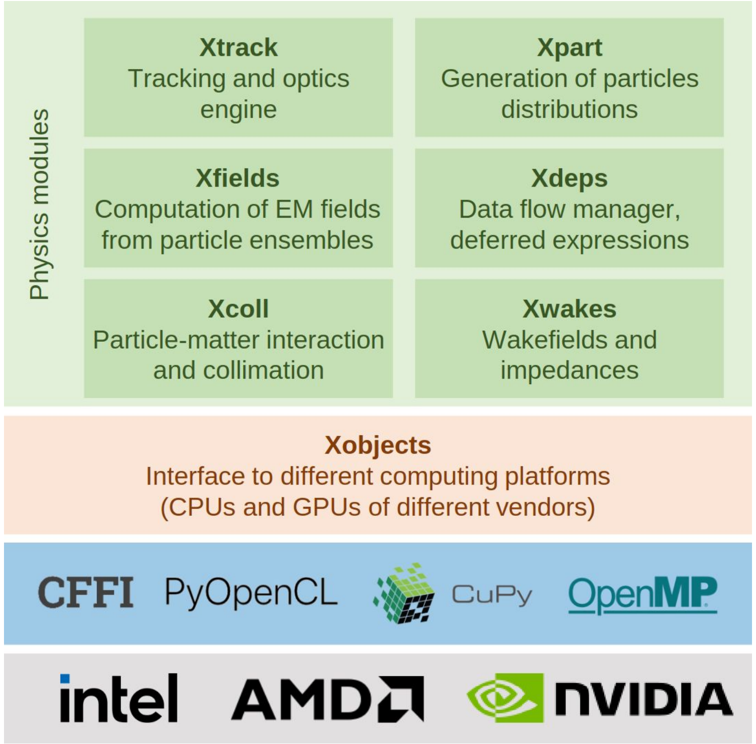


Courtesy of Sz. Łopaciuk

- We already have an accelerator lattice with defined magnets → need to change magnet strengths to get lower results
- Optimization problem with 14 targets and 20 variables (quadrupole magnets)

Xsuite

- Beam dynamics simulation toolkit
- Written in Python, some parts C++ and CUDA
- Implements (among others)
  - Matching routines, e.g. for optics
  - Particle tracking
  - Optics and orbit calculation
- More info: https://xsuite.web.cern.ch



Define the Optimization Problem in Xsuite

- 20 Variables (Quadrupole Knobs)
- 14 Targets (Optical Parameters)

Start and End

Variables

Targets

```
opt = line.match(
    solve=False,
    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, weight=1),
        xt.TargetSet(at='ip1', betx=0.15, bety=0.1, alfx=0, alfy=0, dx=0, dpx=0, weight=1),
        xt.TargetRelPhaseAdvance('mux', value = tw0['mux', 'ip1.l1'] - tw0['mux', 's.ds.l8.b1'],
                                start='s.ds.l8.b1', end='ip1.l1', weight=1),
        xt.TargetRelPhaseAdvance('muy', value = tw0['muy', 'ip1.l1'] - tw0['muy', 's.ds.l8.b1'],
                                start='s.ds.l8.b1', end='ip1.l1', weight=1),
    ]
)
```

Optimization: JACOBIAN Algorithm

- For optimization, we use the JACOBIAN [1] algorithm
- Based on Newton-Raphson, with a bisect step
- Calculate Jacobian J, e.g. with finite differences
- Calculate step $p = J^{+}y$
    - J^{+} is the pseudo-inverse of J, obtained from SVD
    - y is the weighted error vector
- Set $\alpha = -1$, evaluate $f(x - 2^{-\alpha}p)$ until error reduces while incrementing α

[1] 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.

Baseline: Finite Differences

- Benchmarks performed on this IP1/IP8 matching problem
- Executed on Ubuntu machine with Intel(R) Core^{TM} i7-14700T
- Runtimes averaged over 10 runs

Method	Runtime	Eval Calls	Optimization Steps
JACOBIAN with Finite Differences	2.47s ± 0.06s	117	6

- Can we do better?

Baseline: Finite Differences

Method	Runtime	Eval Calls	Optimization Steps
JACOBIAN with Finite Differences	2.47s ± 0.06s	117	6

- Can we do better?

Total Eval Calls	Eval Calls for Jacobian
117	100

- Jacobian calculation with finite differences is a performance bottleneck, more than 80% of evaluation function calls needed for Jacobian matrix
- Focus will be on ways to optimise Jacobian calculation

First Approach: AD with Linear Optics

- We associate each element with a 6x6 transfer matrix R
- Nonlinear elements are approximated as drifts → higher-order effects discarded
- Computation relies on calculating optical functions from these maps

$$R = \begin{pmatrix} c_x & s_x & 0 & 0 & 0 & 0 \\ -k_x^2 s_x & c_x & 0 & 0 & 0 & 0 \\ 0 & 0 & c_y & s_y & 0 & 0 \\ 0 & 0 & -k_y^2 s_y & c_y & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{L}{\beta^2\gamma^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Quadrupole Map

$$R = \begin{pmatrix} c_x & s_x & 0 & 0 & 0 & \frac{h}{\beta} d_x \\ -k_x^2 s_x & c_x & 0 & 0 & 0 & \frac{h}{\beta} s_x \\ 0 & 0 & c_y & s_y & 0 & 0 \\ 0 & 0 & -k_y^2 s_y & c_y & 0 & 0 \\ -\frac{h}{\beta} s_x & -\frac{h}{\beta} d_x & 0 & 0 & 1 & \frac{L}{\beta^2\gamma^2} - \frac{h^2}{\beta^2} J_1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

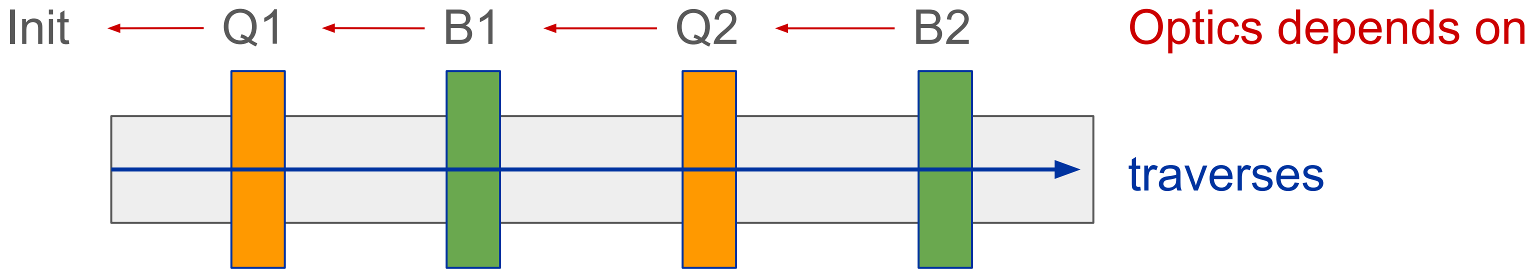
Dipole Map

$$R = \begin{pmatrix} 1 & L & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & L & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{L}{\beta^2\gamma^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Drift Map

First Approach: AD with Linear Optics

- From the transfer map, we can directly calculate new optics from previous, e.g. $\beta_{x2} = \frac{1}{\beta_{x1}}((R_{11}(k_{Q_1},\ldots)\beta_{x1} - R_{12}(k_{Q_1},\ldots)\alpha_{x1})^2 + R_{12}(k_{Q_1},\ldots)^2)$
- Instead of multiplying the transfer matrices, we propagate the optics
- Sequential dependencies



- Optics at B2 depend on Optics at Q2, Optics at Q2 depend on B1, …

Performance Implications

- JIT compilation possible as lattice does not change
  - Better performance as called frequently
- Use JAX to calculate forward-mode gradients
  - Scan operation through elements while carrying over optics/derivatives
- This implementation more than four times faster for this case

Method	Runtime	Eval Calls	Optimization Steps	Speedup
Finite Differences	2.47s ± 0.06s	117	6	1
AD (JAX) Linear Optics	0.58s ± 0.04s	12	6	4.26

- However: Works only for linear optics, no higher-order effects captured

Second Approach: AD/TPSA

- Calculating the optics from the transfer map is a linear approximation
- More precise to use derivatives from coordinates
- Instead of scalars, polynomials are propagated, yielding higher-order derivatives
- In this domain referred to as Truncated Power Series Algebra (TPSA) [2]
  - Similar to forward-mode AD, but uses polynomials instead of scalars
  - Operator overloading on polynomial level, creating higher-order terms
- Method implemented in MAD-NG, a beam optics code developed inhouse
- More details in L. Deniau's talk!

[2] L. Deniau, C. Tomoiagă, "Generalised Truncated Power Series Algebra For Fast Particle Accelerator Transport Maps", IPAC'15, 2015.

Initialization TPSA

- For each coordinate (x, p_x, …), there is one polynomial
- Coefficients are initialized based on W matrix
- 0^{th} order (constant) is set to current coordinates
- 1^{st} order set based on initial conditions (W-matrix):
  - e.g. TPSA for x follows first row of W:

$$\mathcal{T}_x = \underbrace{\underbrace{\sqrt{\beta_x}}_{W_{11}} \cdot x + \underbrace{D_x}_{W_{16}} \cdot \delta}_{\text{Order 1}} + \underbrace{x_0}_{\text{Order 0}}$$

Monomials

Coefficients

Polynomial

$$W = \begin{pmatrix} \sqrt{\beta_x} & 0 & 0 & 0 & 0 & D_x \\ -\frac{\alpha_x}{\sqrt{\beta_x}} & \frac{1}{\sqrt{\beta_x}} & 0 & 0 & 0 & D_{px} \\ 0 & 0 & \sqrt{\beta_y} & 0 & 0 & D_y \\ 0 & 0 & -\frac{\alpha_y}{\sqrt{\beta_y}} & \frac{1}{\sqrt{\beta_y}} & 0 & D_{py} \\ 0 & 0 & 0 & 0 & \sqrt{\beta_s} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{\beta_s}} \end{pmatrix}$$

Second Approach: AD/TPSA (MAD-NG)

- MAD-NG [3] is a beam optics code, focused on calculating optics from differential algebra
- Optimized for runtime
  - TPSA implementation in C/C++
  - JIT-compilation with LuaJIT
- Efforts have been made to integrate this into the Xsuite matching routine to profit from faster Jacobian calculations

[3] Deniau, Laurent, "MAD-NG, a standalone multiplatform tool for linear and non-linear optics design and optimisation.", 2024, 10.48550/arXiv.2412.16006.

From TPSA to Targets

- Quantities can be calculated directly from the W matrix
- Through applying equations of motion to polynomials, quantities can be obtained analogously directly from the coefficients

$$\beta_x = W_{11}^2 + W_{12}^2$$
$$\beta_y = W_{33}^2 + W_{34}^2$$
$$\alpha_x = -W_{11}W_{21} - W_{12}W_{22}$$
$$\alpha_y = -W_{33}W_{43} - W_{34}W_{44}$$
$$D_x = \left(W_{16} - \frac{W_{15}W_{56}}{W_{55}}\right)\left(W_{66} - \frac{W_{65}W_{56}}{W_{55}}\right)^{-1}$$
$$D_{px} = \left(W_{26} - \frac{W_{25}W_{56}}{W_{55}}\right)\left(W_{66} - \frac{W_{65}W_{56}}{W_{55}}\right)^{-1}$$
$$D_y = \left(W_{36} - \frac{W_{35}W_{56}}{W_{55}}\right)\left(W_{66} - \frac{W_{65}W_{56}}{W_{55}}\right)^{-1}$$
$$D_{py} = \left(W_{46} - \frac{W_{45}W_{56}}{W_{55}}\right)\left(W_{66} - \frac{W_{65}W_{56}}{W_{55}}\right)^{-1}$$

$$\beta_x = [\mathcal{T}_x]_x + [\mathcal{T}_x]_{p_x}$$
$$\alpha_x = -[\mathcal{T}_x]_x \cdot [\mathcal{T}_{p_x}]_x - [\mathcal{T}_x]_{p_x} \cdot [\mathcal{T}_{p_x}]_{p_x}$$

Coefficient of x with monomial x

Coefficient of p_x with monomial x

From TPSA to Derivatives

- TPSA at locations contain also sensitivities towards parameters
- With that, we can calculate the derivatives to build the Jacobian
- Differentiating the equation for β_x from the previous slide yields:

$$\frac{\partial \beta_x}{\partial K_j} = 2\Big(W_{11}\underbrace{\frac{\partial W_{11}}{\partial K_j}} + W_{12}\underbrace{\frac{\partial W_{12}}{\partial K_j}}\Big) = 2\big([\mathcal{T}_x]_x \cdot \underbrace{[\mathcal{T}_x]_{x\cdot K_j}} + [\mathcal{T}_x]_{p_x} \cdot \underbrace{[\mathcal{T}_x]_{p_x\cdot K_j}}\big)$$

Sensitivity towards knob K_j — Coefficients for knob K_j

Benchmark

- TPSA reaches speedup of almost seven
- Same number of calls and optimization steps regarding linear optics approach

Method	Runtime	Eval Calls	Optimization Steps	Speedup
Finite Differences	2.47s ± 0.06s	117	6	1
AD (JAX) Linear Optics	0.58s ± 0.04s	12	6	4.26
TPSA	0.36 ± 0.01s	12	6	6.86

Match Optics in Xsuite with TPSA: User View

- Simple interface to MAD-NG implemented
- Set flag `use_tpsa`, that's it!

TPSA flag

```
opt = line.match(
    solve=False,
    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,
    use_tpsa=True,
    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, weight=1),
        xt.TargetSet(at='ip1', betx=0.15, bety=0.1, alfx=0, alfy=0, dx=0, dpx=0, weight=1),
        xt.TargetRelPhaseAdvance('mux', value = tw0['mux', 'ip1.l1'] - tw0['mux', 's.ds.l8.b1'],
                                 start='s.ds.l8.b1', end='ip1.l1', weight=1),
        xt.TargetRelPhaseAdvance('muy', value = tw0['muy', 'ip1.l1'] - tw0['muy', 's.ds.l8.b1'],
                                 start='s.ds.l8.b1', end='ip1.l1', weight=1),
    ],
)
```

Conclusion

- Studied strategies to use AD for optics matching
  - Very efficient in calculation, enabling speedups, making use of JIT-compilation and optimized code (e.g. TPSA library in C/C++)
  - With TPSA, higher-order parameters can be matched, simple interface for user
  - Overhead and complexity through interfacing between Xsuite and MAD-NG, code generation via string building
- Next Steps
  - Apply method on problems for LHC and FCC
  - Support more (higher-order) targets