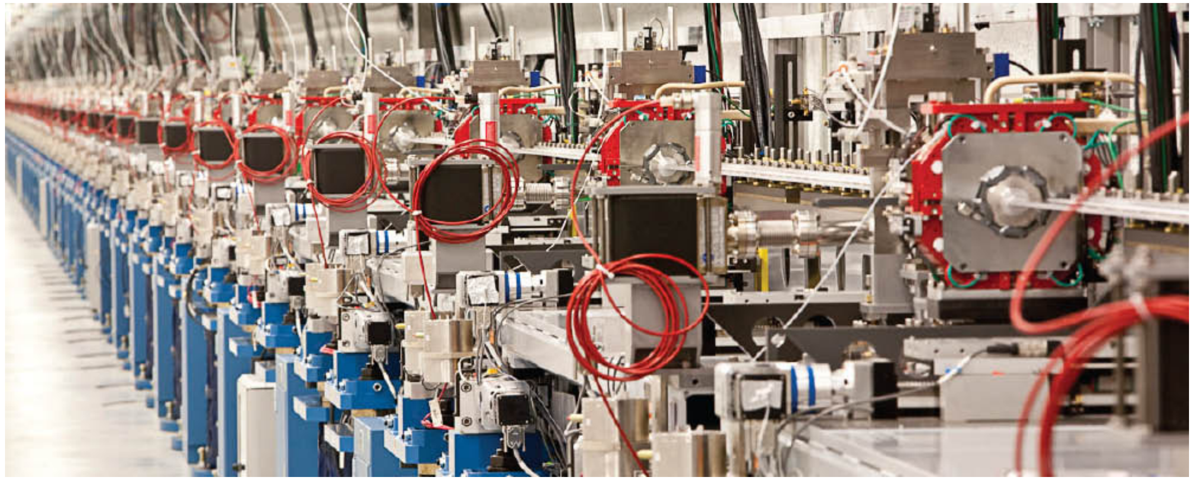
Towards End-to-End Differentiable Modeling of Particle Accelerators

J. P. Gonzalez-Aguilera*, Y.-K. Kim
Department of Physics and Enrico Fermi Institute,
University of Chicago, Chicago, IL

R. Roussel, A. Edelen, C. Mayes
SLAC National Accelerator Laboratory, Menlo Park, CA

* jpga@uchicago.edu

Motivation



https://lcls.slac.stanford.edu/

- Many parameters
- Nonlinear beam response
- Limited beam diagnostics
- Must meet beam quality objectives

Challenges:

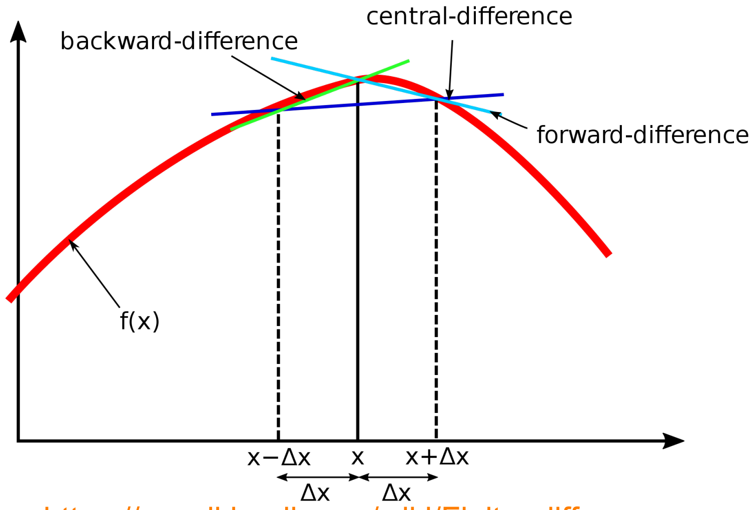
- Design
- Control
- Model calibration

} Optimization

- We need fast and accurate gradient information for high-dimensional gradient-based optimization.

Usual way to calculate gradients

- Numerical differentiation / finite differences
  - Numerical errors
  - Unstable in many situations
  - Computationally expensive
  - Scales badly with dimensions



https://en.wikipedia.org/wiki/Finite_difference

- Symbolic / analytical differentiation
  - Complicated mathematical expressions
  - Infeasible in complicated computer functions / routines
  - Scales badly with dimensions

SymPy

Wolfram Mathematica

Automatic Differentiation (AD)

- Computers execute primitive operations/functions

$$(+, -, \times, \div, \sin, \cos, \exp, \log, \ldots)$$

- Routines are composed sequences of these primitive operations

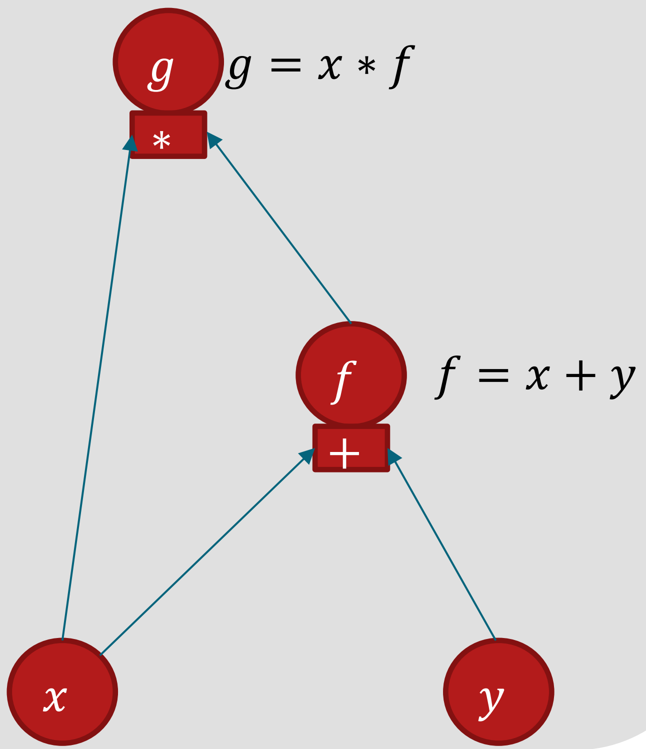
- AD uses the derivatives of these primitive operations and the **chain rule** to evaluate the derivative of a computer function w.r.t. any input

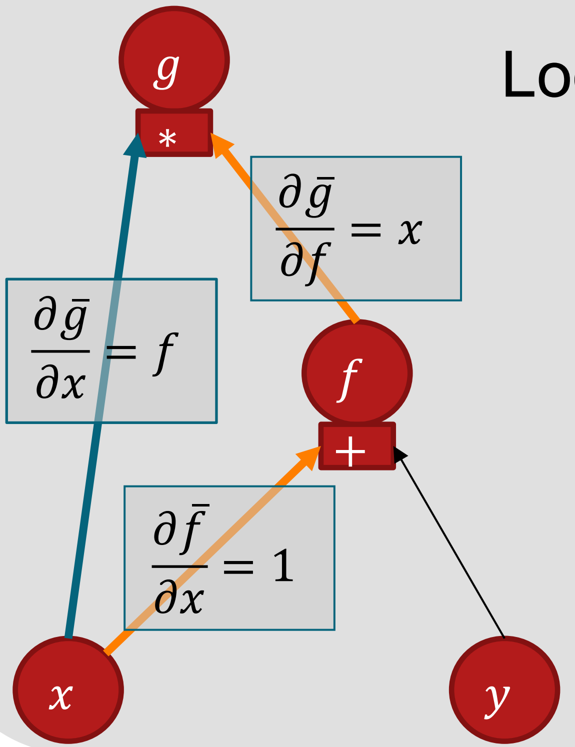
- Results in
  - fast derivatives (linear in the cost of computing the value)
  - numerically stable
  - working precision

Automatic Differentiation Example

$$f(x,y) = x + y,$$
$$g(x, f(x,y)) = x * f(x,y),$$
$$x = 3,$$
$$y = 2.$$

Graph:





Evaluate $\partial g / \partial x$.

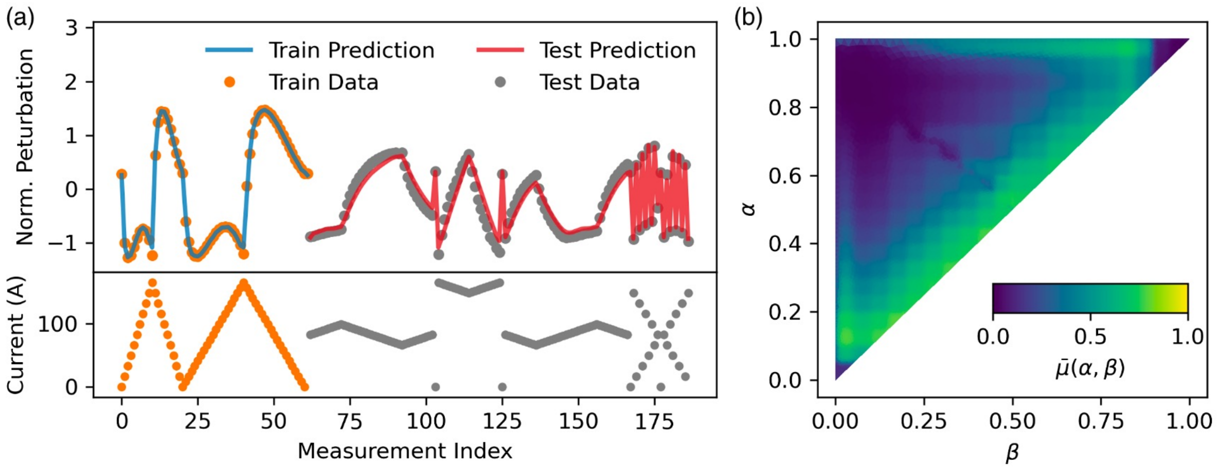
Look for paths from g to x and use chain rule:

$$\frac{\partial g}{\partial x} = \frac{\partial \bar{g}}{\partial x} + \frac{\partial \bar{g}}{\partial f} * \frac{\partial \bar{f}}{\partial x}$$
$$= f + x * 1$$
$$= x + y + x$$
$$= 2x + y = 8.$$

AD in Accelerator Modeling

- "Differential Algebraic" beam dynamics (1988, M. Berz, doi.org/10.2172/6876262)
  - Uses AD to calculate derivatives of phase-space coordinates
  - Enables computation of **arbitrary order Taylor maps**
  - Can add beamline parameters as "knobs"

- Modeling of hysteresis in accelerator magnets
  - AD enables gradient based optimization of ~ **7K mesh points**

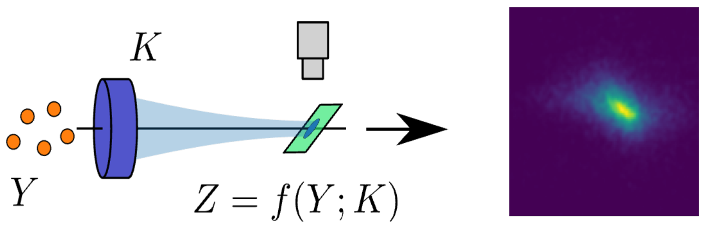


Roussel *et al.* PRL 2022

Differentiable Accelerator Modeling

But we want **fully differentiable** accelerator modeling:

- Use AD to evaluate derivatives of **any output** w.r.t. **any input**
- Enabling **high-dimensional gradient-based optimization** of any output



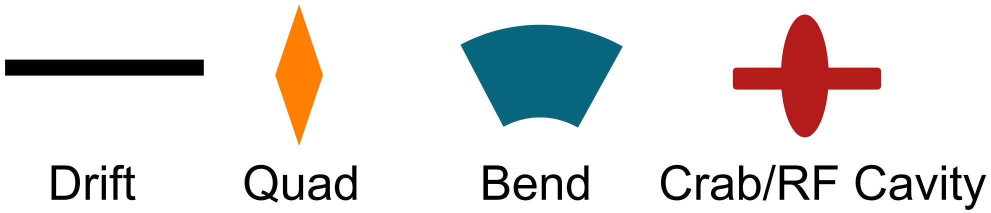
$$\frac{\partial \sigma_x}{\partial K}, \frac{\partial \sigma_x}{\partial Y}, \dots$$

$$Q^{(i,j)} = \mathrm{KDE}(Z)$$

$$\frac{\partial f(Q^{(i,j)})}{\partial K}, \frac{\partial f(Q^{(i,j)})}{\partial Y}, \dots$$

How:

- Implementation of Bmad* standard tracking routines in Python in a **library agnostic way**
- Can be used with PyTorch, Numba, etc.
  - Automatic Differentiation
  - JIT compilation
  - GPU support
  - ML Modules: NN, Optimization, …
- Current elements:



* classe.cornell.edu/bmad/

Library Agnostic Tracking

```python
def make_track_a_crab_cavity(lib):
    """Makes track_a_crab_cavity given the library lib."""
    sqrt = lib.sqrt
    sin = lib.sin
    cos = lib.cos
    track_this_drift = make_track_a_drift(lib)
    offset_particle_entrance = make_f(lib, 'offset_particle_entrance')
    offset_particle_exit = make_f(lib, 'offset_particle_exit')
    particle_rf_time = make_f(lib, 'particle_rf_time')

    def track_a_crab_cavity(p_in, cav):
        """Tracks an incomming Particle p_in through crab cavity and
        returns the ourgoing particle.
        See Bmad manual section 4.9
        """
        s = p_in.s
        p0c = p_in.p0c
        mc2 = p_in.mc2

        l = cav.L

        x_off = cav.X_OFFSET
        y_off = cav.Y_OFFSET
        tilt = cav.TILT

        par = offset_particle_entrance(x_off, y_off, tilt, p_in)

        par = track_this_drift(par, Drift(l/2))
        x, px, y, py, z, pz = par.x, par.px, par.y, par.py, par.z, par.pz

        voltage = cav.VOLTAGE / p0c
        k_rf = 2 * pi * cav.RF_FREQUENCY / c_light
        phase = 2 * pi * (cav.PHI0 - (particle_rf_time(par)*cav.RF_FREQUENCY))
```

.
.
.

Elementary functions

Auxiliary functions

Elementary operations

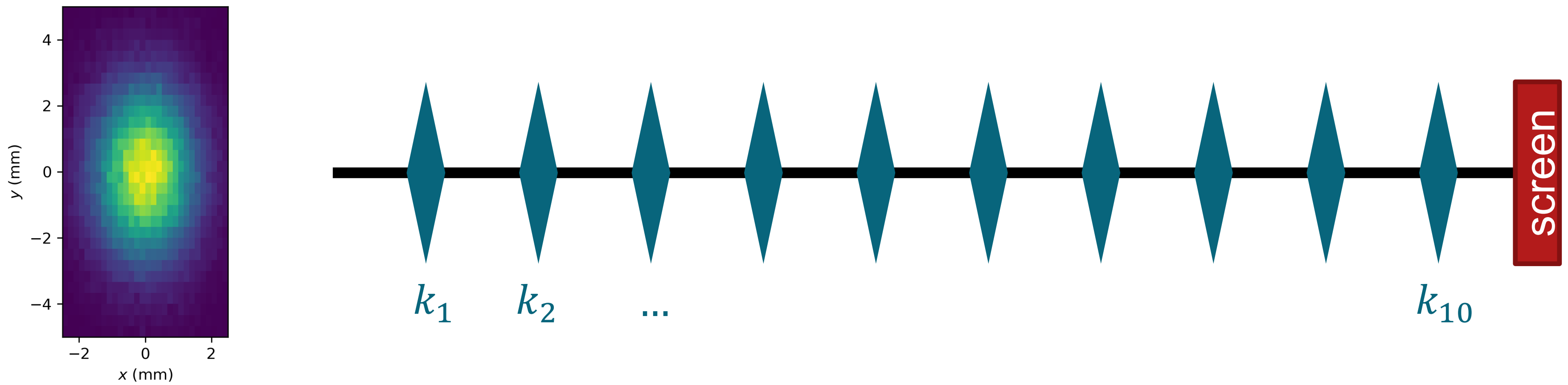
PyTorch, NumPy, Numba, NVIDIA CUDA, CuPy

PyTorch autograd example:

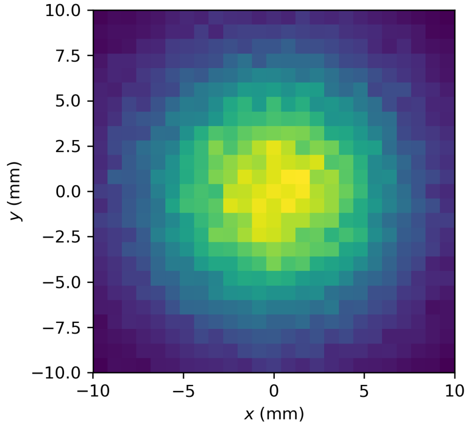
```python
track_a_quadrupole_torch = track.make_track_a_quadrupole(torch)
f_quadrupole = lambda x: track_a_quadrupole_torch(track.Particle(*x,ts, tp0c, tmc2), q1)[:6]
from torch.autograd.functional import jacobian
J = jacobian(f_quadrupole, tvec1)
mat_py = torch.vstack(J)
mat_py
```

```
tensor([[ 9.503167431875498e-01,  9.853541097581728e-02,  0.000000000000000e+00,
          0.000000000000000e+00,  0.000000000000000e+00, -1.924858550317723e-04],
        [-9.833834015386563e-01,  9.503167431875498e-01, -0.000000000000000e+00,
          0.000000000000000e+00,  0.000000000000000e+00,  1.149663908944082e-04],
        [ 0.000000000000000e+00,  0.000000000000000e+00,  1.050519938506054e+00,
          1.018821577510623e-01,  0.000000000000000e+00,  2.569093937337833e-04],
        [-0.000000000000000e+00,  0.000000000000000e+00,  1.016783934355602e+00,
          1.050519938506054e+00,  0.000000000000000e+00,  1.017485822657404e-04],
        [ 8.003290869842023e-05, -1.942507914386516e-04,  1.543324297486644e-04,
          2.595220753974964e-04,  1.000000000000000e+00,  1.756709202142694e-05],
        [ 0.000000000000000e+00,  0.000000000000000e+00,  0.000000000000000e+00,
          0.000000000000000e+00,  0.000000000000000e+00,  1.000000000000000e+00]],
       dtype=torch.float64)
```

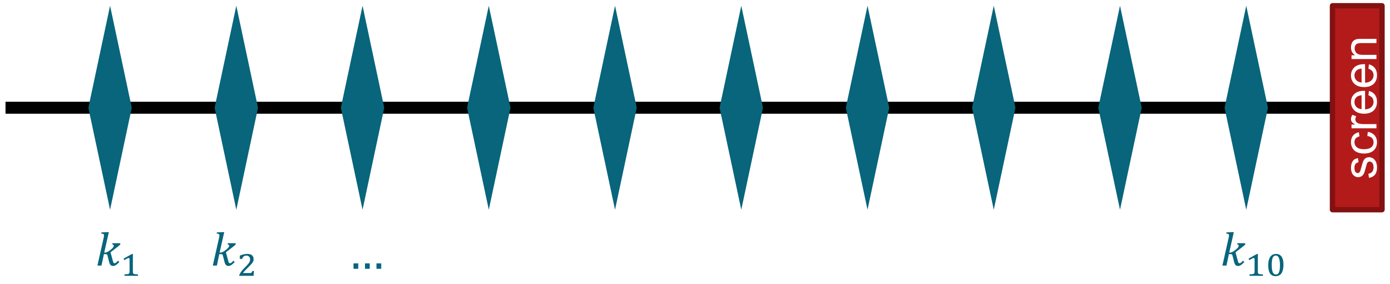
Application 1: High-dimensional Optimization

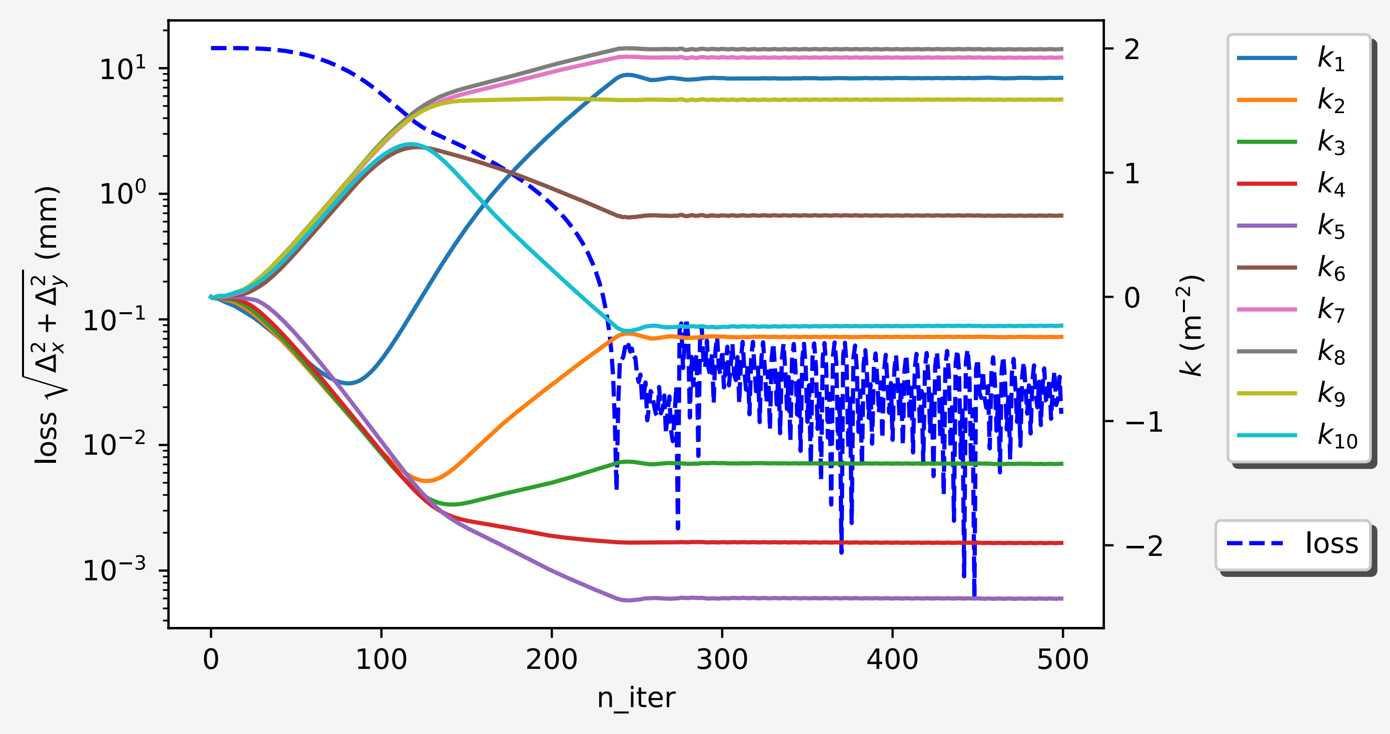


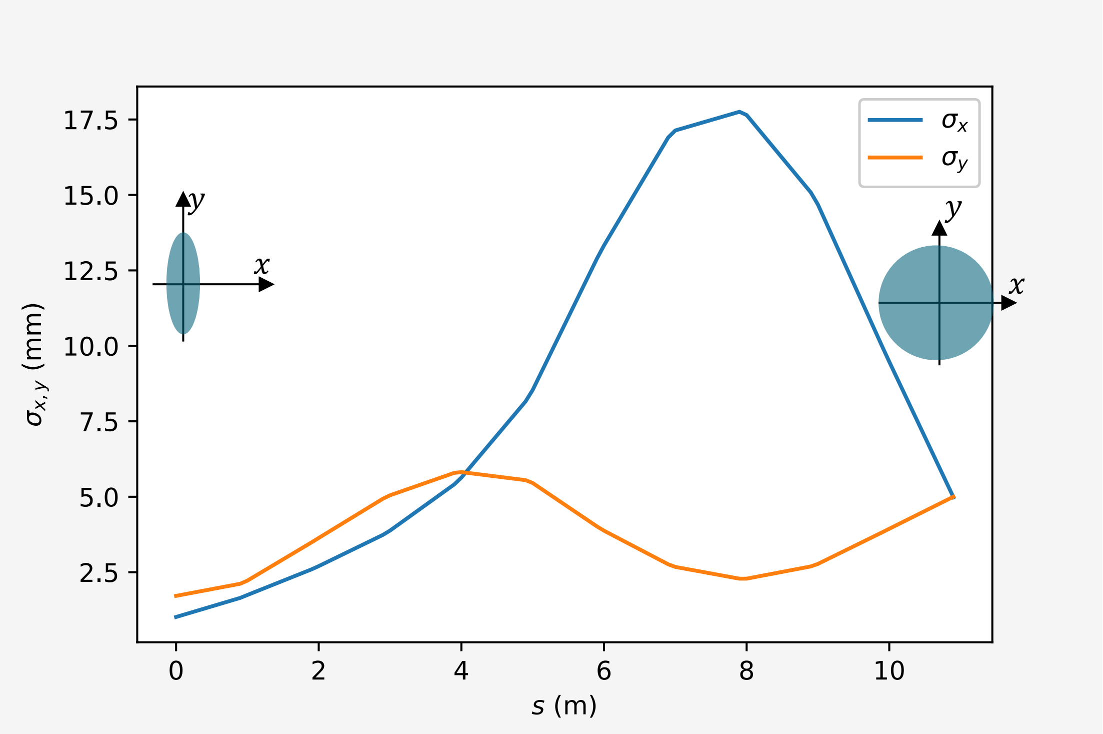
- Target: round beam with $\sigma_t = 5.00$ mm
- $\min \sqrt{(\sigma_x - \sigma_t)^2 + \left(\sigma_y - \sigma_t\right)^2}$
- Free parameters: $\{k_1, \ldots, k_{10}\}$
- Optimizer: ADAM



Results: 10 Quad Optimization





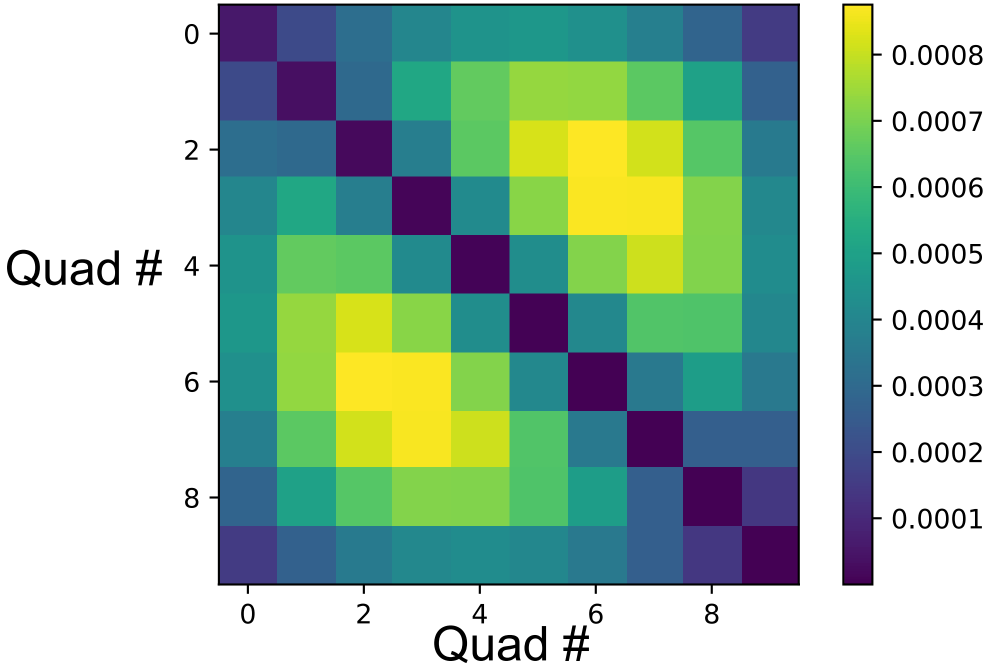


Application 2: Arbitrary derivative computation

Derivatives of **any output** WRT **any input**, regardless dimension and order.

Example:

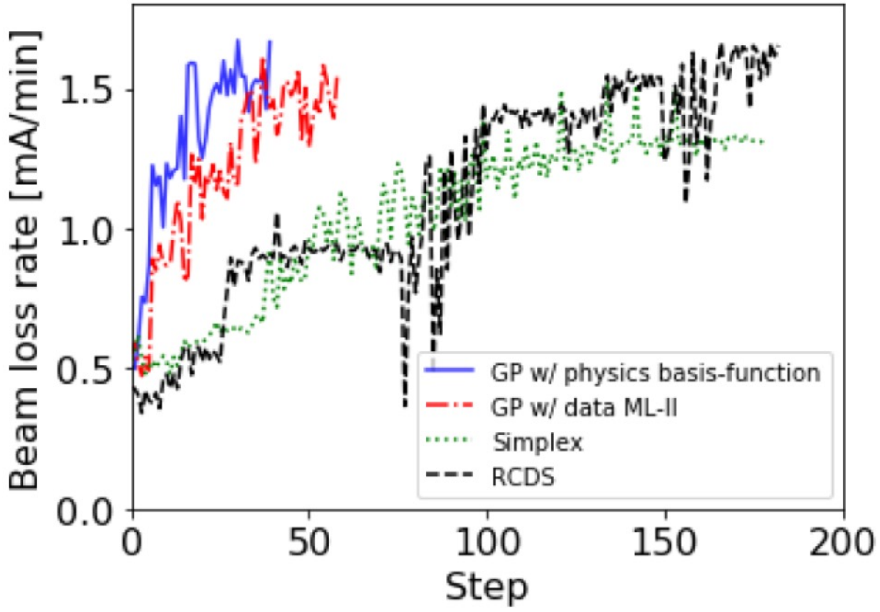
Hessian of beam size WRT 10 quad strengths



$$\frac{\partial^2 \sigma_x}{\partial k_i \partial k_j}$$

2 orders of magnitude faster than numerical differentiation

Physics informed Gaussian process for online optimization



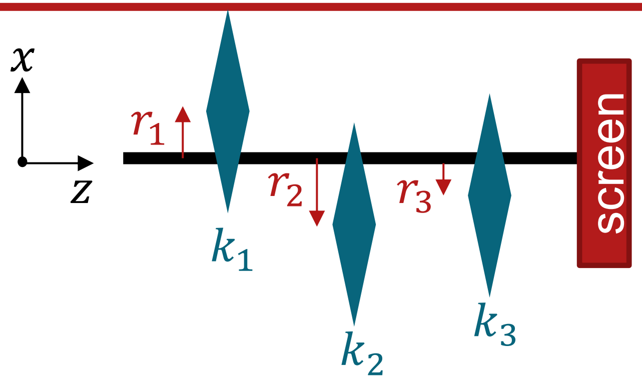
(a) Online machine optimization - Comparison of optimizers

A. Hanuka *et al.*, PRAB (2021)

Application 3: Model Calibration

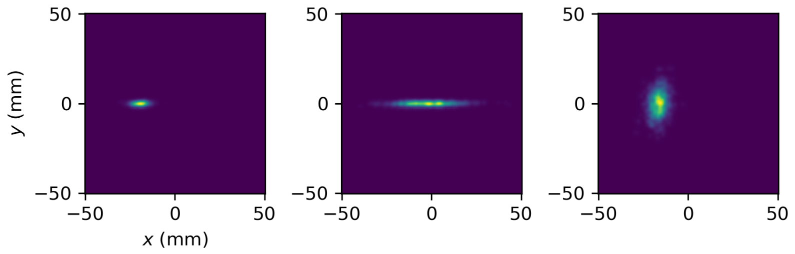
We want:

- Find x offsets $\{r_1, r_2, r_3\}$ of 3 quads



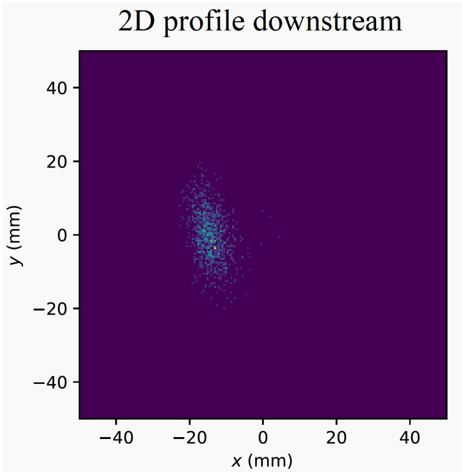
We have:

- 3 x-y "ground truth" beam profiles downstream
- 3 different sets of $\{k_1, k_2, k_3\}$

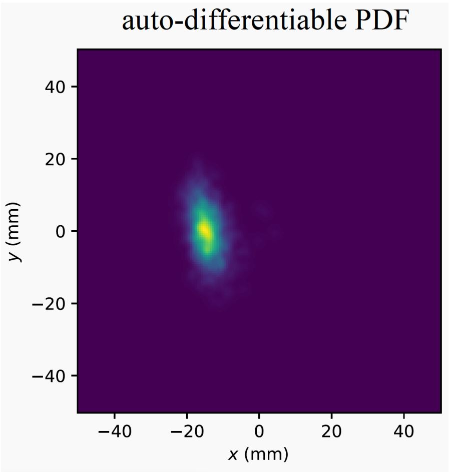


Procedure:

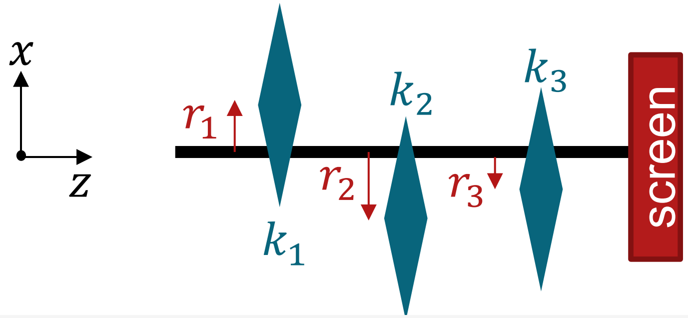
- $\{r_1, r_2, r_3\}$ such that beam profiles are as close as possible to ground truth
  - Loss function: KL Divergence
  - Differentiable beam profiles

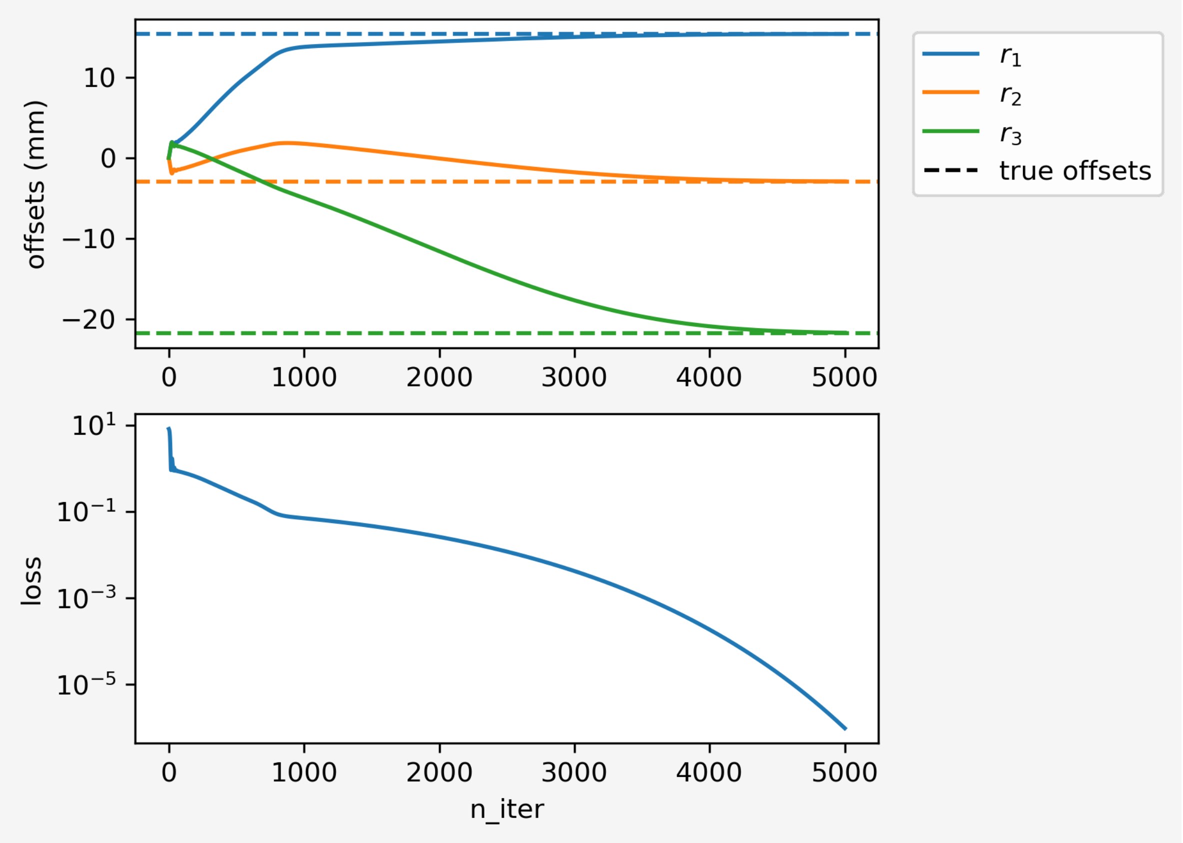


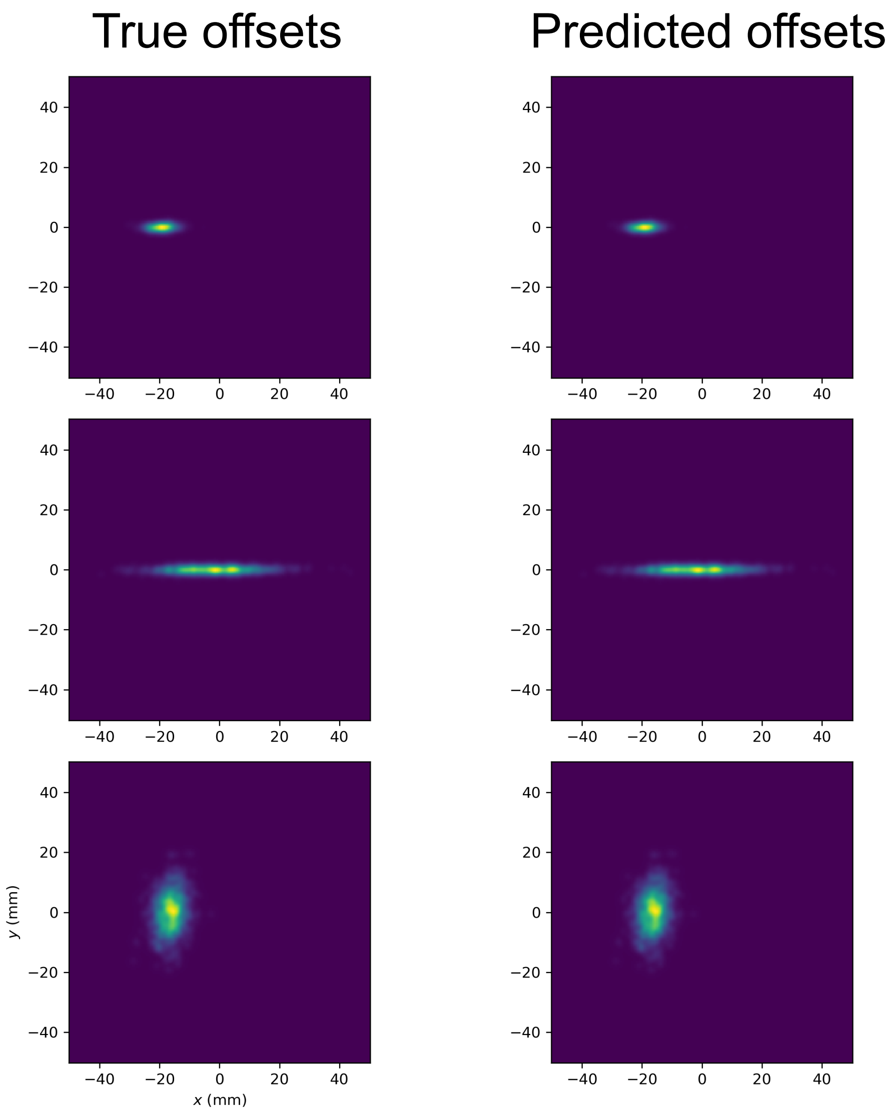
Kernel density estimation



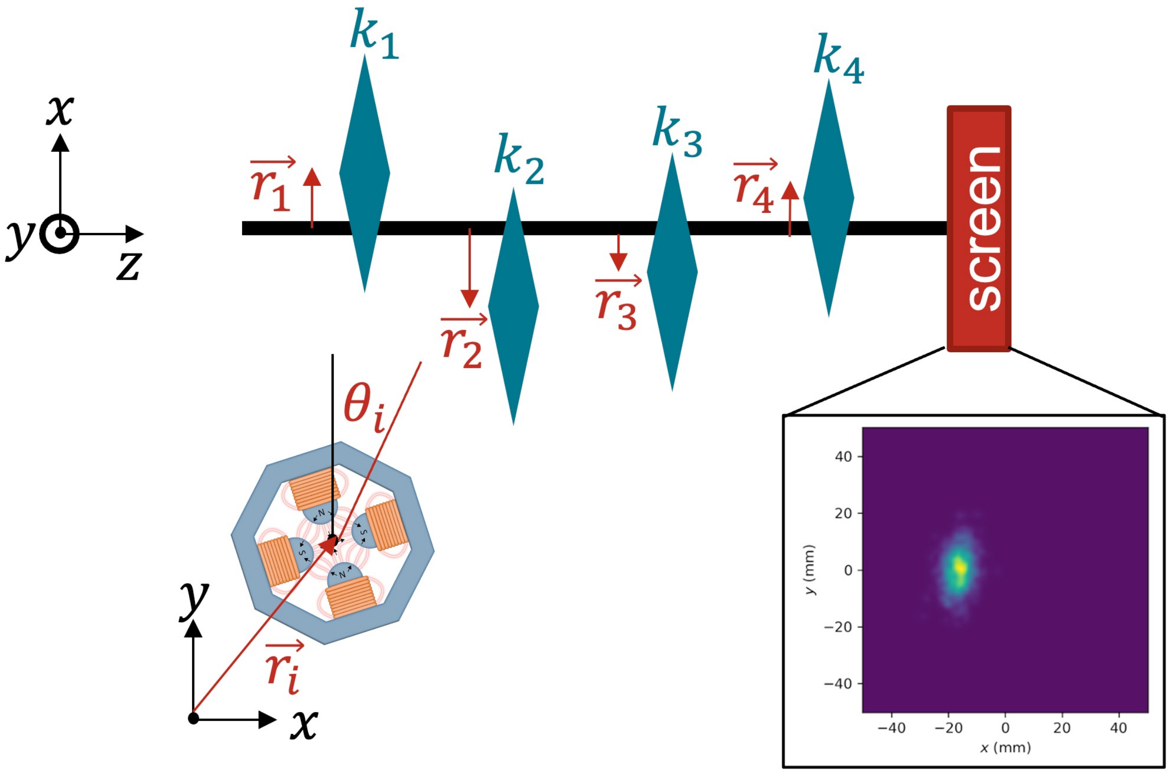
Results: Model Calibration

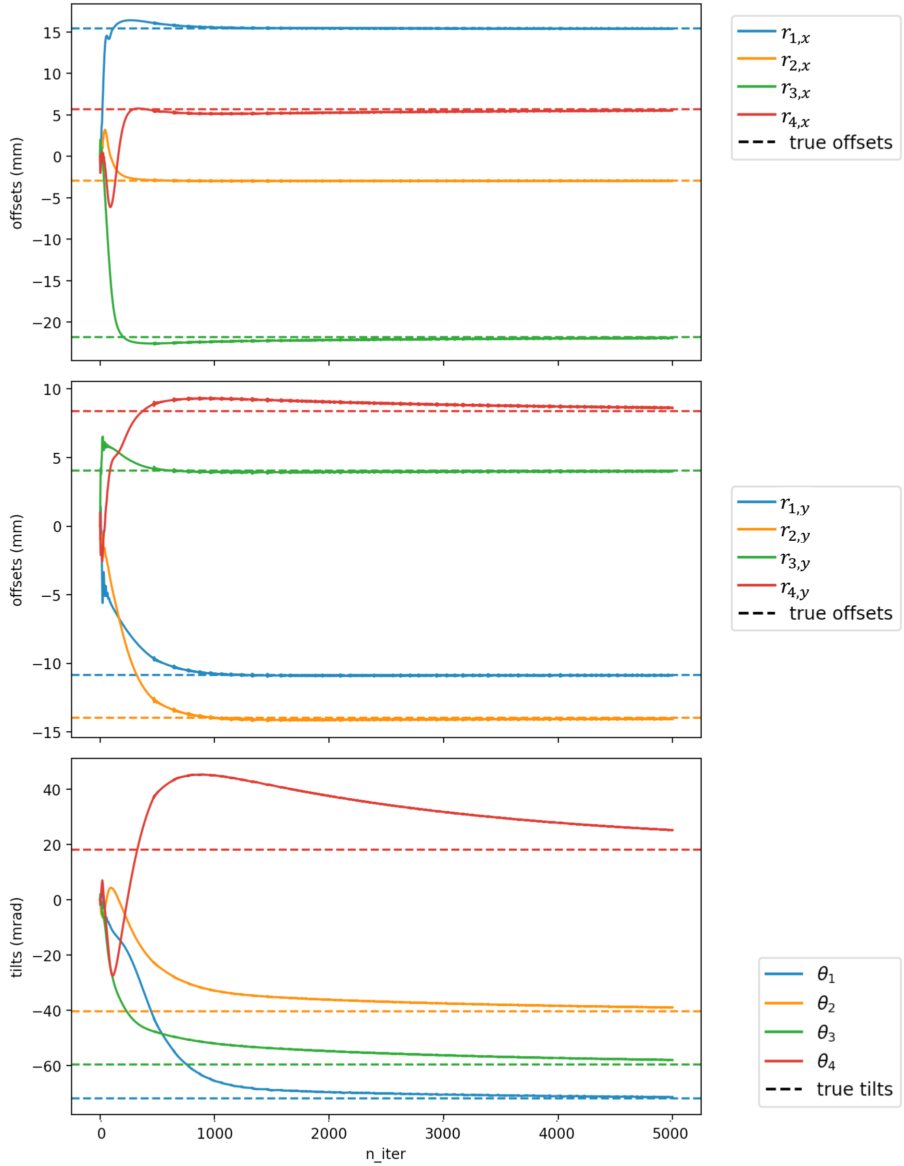




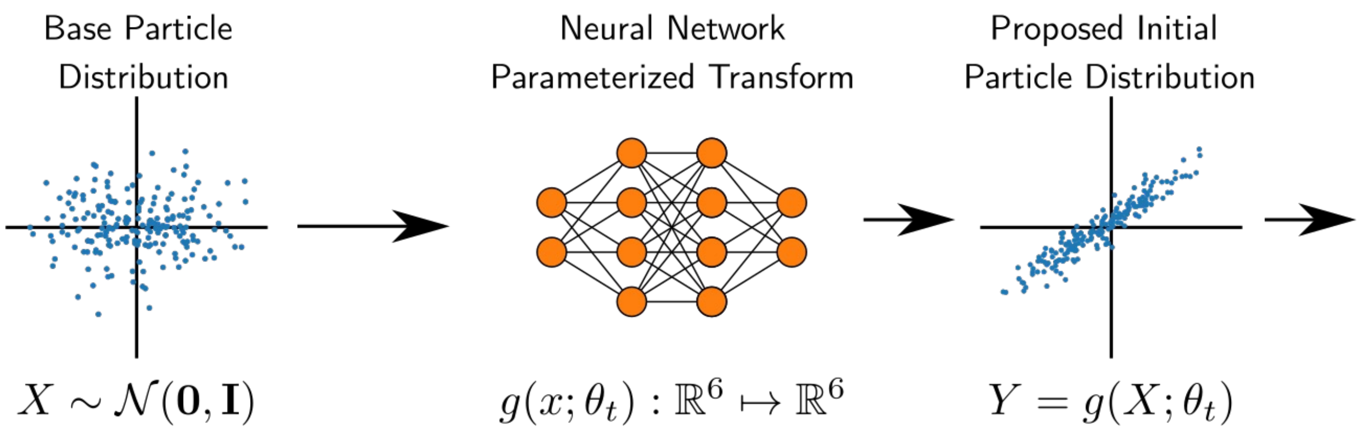


Model Calibration: 2D Offsets and Tilt



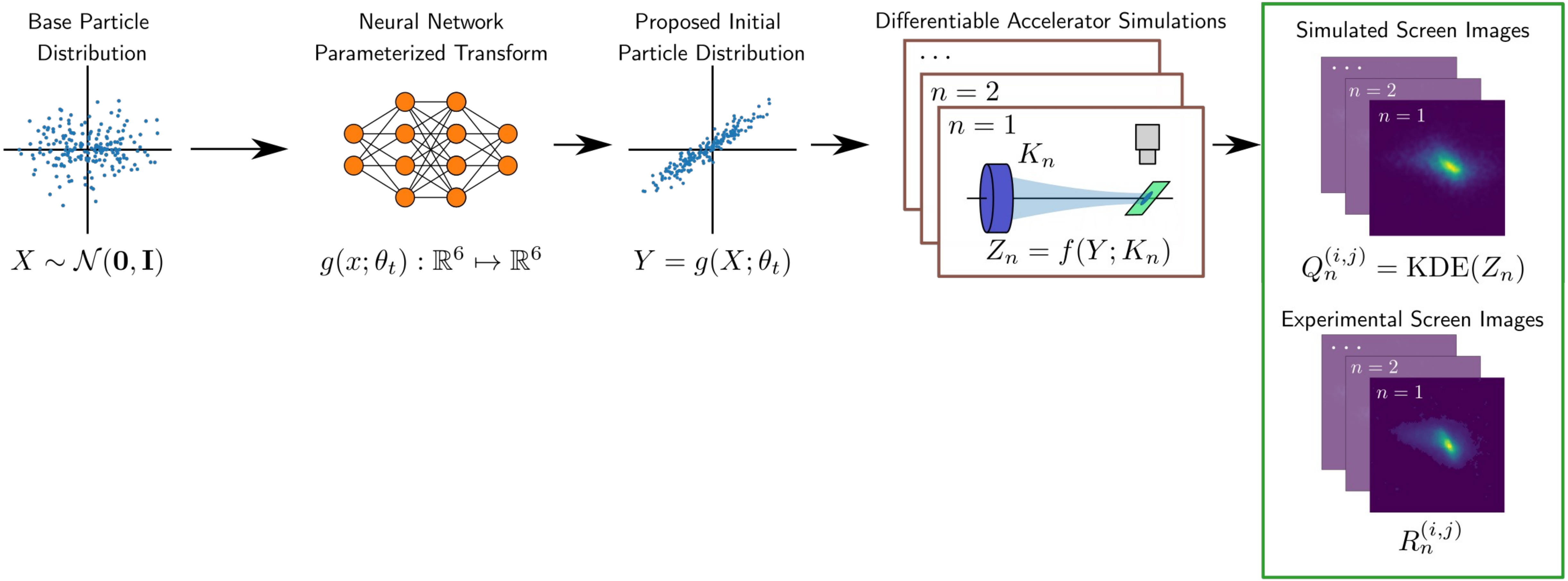


Application 4: Phase Space Reconstruction



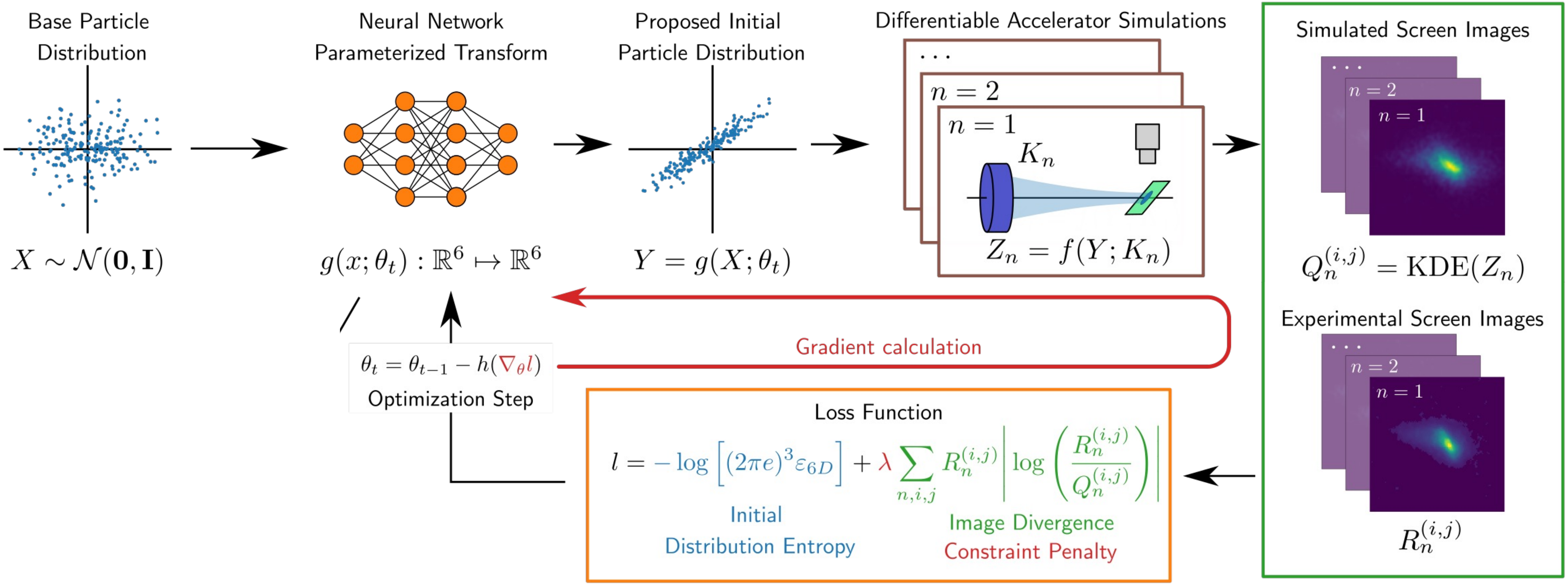
arXiv:2209.04505

Application 4: Phase Space Reconstruction



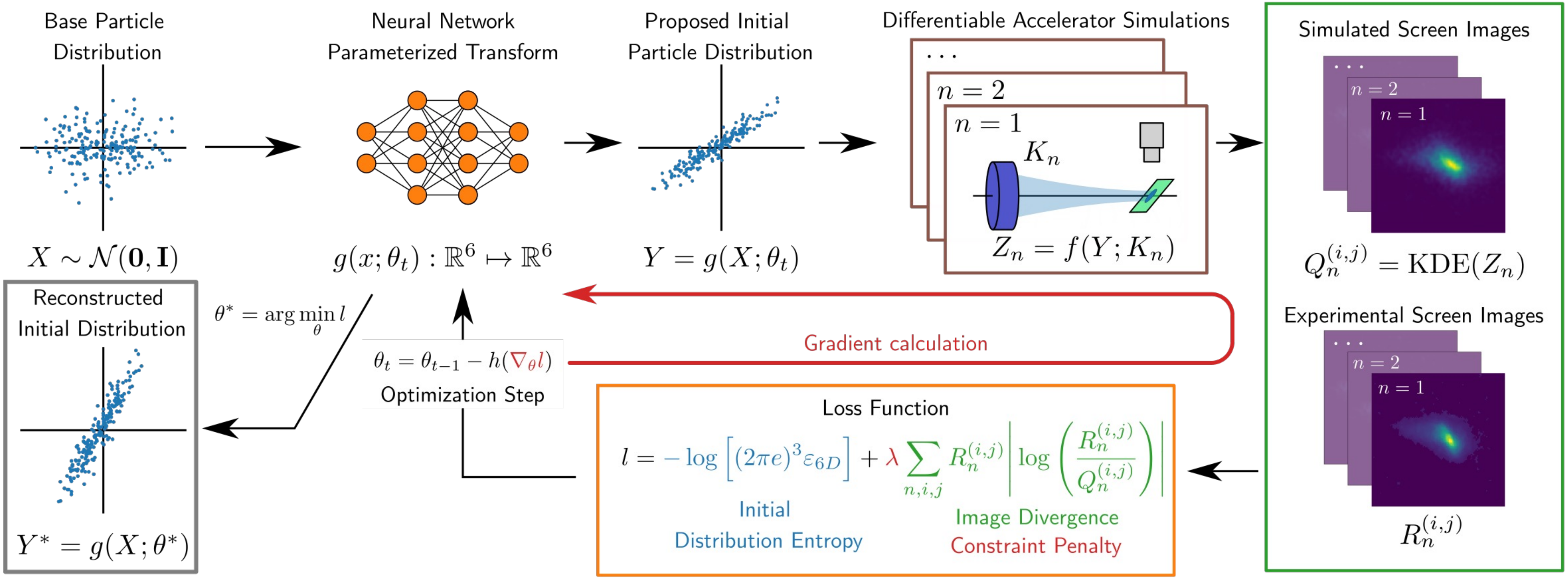
arXiv:2209.04505

Application 4: Phase Space Reconstruction



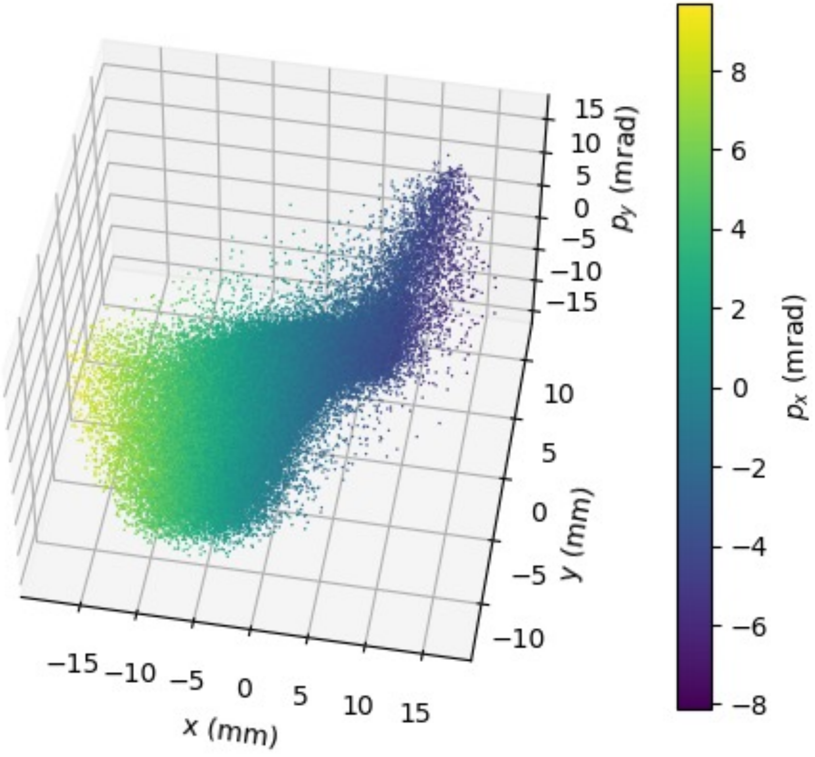
arXiv:2209.04505

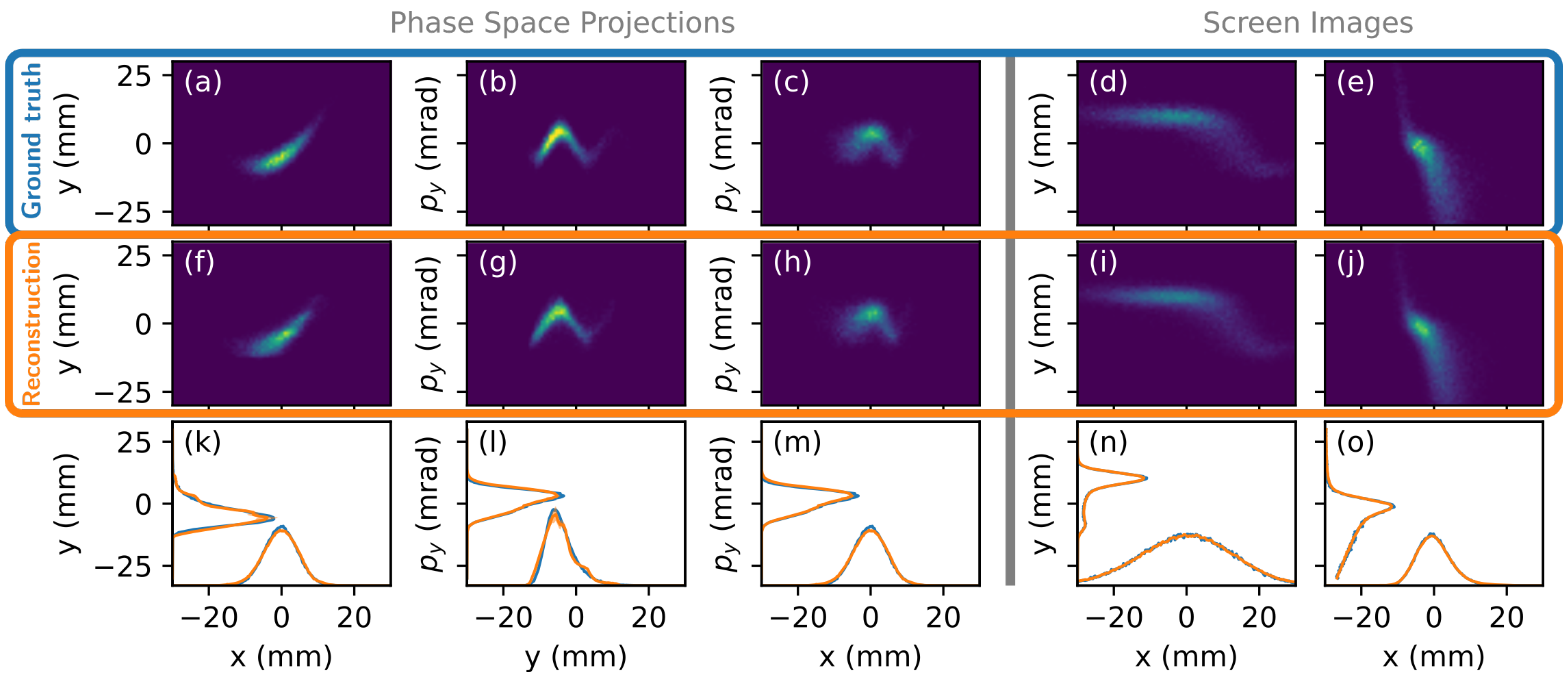
Application 4: Phase Space Reconstruction



arXiv:2209.04505

PS Reconstruction (Synthetic)

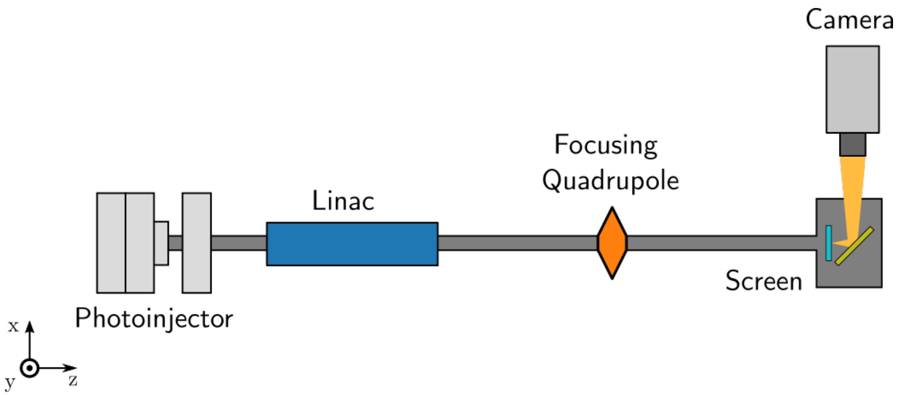


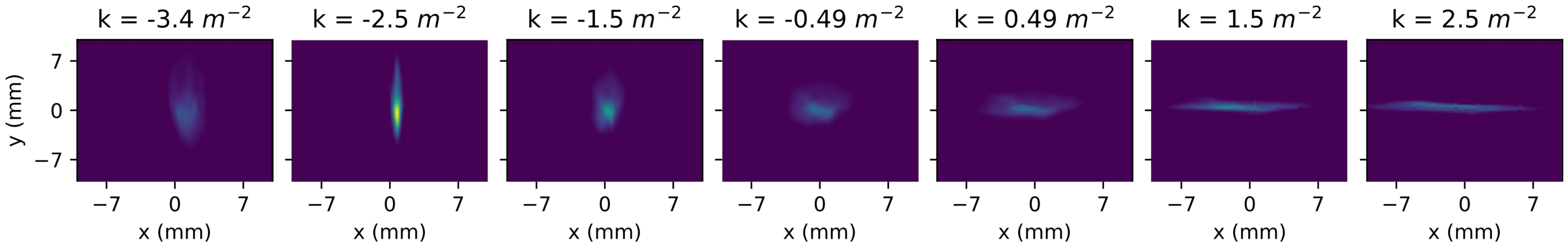


Parameter	Ground truth	RMS Prediction	Reconstruction	Unit
ε_x	2.00	2.47	2.00 ± 0.01	mm-mrad
ε_y	11.45	14.10	10.84 ± 0.04	mm-mrad
ε_{4D}	18.51	34.83*	17.34 ± 0.08	mm^2-$mrad^2$

arXiv:2209.04505

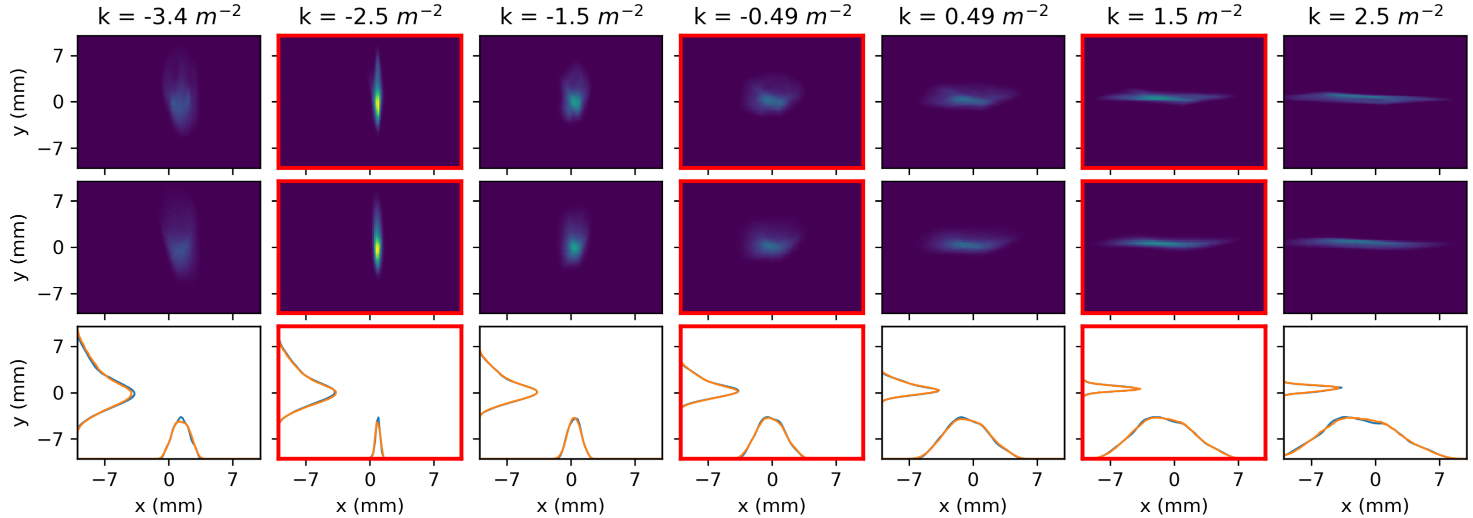
PS Reconstruction (Experiment at AWA)

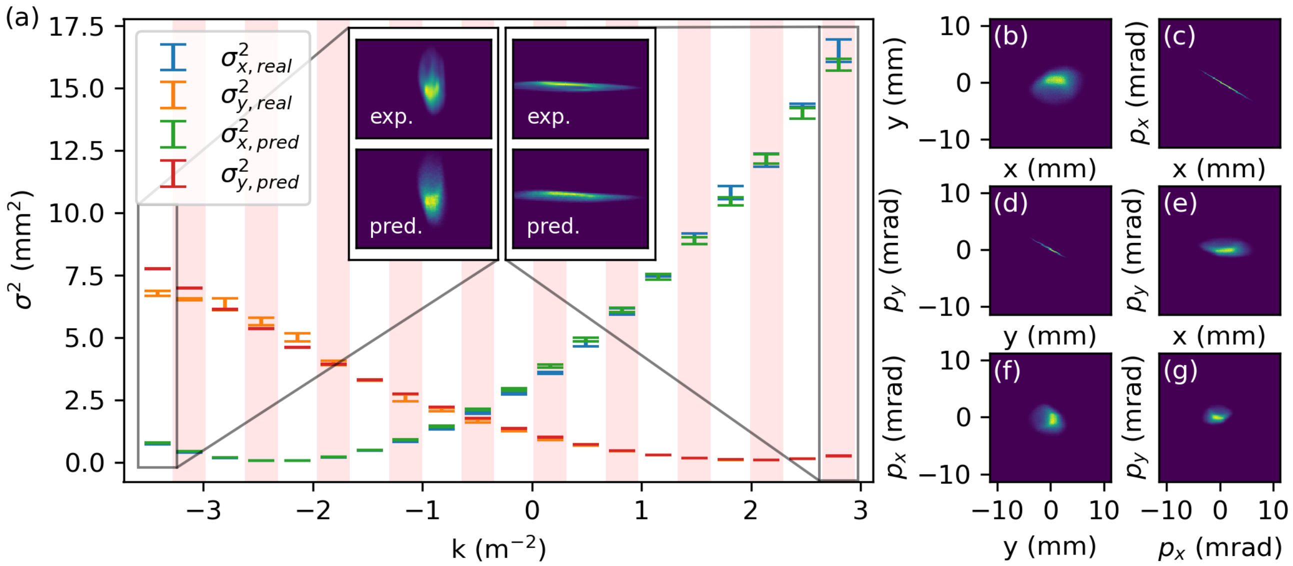




arXiv:2209.04505

PS Reconstruction (Experiment at AWA)





Parameter	RMS Prediction	Reconstruction	Unit
$\varepsilon_{x,n}$	4.18 ± 0.71	4.23 ± 0.02	mm-mrad
$\varepsilon_{y,n}$	3.65 ± 0.36	3.42 ± 0.02	mm-mrad

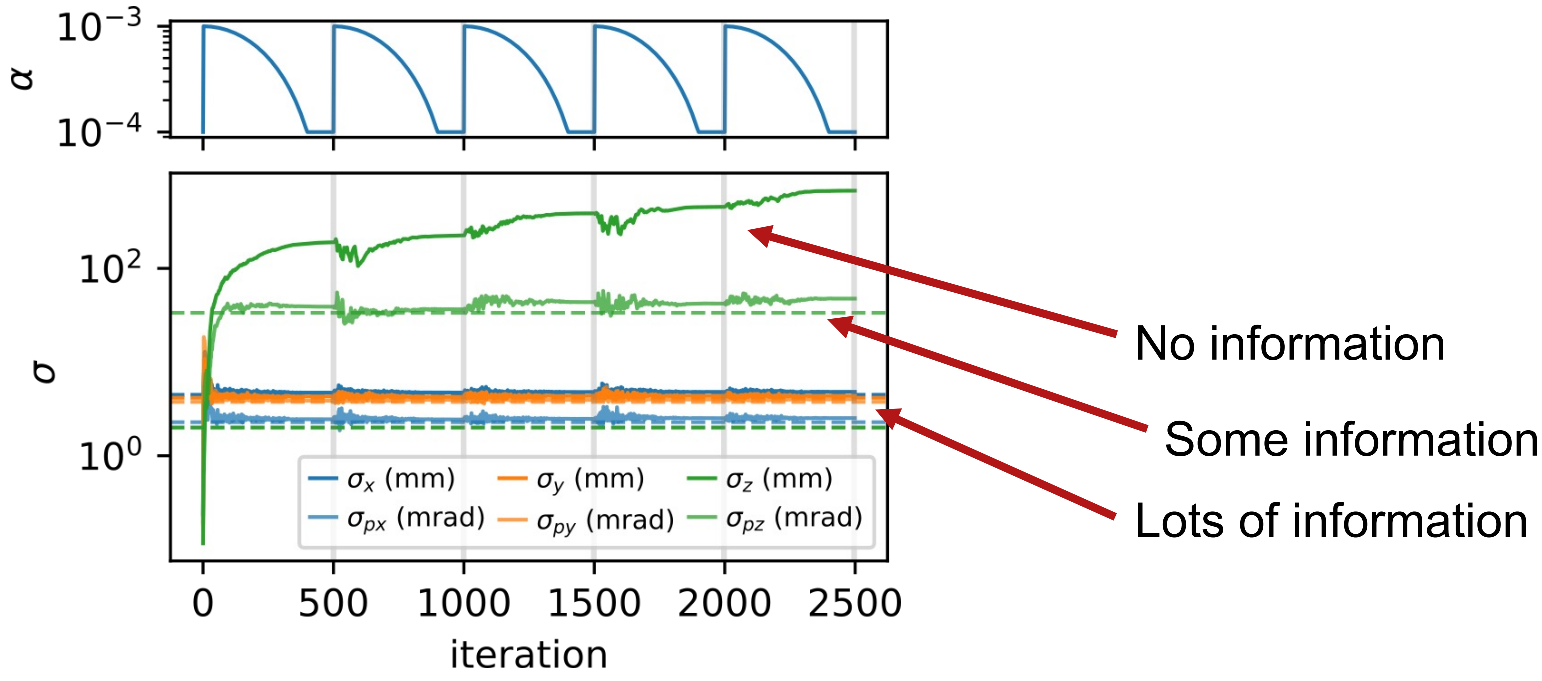
arXiv:2209.04505

Longitudinal Coordinates ?

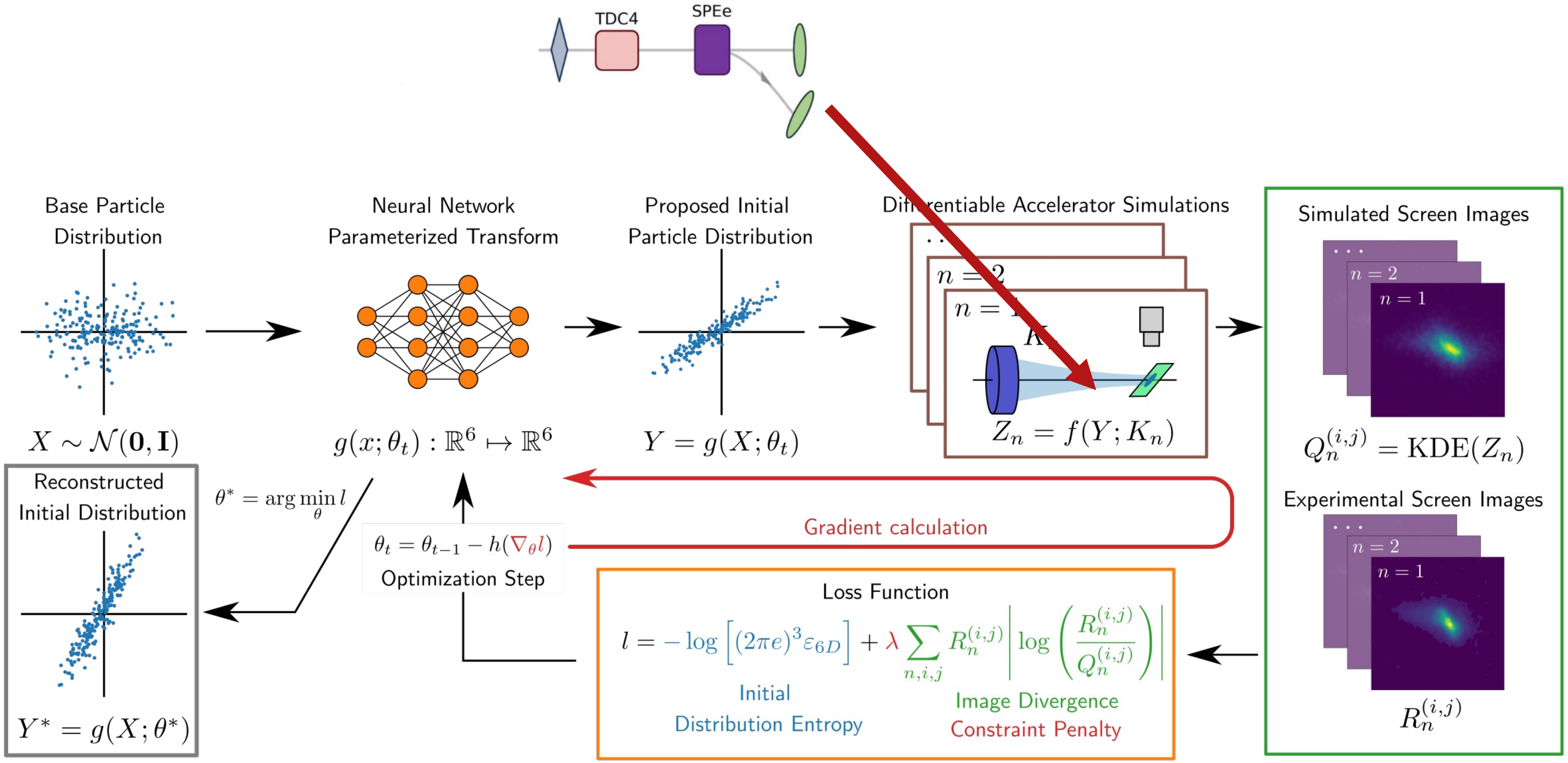
Quadrupole:

$$H = \frac{p_x^2 + p_y^2}{2(1+p_z)} + \frac{k_1(p_z)}{2}(x^2 - y^2)$$

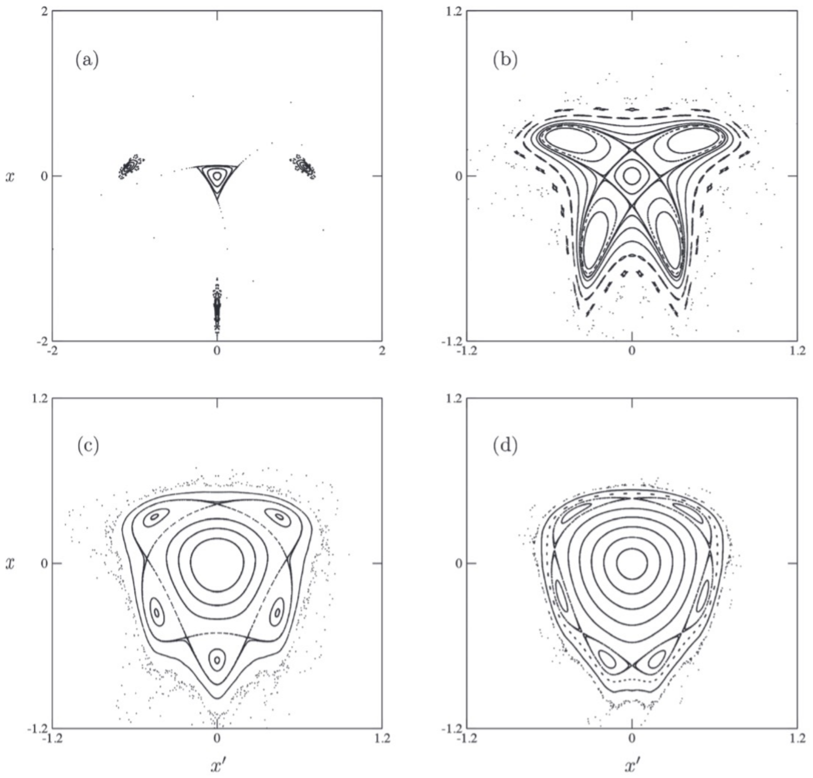
- Weak dependence on p_z via chromatic effects
- No dependence on z

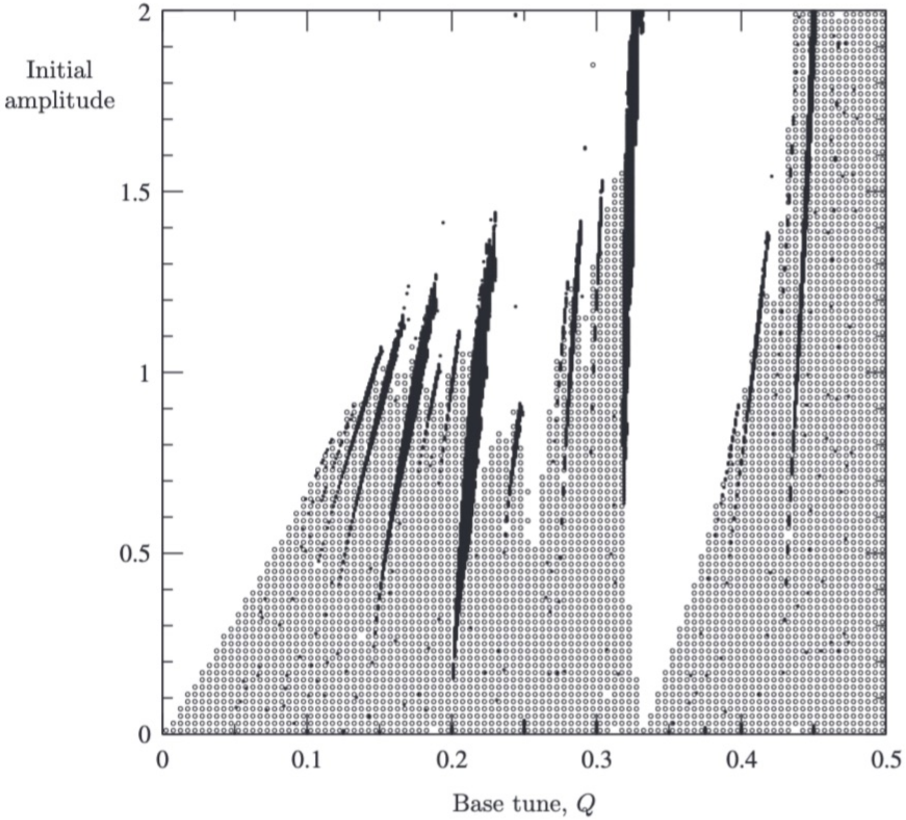


6D Phase Space Reconstruction



- Reverse-mode AD is memory intensive
- Costly tracking routines → costly derivative calculations
- Some quantities are inherently non-differentiable:





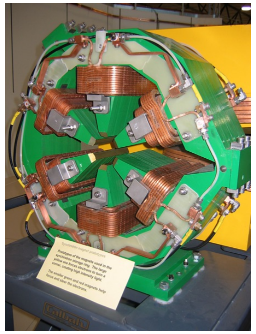
Peggs, Satogata, Introduction to Accelerator Dynamics

Summary

- Implemented fully differentiable Bmad routines in Python
  – Drift, Quad, Crab Cavity, RF Cavity, Bend
- Library agnostic: PyTorch, Numpy, Numba, CuPy, …
- Very flexible.
  – Derivatives of any output w.r.t. any input using auto-diff.
  – Full integration with ML modules from libraries such as neural nets
  – GPU compatible using Numba, CuPy
- Enables:
  – High-dimensional optimization.
  – Model calibration: alignment errors
  – Phase space reconstruction with limited diagnostics
- Open Source! “Bmad-X” github.com/bmad-sim/Bmad-X

Future work

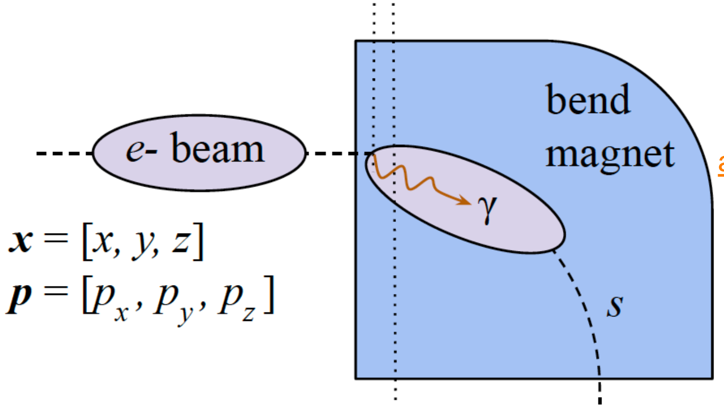
- More elements



en.wikipedia.org/wiki/Sextupole_magnet

en.wikipedia.org/wiki/Superconducting_radio_frequency

- Collective effects
  - CSR
  - Spacecharge



arxiv.org/abs/2203.07542

- More applications
  - Model calibration in experiment
  - Online optimization
  - Non-linear optics
  - Circular accelerators

Acknowledgements

The authors would like to thank **William Lou** for developing the differentiable bend and **David Sagan** for providing help with Bmad.

This work was supported by the U.S. National Science Foundation under Award PHY-1549132, the Center for Bright Beams.