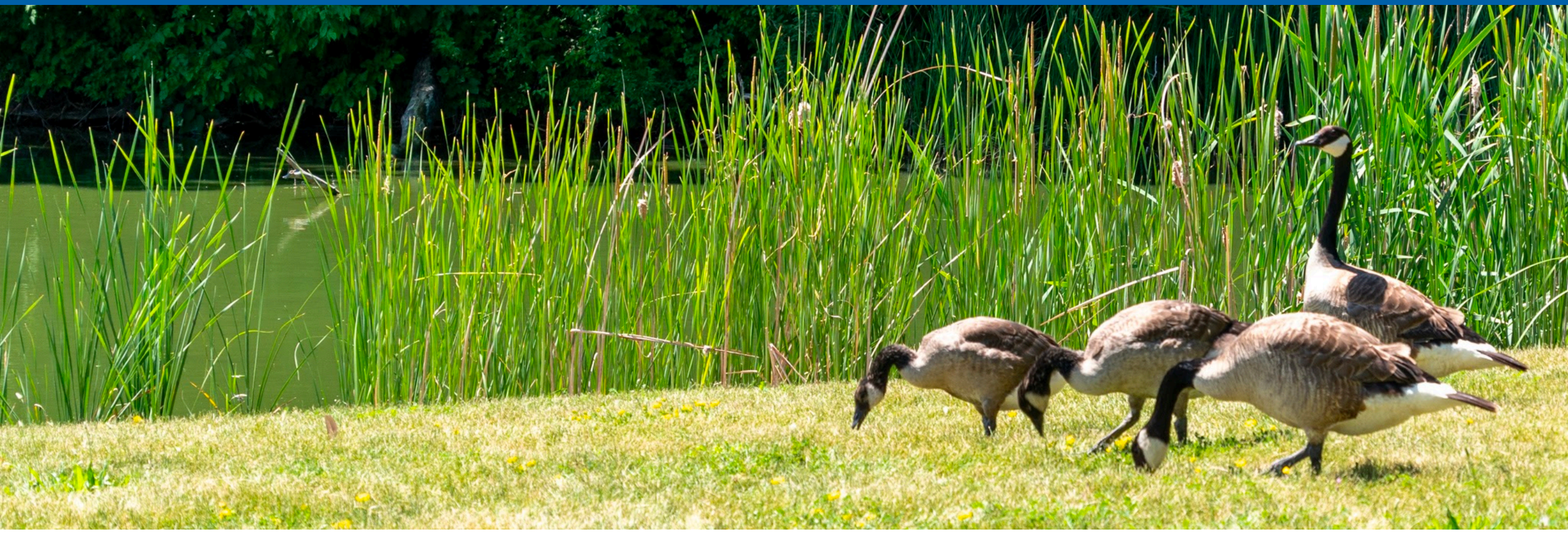


Image credit: Marguerite Tonjes

Differentiable Simulation for Muon Cooling

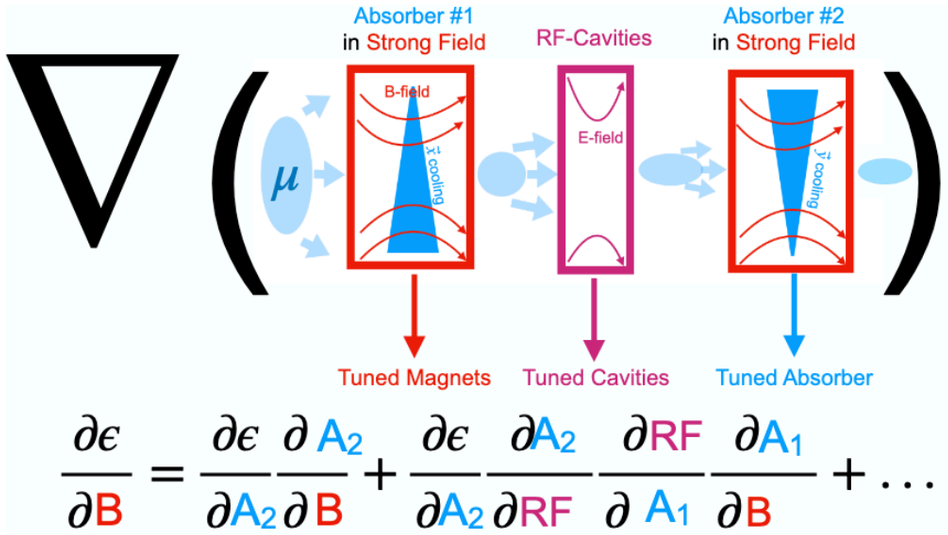
Kickoff meeting

Nick Smith

Dec 18, 2025

Deliverables

- Auto-differentiable beamline simulation
  - A reasonable set of building blocks is understood
    - Already a benchmarking activity
- Optimized 6D Muon Cooling
  - Huge parameter space, also competing designs
  - We can join an ongoing working group: IMCC WG4
- Optimized Final Muon Cooling
  - Current challenge seems to be high-B magnets?



$$\frac{\partial \epsilon}{\partial B} = \frac{\partial \epsilon}{\partial A_2}\frac{\partial A_2}{\partial B} + \frac{\partial \epsilon}{\partial A_2}\frac{\partial A_2}{\partial RF}\frac{\partial RF}{\partial A_1}\frac{\partial A_1}{\partial B} + \dots$$

- Beam Dynamics and Controls Simulator
  - Here we should tie in with MOAT project

Task	Year 1	Year 2	Year 3
Auto differentiable Beamline			
Optimized 6D Cooling			
Optimized End-to-End cooling			
Muon Cooling Control Simulator			

Table 1: Year by Year outline of major objectives, the Red targets digital twin construction and downstream optimization whereas Blue targets AI/ML based operations.

Auto-differentiable beamline simulation

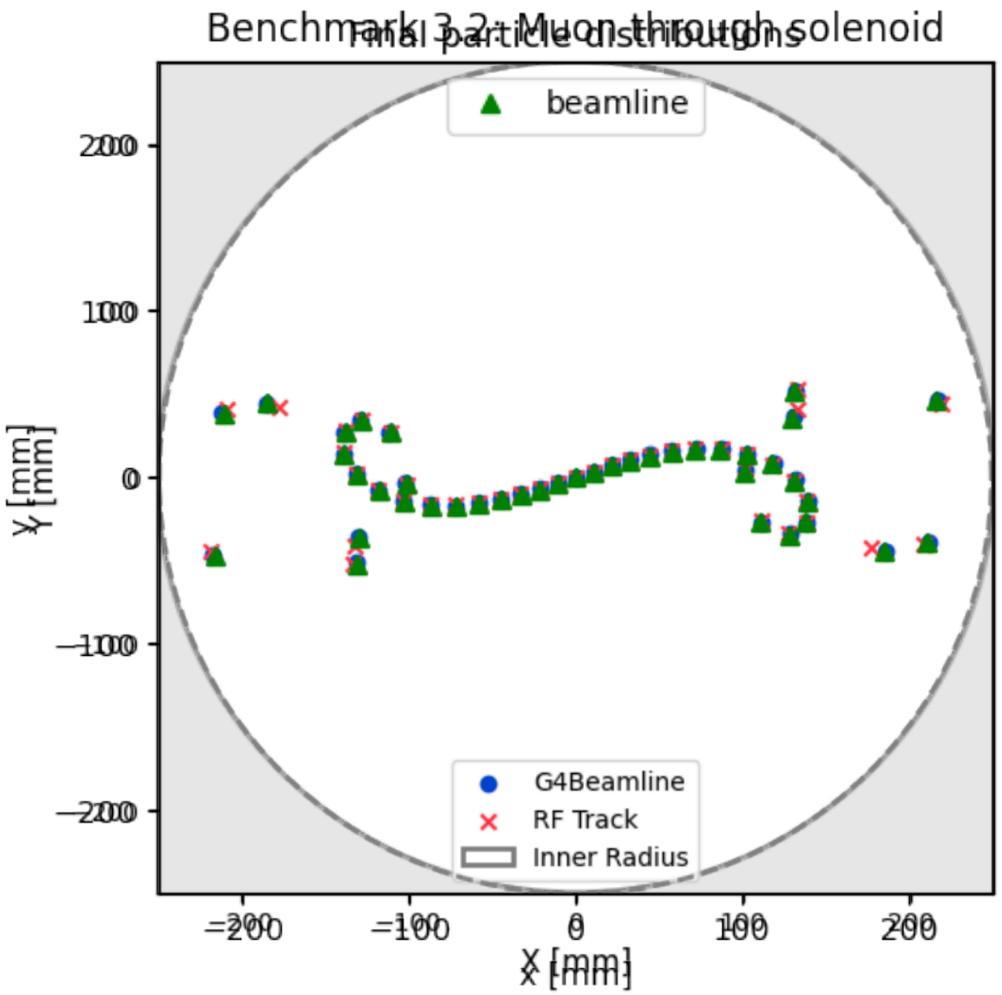
- How to approach? As part of LDRD proposal, identified codes:
  - **iCOOL**, a Fortran code specifically written for muon cooling
  - **G4Beamline**, a C++ application (with GUI) based on Geant4 toolkit with custom extensions to model beamline elements, maintained by T. Roberts (Muons, inc.)
  - **BDSim**, a C++ application based on Geant4 toolkit, maintained by Imperial College London
  - **RF-Track**, a C++ application with python & octave bindings, maintained by A. Latina (CERN)
  - **Synergia**, a C++ application leveraging MPI and the Kokkos performance portability framework for GPU acceleration, maintained by E. Stern (FNAL)
  - **Cheetah**, a Python application, using PyTorch for GPU acceleration and AD capabilities, maintained by DESY
  - **WarpX**, a C++ application also leveraging a variety of accelerated computing backends and capable of GPU and multi-CPU parallelization, maintained by Berkely Lab
- The plan was to enumerate how many requirements each met and how hard it would be to retrofit AD, and then go from there
  - Actually I started a spreadsheet while drafting the proposal
- But this was not funded, so I did what I wanted in my spare time
  - Build from scratch in Numpy ➜ Jax

beamline

- Building on a base of NumPy, then Jax
  - NumPy+SciPy is more “batteries included” with regard to special functions, methods
    - ODE integrators, Bessel functions, Elliptic integrals
    - Scikit-HEP/vector for 3- and 4-vector math
  - Jax has autodiff and GPU/TPU execution
    - Diffrax: Jax-based library for ODEs and also stochastic diff. eq. (but not jump)
  - Scikit-HEP/hepunits for dimensions (same as CLHEP, used e.g. in GEANT)
  - Build benchmarks and validate physics with NumPy, then go fast and optimize with Jax
- The hard(er) parts:
  - For absorber simulation, need multiple scattering and energy loss/straggling
    - Jump stochastic diff. eq. can formalize this, but only off-the-shelf in Julia
    - I do not understand why GEANT does not have a refinable algorithm with error estimates
  - For collective effects, need Poisson solver
    - Implementing multigrid using stencils from convolution kernels in Jax seems doable
  - Real life is always field maps, not toy sheet current solenoids, so we need good splines
- Currently very work-in-progress / exploratory phase
  - https://github.com/nsmith-/beamline

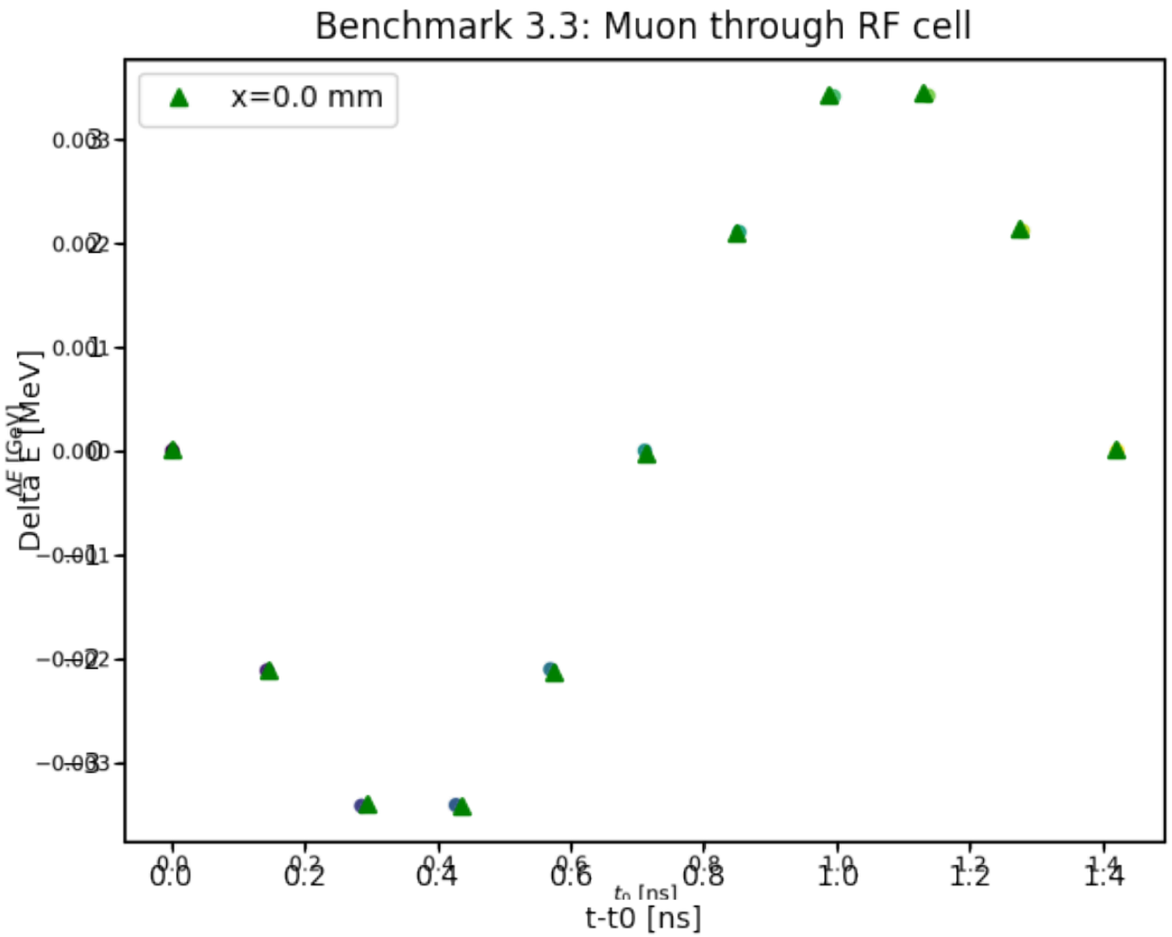
Solenoid benchmark

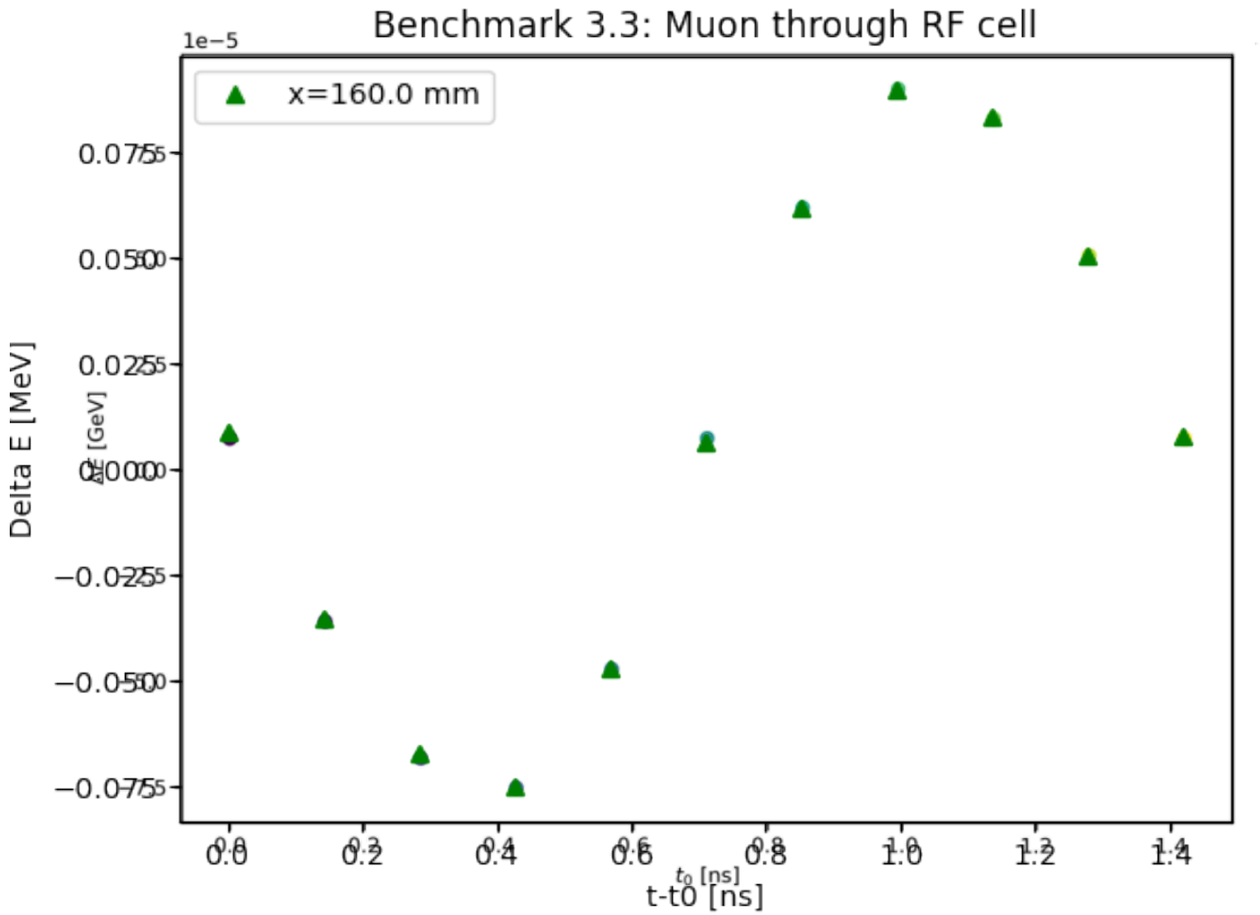
- Comparing to the RF-Track presentation
  - Off to a good start



RF Cavity benchmark

- Comparing to the BDSIM presentation
  - Off to a good start





More directions

- Large community interested in applying differentiable programming techniques to HEP problems, such as detector design and beam tuning
  - Calorimeter geometry optimization ≈ wedge absorber
  - Demonstration of optimizing 10 quadrupoles to a target beam profile in 1/100 the time of numeric differentiation