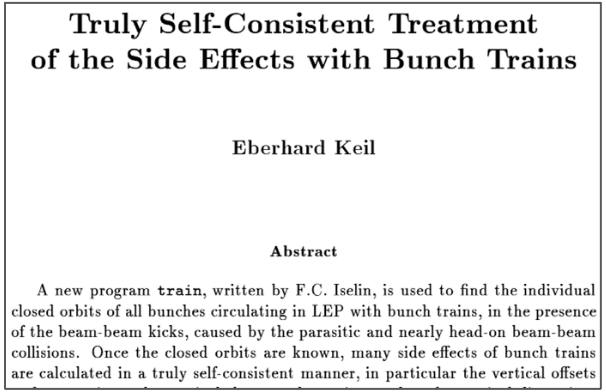
pyTRAIN

a modern TRAIN implementation

Michi Hostettler, Xavier Buffat, Tobias Persson, Tatiana Pieloni, Jorg Wenninger

a brief history TRAIN

- **iteratively find self-consistent closed orbits**
  - under the presence of beam-beam effects
  - in the many-bunch case ("trains")
  - using second-order sectormaps from MAD-X
  - “soft-Gaussian” approach
- **pioneered by E. Keil, F.C. Iselin for LEP**
- **applied to LHC design by W. Herr, H. Grote**
  - showed clear advantage of V-H alternating crossing scheme over H-H crossing
  - showed "PACMAN" effects (bunches missing long-range encounters due to kicker gaps)
  - later further extended by T. Pieloni, A. Gorzawski, M. Hostettler, X. Buffat, A. Ribes

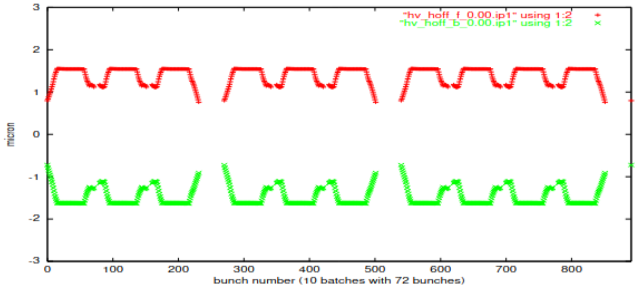


Truly Self-Consistent Treatment of the Side Effects with Bunch Trains

Eberhard Keil

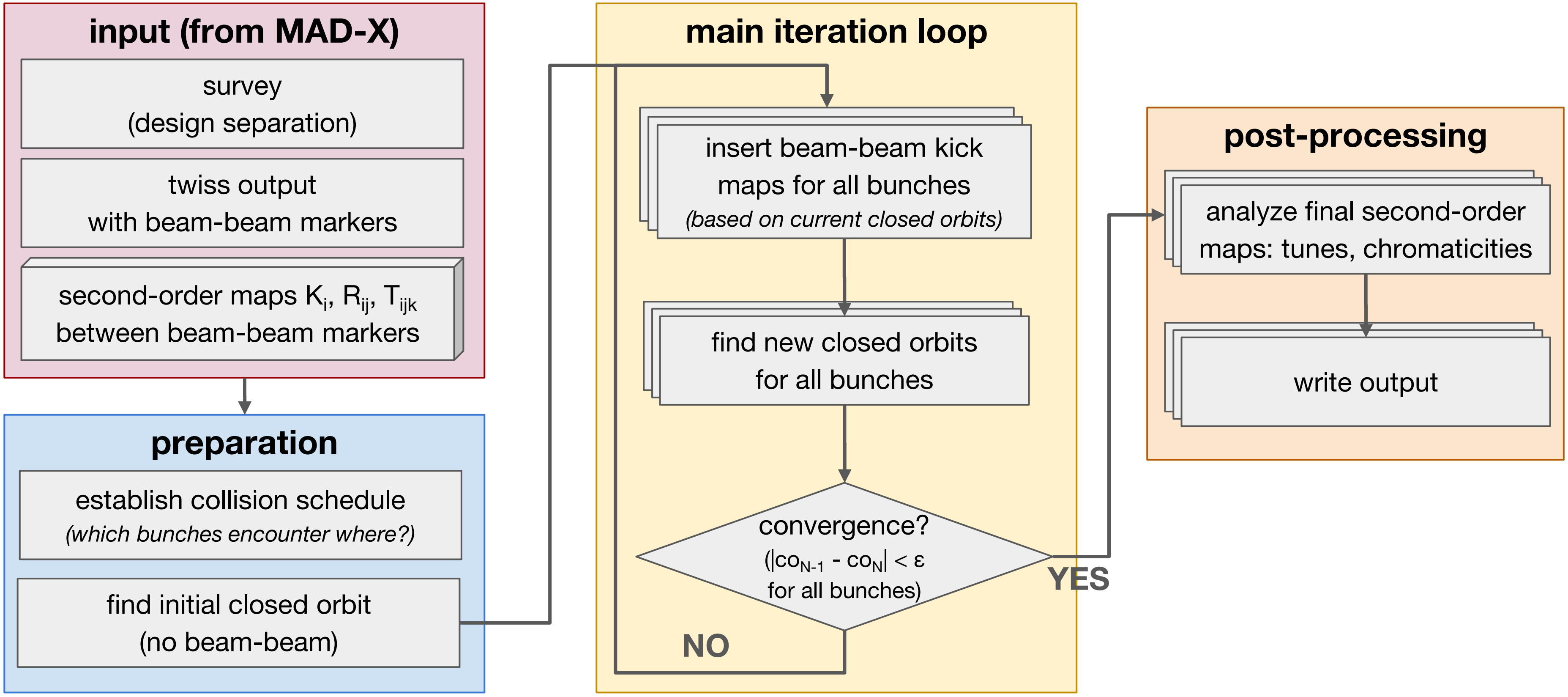
Abstract

A new program train, written by F.C. Iselin, is used to find the individual closed orbits of all bunches circulating in LEP with bunch trains, in the presence of the beam-beam kicks, caused by the parasitic and nearly head-on beam-beam collisions. Once the closed orbits are known, many side effects of bunch trains are calculated in a truly self-consistent manner, in particular the vertical offsets



W. Herr, "Features and implications of different LHC crossing schemes", LHC project report 628, 2003.

TRAIN in a nutshell



historic TRAIN limitations

- **`ltrain.f`, one flat file, 12145 lines of FORTAN-77**
  - local implementation of all numeric primitives (linear algebra, …)
    - partially copied from historic MAD-8 or MAD-X code
    - not always numerically stable or the most efficient
  - no version control, no changelog (ktrain, ltrain, mtrain, …)
- **historic input formats**
  - flat files of records/numbers

    `read(..., *)` in
  - MAD-X maps: historic scripts

```
# 1=full_coll 2=nturn 3=debug 4=# of out_bunches 5=out_pos 6=write_norm
# 7=xi_fact 8=hofact 9=amp_bunch (0=all,- every..) 10= amp_fac (see below)
# 11=lumi_hist 12=beam_2 offset (half-buckets)
# 1    2    3    4    5    6    7    8    9   10   11   12
  1    1    1    0   16    0    1    1    0    0    0    0
# list of out_bunches
 10 16 22 0 0 0 0 0 0 0 0 0 0 0
```

- **limited extensibility & scriptability**
  - today most analysis is done in python / Jupyter
  - first version of pyTRAIN: TRAIN python interface
    - limited to running TRAIN in full, no control over the process, output only for BB points

the FORTRAN TRAIN code served us well for many years

now it is time to move on

pyTRAIN: a modern re-implementation

- **complete re-implementation in python**
- **using numpy and scipy primitives**
  - linear algebra, Faddeeva function, …
- **interface to MAD-X via cpymad**
  - reading MAD-X output from files also possible
- **total 1050 lines of python code**
  - 44 lines of Cython: concatenation of second-order maps
- **performance similar to FORTAN-77 code**
  - slightly slower - not the first priority
  - few minutes to solve full LHC with ~2400 bunches

```
> wc -l *.py *.pyx
  102 beambeam.py
  244 cpymad.py
   79 fileio.py
    6 __init__.py
   80 machine.py
   18 particles.py
   89 processor.py
  229 solver.py
  203 twiss.py
   44 operations.pyx
 1094 total
```

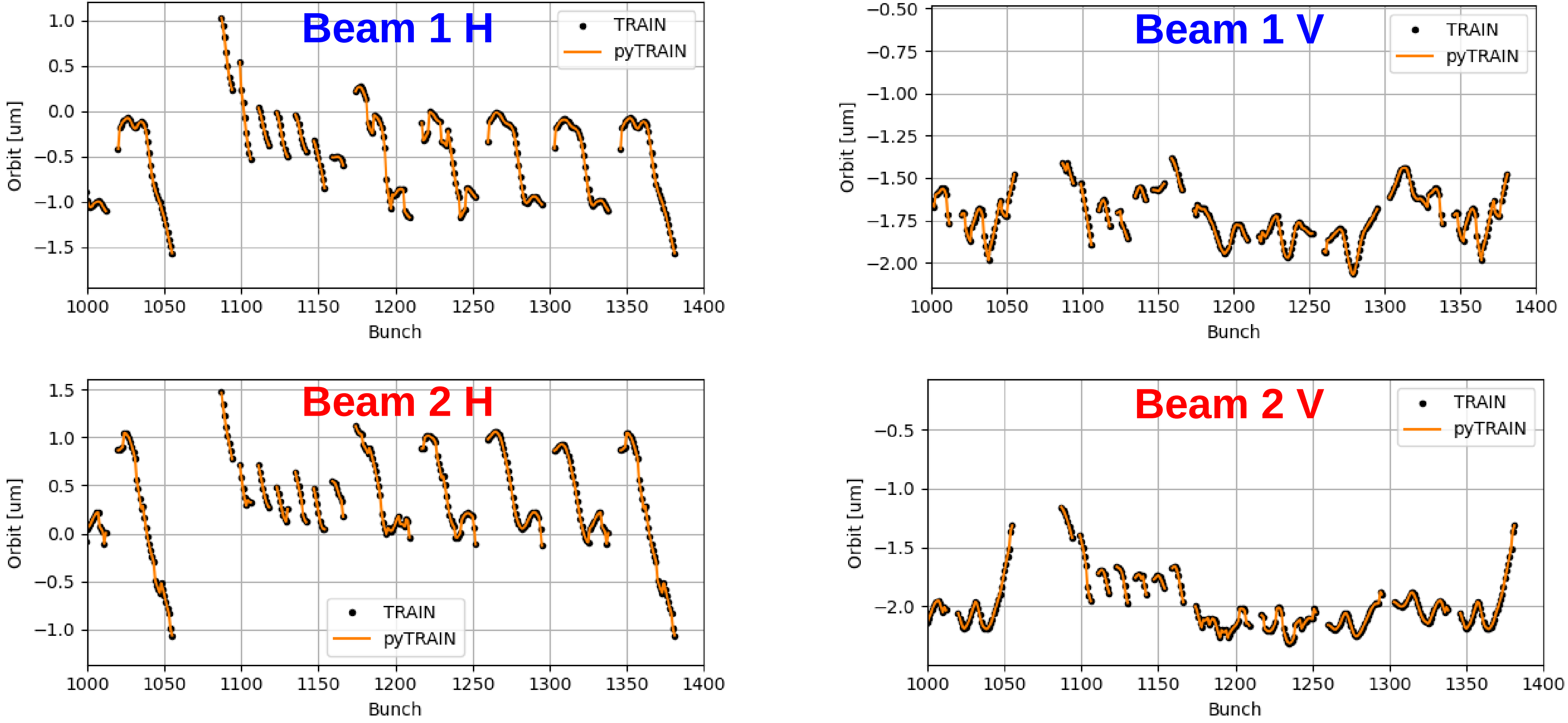
https://gitlab.cern.ch/mihostet/pytrain/-/tree/pure-python

pyTRAIN - basic usage

```
from pytrain.fileio import read_train_files
from pytrain.machine import FillingScheme
from pytrain.solver import solve_train

# read survey, twiss & sector map input files (alternatively use included cpymad utils)
machine, twiss_b1, twiss_b2, maps_b1, maps_b2 = read_train_files('train-output')

# construct a "filling scheme": bunch intensities & normalized emittances
filling_scheme = FillingScheme(int_b1, int_b2, emit_b1x, emit_b1y, emit_b2x, emit_b2y)

# solve self-consistent orbits with BBLR interactions
result = solve_train(machine, filling_scheme, twiss_b1, maps_b1, twiss_b2, maps_b2)

# bunch-by-bunch closed orbit at any element
co_b1_x, co_b1_y = result.bunch_positions_b1('MKIP5')
co_b2_x, co_b2_y = result.bunch_positions_b2('MKIP5')
```

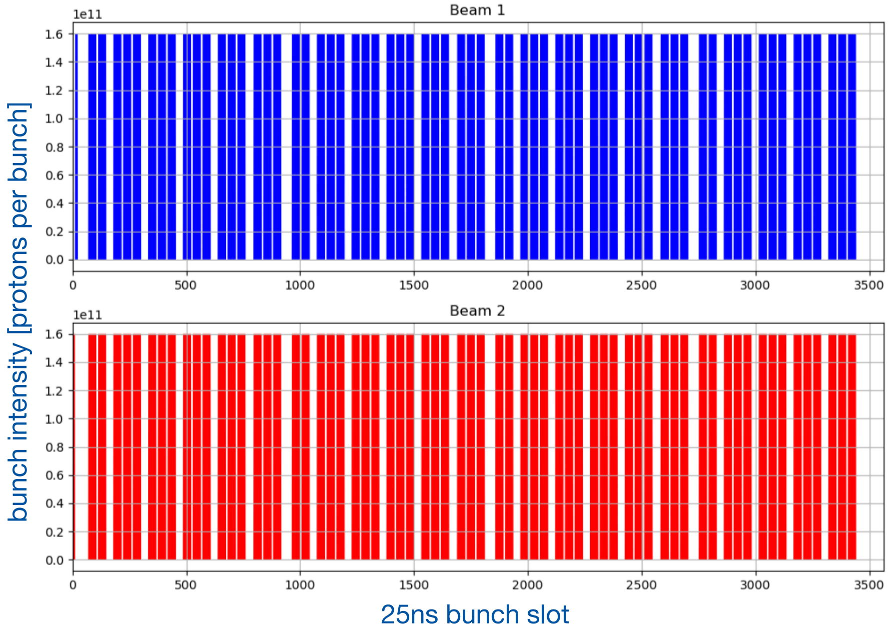
https://gitlab.cern.ch/mihostet/pytrain/-/tree/pure-python

benchmarking: TRAIN vs pyTRAIN



beam-beam long-range effects in LHC

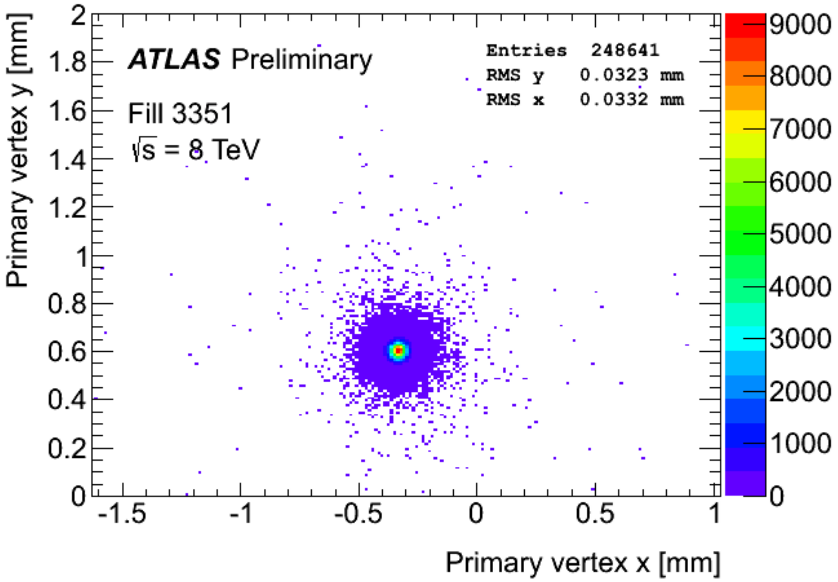
- **~2400 bunches spaced by 25ns**
  - longer gaps for kicker rise times
- **4 interaction regions with common vacuum chamber**
  - long-range beam-beam encounters
  - "pacman" effects due to kicker gaps (missing LR encounters)
  - "super-pacman" effects as IRs not symmetric (missing head-on colls.)
- **luminosity levelling by beta* and separation: changing optics**



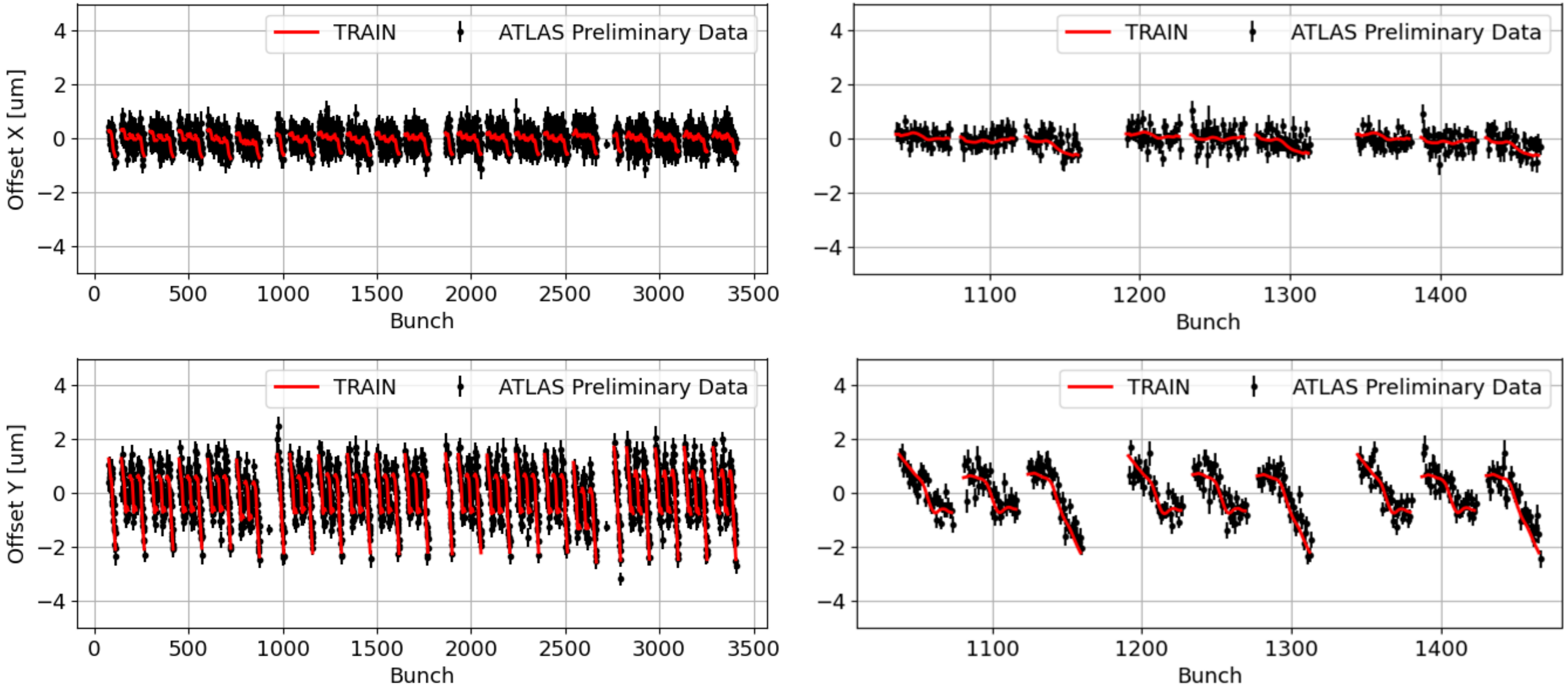
more details in talk of T. Pieloni

luminous centroid position at experiments

- **experiments measure the primary vertex positions ("beam spot")**
  - offline reconstructed from tracker data
  - "luminous region" size
  - "luminous centroid" position
  - high-statistics data collected during calibration sessions
    - interferes with physics data taking
- **"luminous centroid": center of the overlap region**
  - average position of the two beams
  - measured bunch-by-bunch



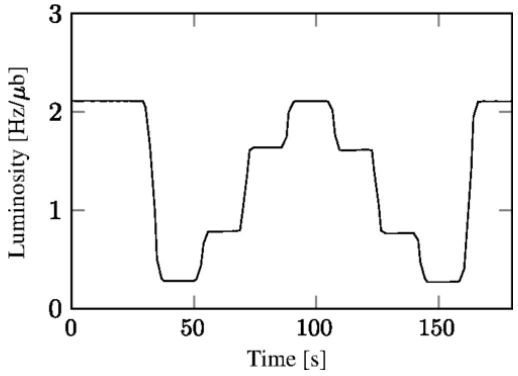
ATLAS luminous centroid position



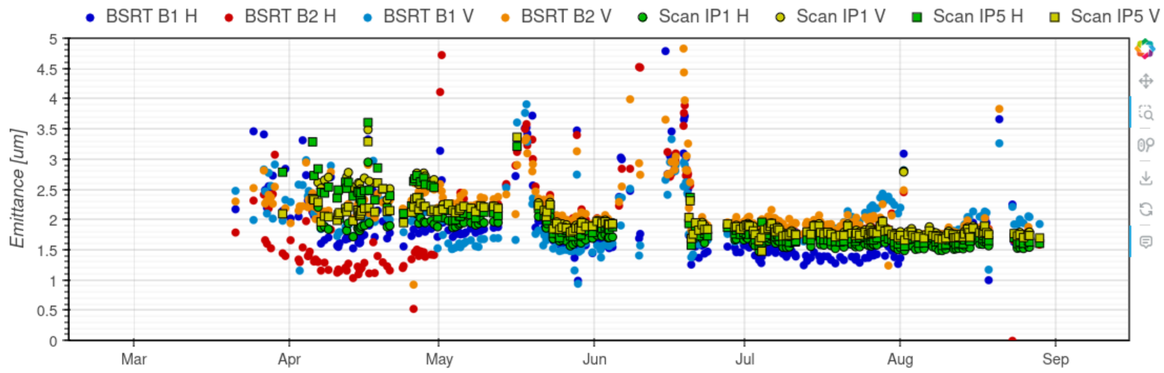
Preliminary BeamSpot data courtesy of the ATLAS collaboration. LHC fill 9653 / ATLAS run 476033, 2024-05-28

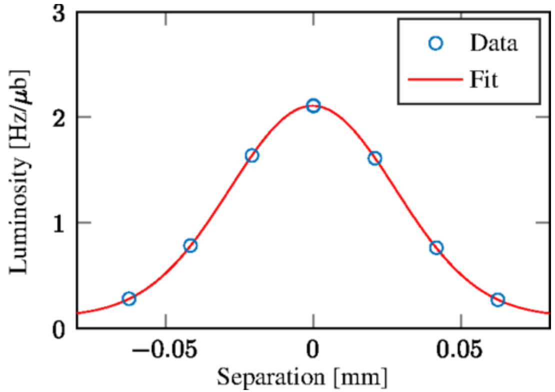
emittance scans - beam separations

- **beam separation scans ("mini-VdM")**
  - luminosity vs. separation fitted with Gaussian bunch-by-bunch
  - done regularly in LHC for emittances and tracking of luminosity monitor degradation



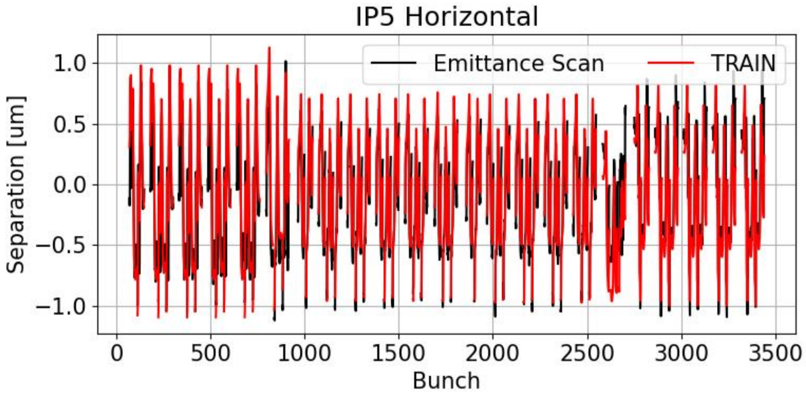
- **fit centre gives bunch-by-bunch beam separation**

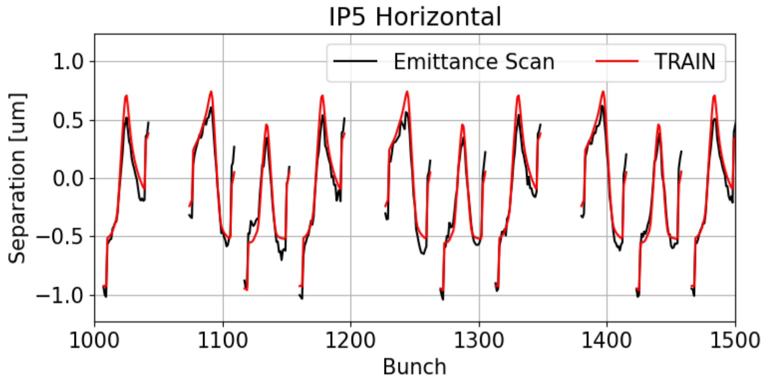


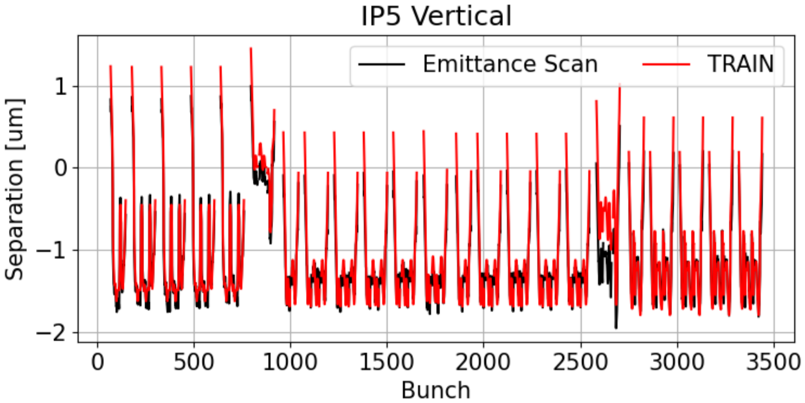


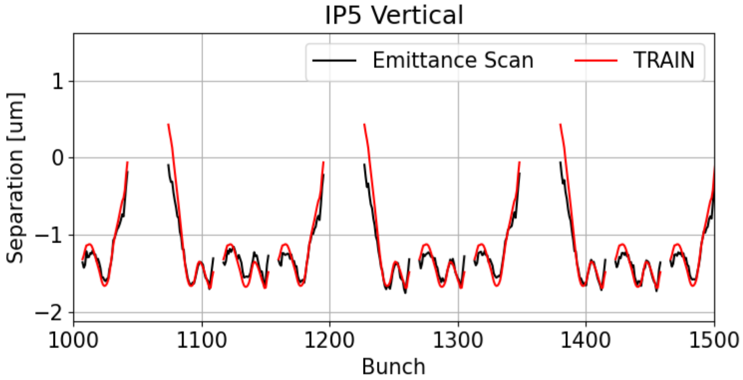
details in: M. Hostettler et al., "Luminosity scans for beam diagnostics", PRAB 21, 2018

emittance scans - beam separations





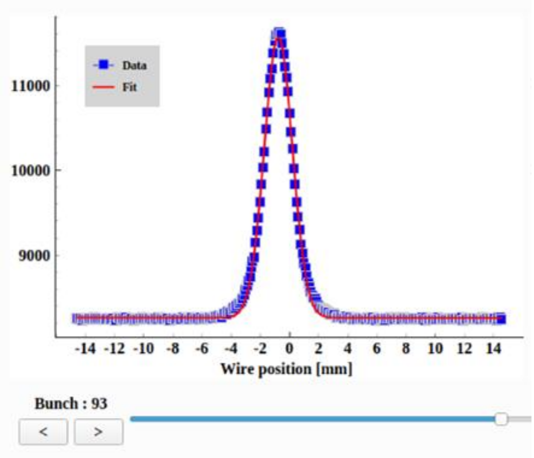


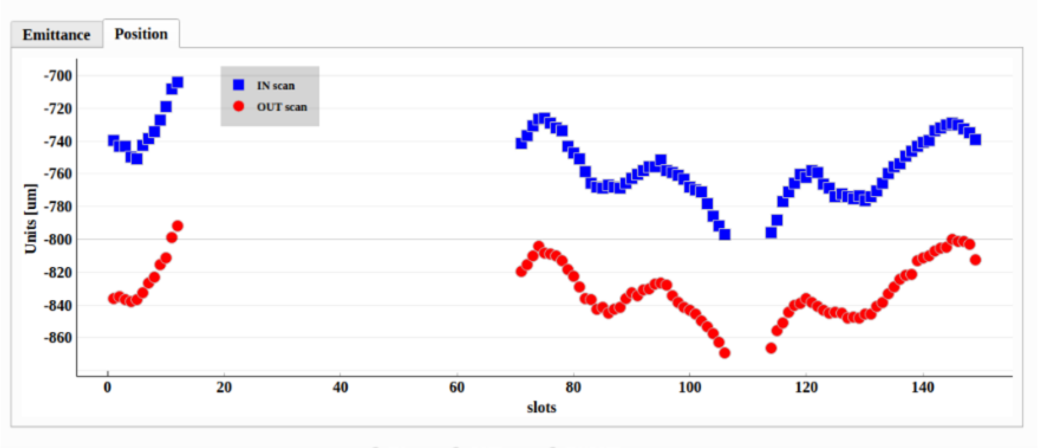


emittance scans in CMS, at beta=1.2m. Luminosity data courtesy of the CMS collaboration. LHC fill 10066, 2024-08-28*

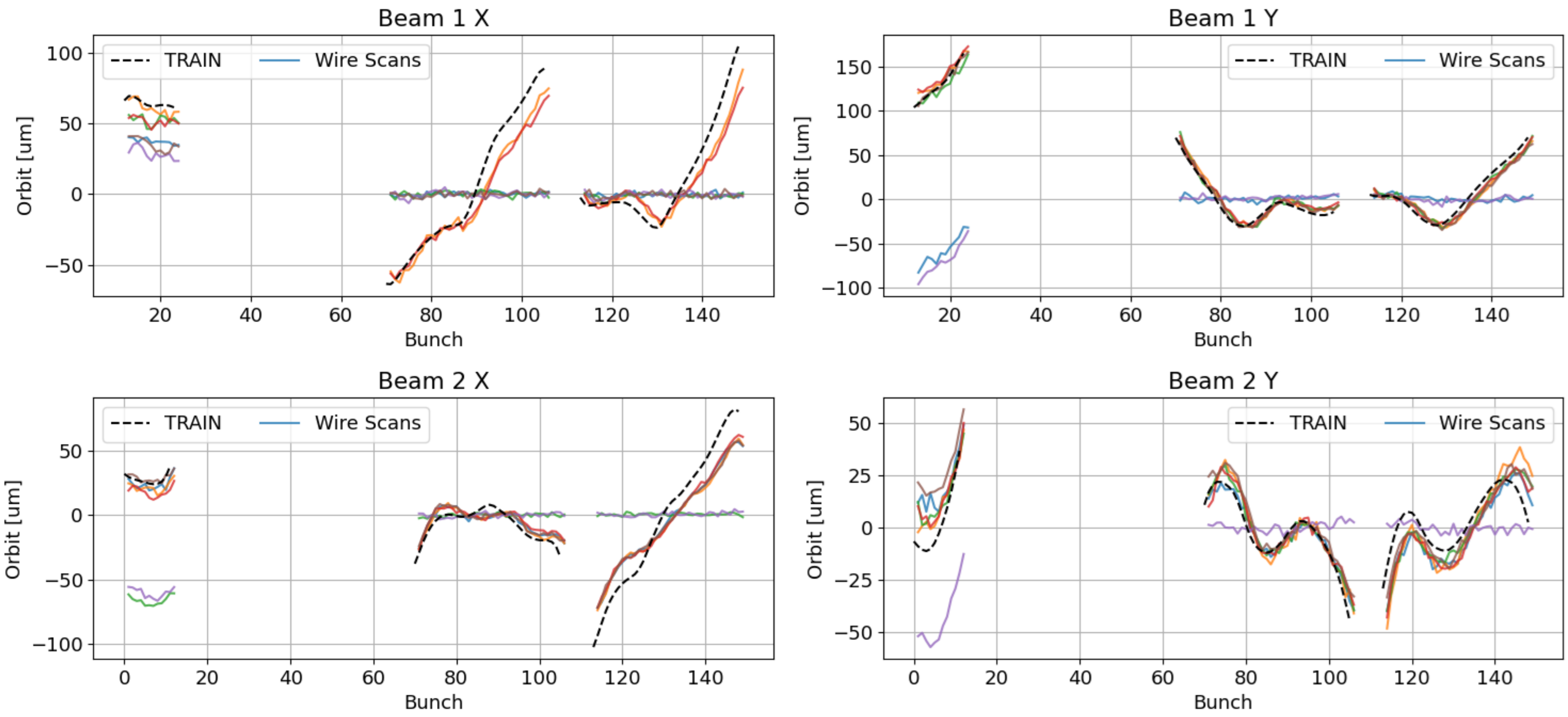
beam positions at wire scanners

- **wire scans regularly taken during LHC injection**
  - first 108 bunches only (scanner intensity limit)
  - bunch positions from centre of Gaussian fit
  - per-beam, per-plane, per-bunch data
- **no head-on collisions, but long-range encounters present!**



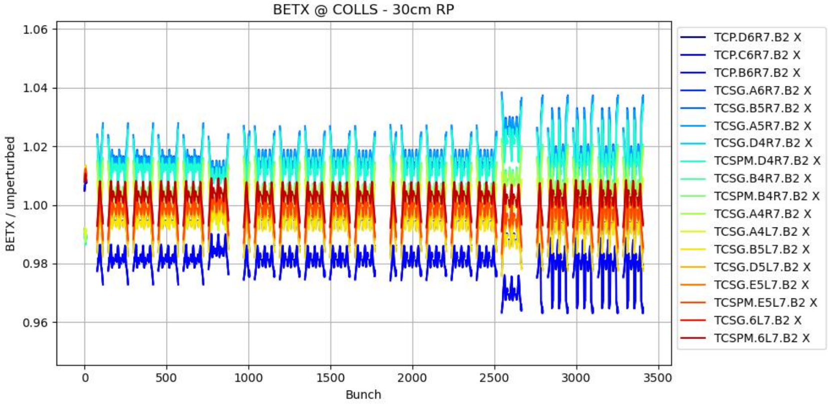


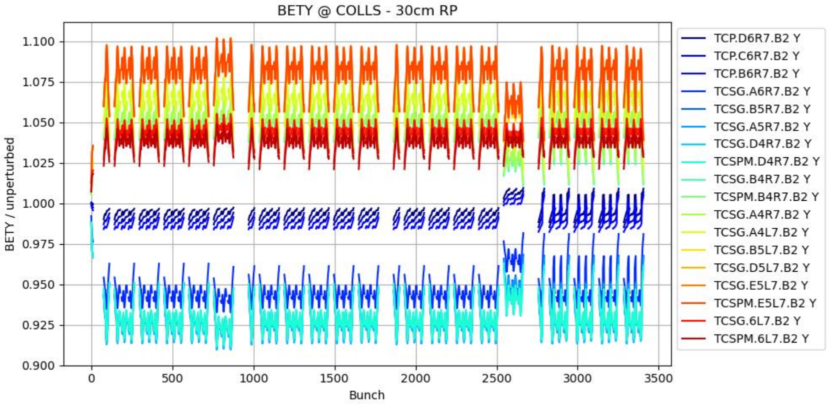
beam positions at wire scanners



from orbits to optics parameters

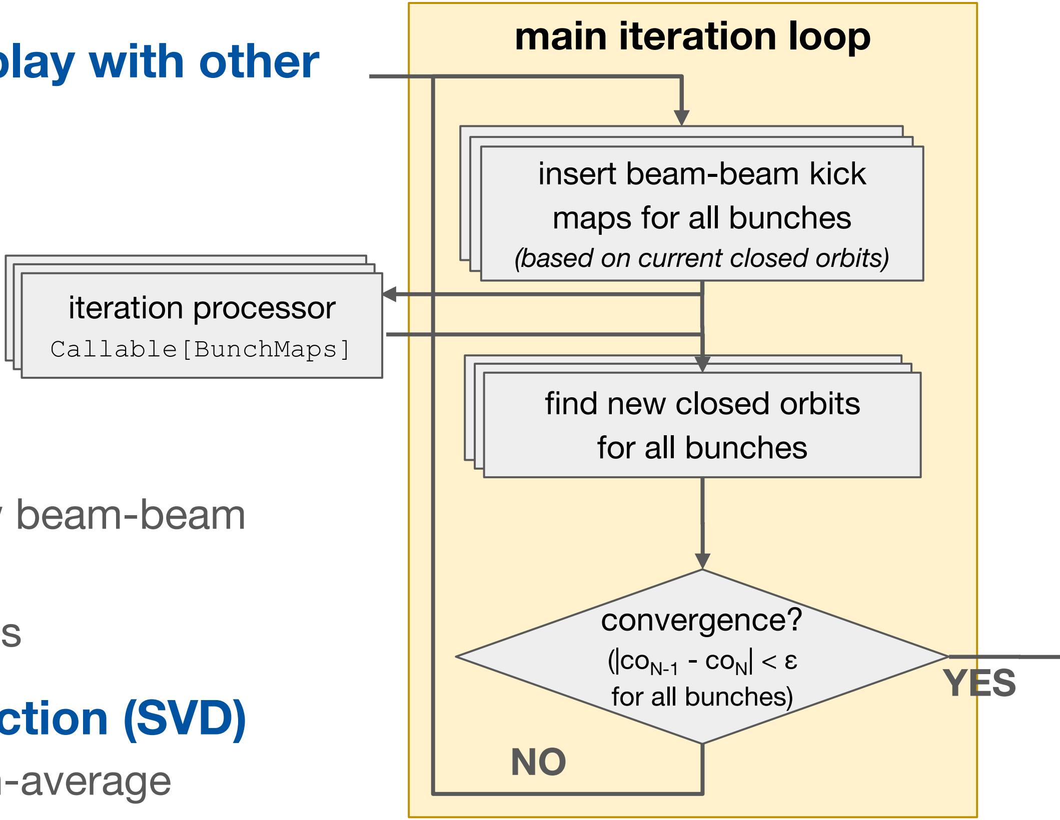
- **classic TRAIN output: orbits, tunes, chromaticities per bunch**
- **internal calculation based on second-order maps (per bunch)**
  - the maps contain all optics information up to second order!
  - ➜ calculate **twiss parameters & dispersion** - for any bunch, at any location
    - based on MAD-X code translated to python
    - coupling not yet treated - future improvement





iteration processors

- **beam-beam effects can interplay with other machine systems**
  - e.g. feedback corrections
  - self-consistent treatment needs to hook into TRAIN iteration loop
- **iteration processors**
  - callback per iteration after new beam-beam maps are established
  - can insert / mutate bunch maps
- **pre-defined: mean orbit correction (SVD)**
  - simulates the effect of a bunch-average feedback



conclusions

- **pyTRAIN: a reimplementation of TRAIN in modern python**
  - using numpy / scipy primitives where possible
  - interface to MAD-X via cpymad
  - scriptable from python
- **results look promising**
  - reproduces well the BBLR patterns observed in LHC
- **allows for novel features**
  - orbit anywhere in the ring (not just BB interaction points)
  - twiss parameters anywhere in the ring
  - iteration processors
  - extensible in the future!

future improvements

- **physics improvements**
  - finite bunch length effect on the effective beam size due to beam angles
    - A. Babaev, https://arxiv.org/abs/2104.02595
  - fully self-consistent beam sizes, iterating on perturbed twiss parameters
  - treatment of beam-beam introduced coupling
- **validation & application to other machines**
  - currently only tested on LHC - any collaborators welcome!
- **integration with Xsuite**
  - at least for input map generation
  - install beam-beam elements at final separation for tracking?
- **performance**
  - nice to have: clean and readable code has higher priority
  - possible synergy with Xsuite integration (fast primitives and data structures)