ESRF | The European Synchrotron

Collective Effects Development in PyAT

AT Workshop - 03/10/23

Lee Carver

Acknowledgements: Simon White

OVERVIEW

- **Introduction**
- **General PyAT developments**
- **Major new functionalities**
  - Multi-bunch, parallelised collective effects
  - New PassMethods
- **Future developments**

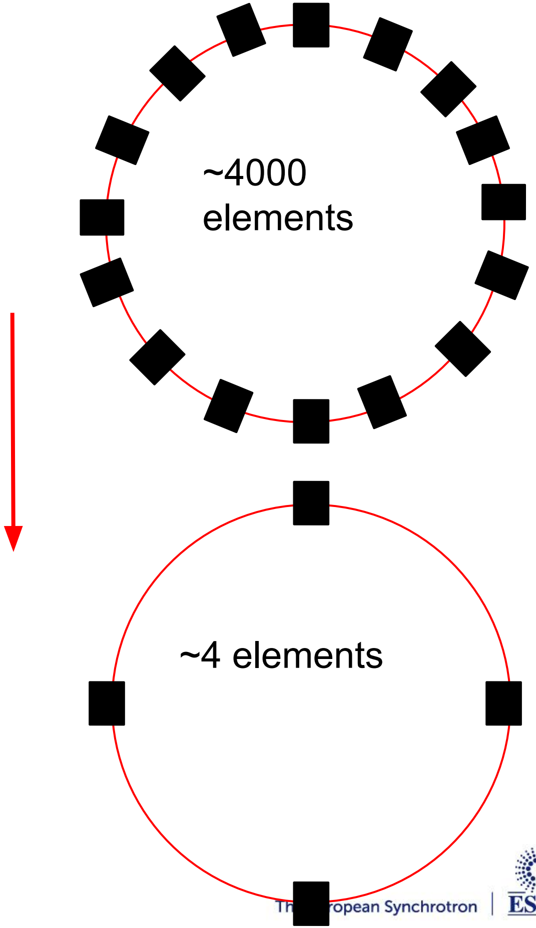
Lots more information at:
https://atcollab.github.io/at/p/howto/Collective.html

GENERAL PYAT DEVELOPMENTS

- **Convert 'atfastring' to python ('fast_ring').**
  - This function reduces the number of lattice elements from a full ring down to 4: an RF Cavity, a linear M66 matrix, a non linear element, and a quantum diffusion element.
  - This function is essential if you want to track very large numbers of particles.

- **New function added: 'simple_ring' ('atsimplering' in MatLab).**
  - The user provides a list of global machine parameters: energy, circumference, harmonic number, Qx, Qy, momentum compaction factor (other optional arguments possible).
  - Returns a simple lattice, similar to the output of 'fast_ring'.

- **It is now easier to generate a lattice with few elements that can easily be used for collective effects simulations.**

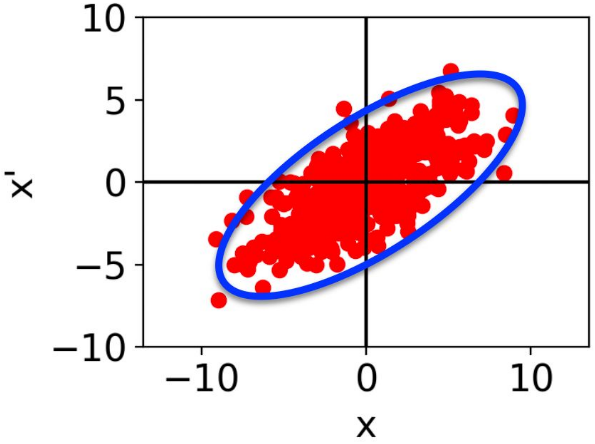


GENERAL PYAT DEVELOPMENTS

- **Beam generation functions were also added to PyAT**
  - atsigmamatrix → sigma_matrix
  - atbeam → beam

- **For sigma_matrix a variety of inputs can be provided:**
  - A lattice
  - A twiss_in
  - A list of lattice parameters
  - An array of shape (6,N) representing a beam distribution

- **For beam the input is a sigma_matrix and a number of particles.**
  - Same as MatLab



MULTIBUNCH

- **PyAT is now able to do multi-bunch simulations. Very simple to setup:**

  ring.set_fillpattern(arg), where arg can be:
    An integer saying number of symmetrically filled buckets
    An array of length harmonic number. The sum of this array must equal 1.

  ring.set_beam_current(total_current):
    Where total_current is in Amperes.

- **Once you set these parameters, other key attributes are set:**

  ring.nbunches
  ring.bunch_currents
  ring.bunch_spos

- **And you are all set! But how does it work?**

MULTIBUNCH

- **When you define your array of particles, you create one big array containing particles for all bunches. E.g.**

  Nbunches = 10
  Nparticles_per_bunch = 100
  Particles is an array of shape (6, 1000)

- **All collective effects PassMethods, the particles are accessed in the following way:**

[b#0p#0, b#1p#0, b#2p#0,... b#0p#1, b#1p#1, b#2p#1,...]

- **Or in python speak:**

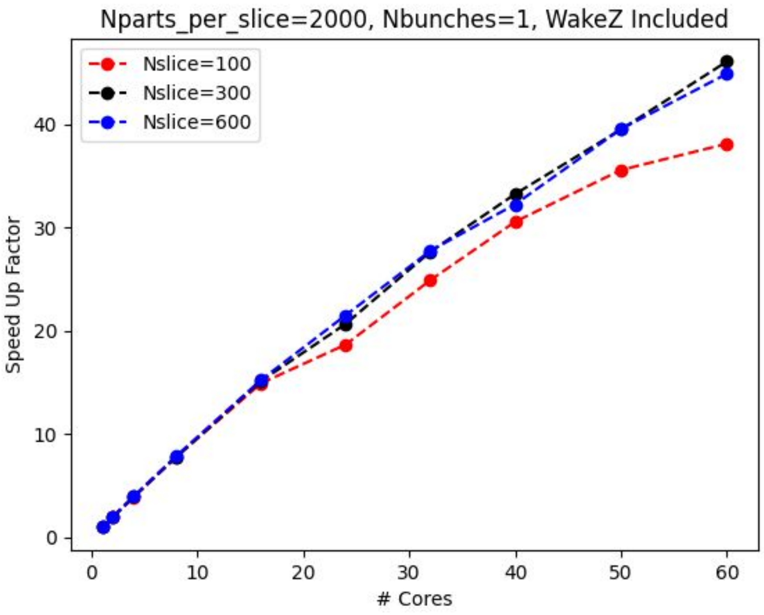
```
Particles[:, BunchNumber::Nbunches] #All particles of bunch BunchNumber
```

MULTIBUNCH

- **This gives flexibility for parallelisation.**
  - Each particle has a bunch index that is dependent on its position in the array and not on the mpi rank.

- **Parallelisation is coded using mpi4py.**
  - n instances of the same script are launched simultaneously.
  - Inside the collective effects passmethod, key information is exchanged (slice information, number of particles per slice) which allows each core to compute the correct kick

- **Significant speedup with many cores.**



Single bunch with longitudinal wake.
Parts per slice is fixed.

MULTIBUNCH

Pros:

- **You can easily split any number of bunches over any number of cores.**
    - E.g. 1 bunch over 50 cores or 50 bunches over 25 cores.

- **You can run multi-bunch simulations with 1 particle per bunch.**

- **The CPU load over all cores is even.**

Cons:

- **The number of particles in each thread must be an integer multiple of the number of bunches.**
    - E.g. 4 cores, 100 bunches. Must have at least 100 particles on each core. Otherwise a bucket may be considered empty!
    - 1 particle per bunch does not work for parallelised cases!

- **An extra step is required to unfold your array back into bunches.**

NEW PASSMETHODS

- **ImpedanceTablePass is now obsolete.**
  - It actually took a Wake Field as input.

- **We have a new pass method, WakeFieldPass.**
  - Supported by the new library, **atimplib**, which contains many useful functions that are used within the collective effects PassMethods.

- **A new structure of python functions and classes has been added to provide useful features for collective effects simulations.**
  - These will be introduced in the next slide.

- **A new beam monitor for multibunch simulations has also been added (BeamMoments).**
  - It computes means and standard deviations for all 6 planes bunch-by-bunch and turn-by-turn.
  - Slicing is currently being developed to be able to save the bunch-by-bunch distributions for a subset of turns and bunches.

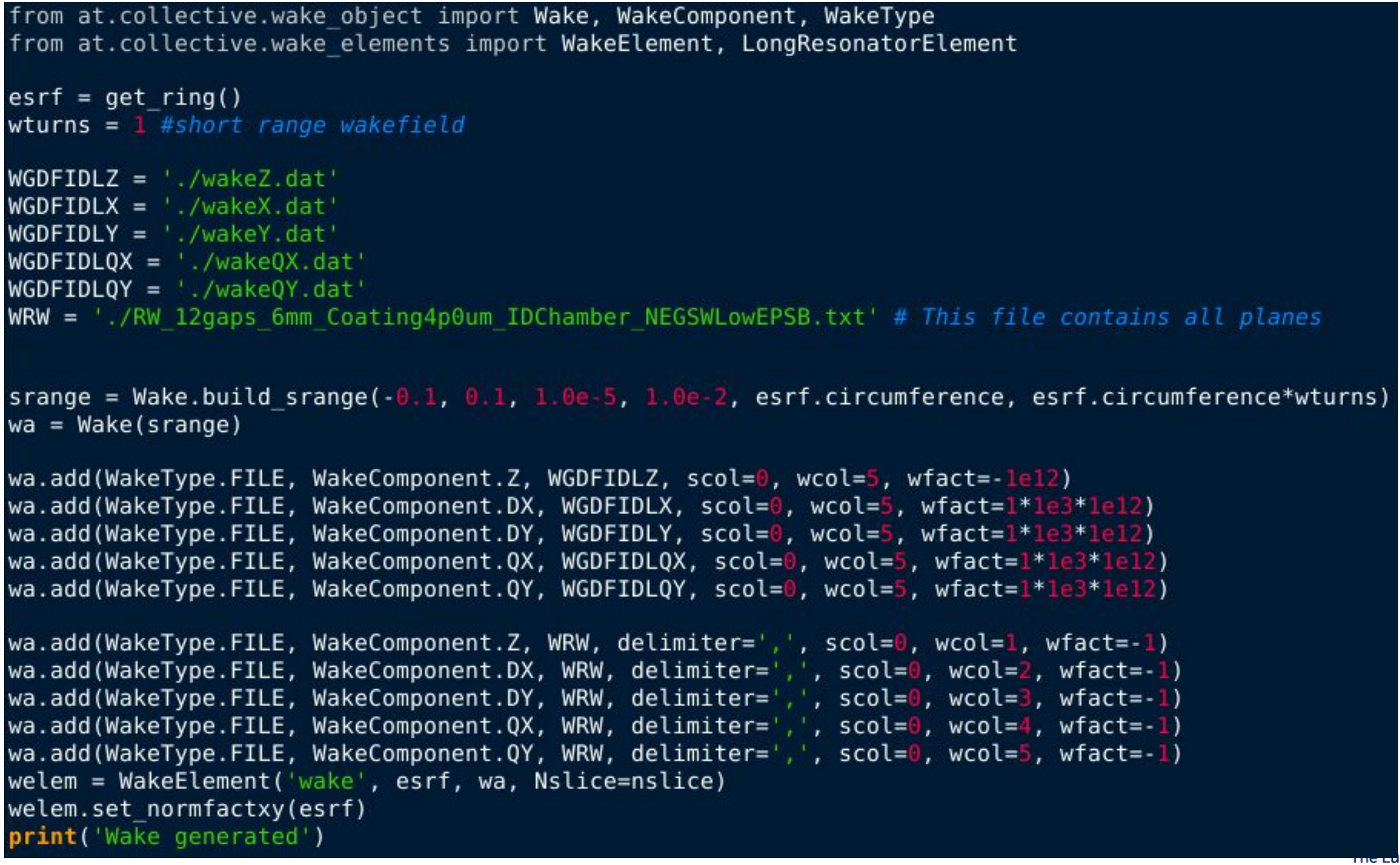
COLLECTIVE EFFECTS

- **In at/pyat/at/collective we have the full package of collective effects functions.**
    - wake_functions.py - contains the definitions for long_resonator, trans_resonator, trans_rw and also a function for convolution of an array with a gaussian

- **There are two main classes, one is called a wake_object.**
    - The wake can be built up with multiple wakes (combination of analytical and from file).
    - All wakes will use a common srange (interpolation will automatically be used).

- **The other is a wake_element.**
    - The wake_element is what is added to the lattice.
    - It can be created using a wake_object that the user has built up.
    - Many standard functions exist for easily creating the commonly used elements (ResWallElement, LongResonatorElement, TransResonatorElement)

COLLECTIVE EFFECTS

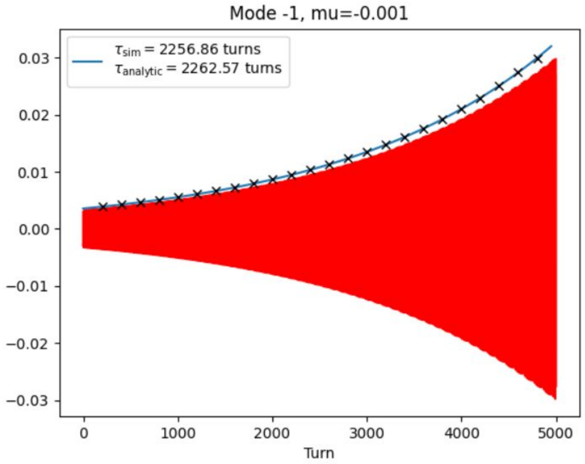


```
from at.collective.wake_object import Wake, WakeComponent, WakeType
from at.collective.wake_elements import WakeElement, LongResonatorElement

esrf = get_ring()
wturns = 1 #short range wakefield

WGDFIDLZ = './wakeZ.dat'
WGDFIDLX = './wakeX.dat'
WGDFIDLY = './wakeY.dat'
WGDFIDLQX = './wakeQX.dat'
WGDFIDLQY = './wakeQY.dat'
WRW = './RW_12gaps_6mm_Coating4p0um_IDChamber_NEGSWLowEPSB.txt' # This file contains all planes


srange = Wake.build_srange(-0.1, 0.1, 1.0e-5, 1.0e-2, esrf.circumference, esrf.circumference*wturns)
wa = Wake(srange)

wa.add(WakeType.FILE, WakeComponent.Z, WGDFIDLZ, scol=0, wcol=5, wfact=-1e12)
wa.add(WakeType.FILE, WakeComponent.DX, WGDFIDLX, scol=0, wcol=5, wfact=1*1e3*1e12)
wa.add(WakeType.FILE, WakeComponent.DY, WGDFIDLY, scol=0, wcol=5, wfact=1*1e3*1e12)
wa.add(WakeType.FILE, WakeComponent.QX, WGDFIDLQX, scol=0, wcol=5, wfact=1*1e3*1e12)
wa.add(WakeType.FILE, WakeComponent.QY, WGDFIDLQY, scol=0, wcol=5, wfact=1*1e3*1e12)

wa.add(WakeType.FILE, WakeComponent.Z, WRW, delimiter=',', scol=0, wcol=1, wfact=-1)
wa.add(WakeType.FILE, WakeComponent.DX, WRW, delimiter=',', scol=0, wcol=2, wfact=-1)
wa.add(WakeType.FILE, WakeComponent.DY, WRW, delimiter=',', scol=0, wcol=3, wfact=-1)
wa.add(WakeType.FILE, WakeComponent.QX, WRW, delimiter=',', scol=0, wcol=4, wfact=-1)
wa.add(WakeType.FILE, WakeComponent.QY, WRW, delimiter=',', scol=0, wcol=5, wfact=-1)
welem = WakeElement('wake', esrf, wa, Nslice=nslice)
welem.set_normfactxy(esrf)
print('Wake generated')
```

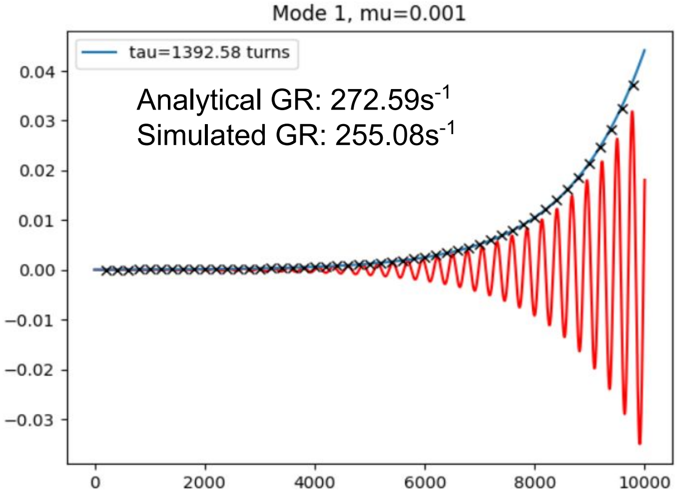
MULTIBUNCH INSTABILITIES

- **Some multibunch instability benchmarking has been made.**
- **We would like to make some more.**
- **Both of these cases can be found in: at/pyat/examples/CollectiveEffects**

Transverse Resistive Wall - MB



Longitudinal Coupled Bunch Instability - MB



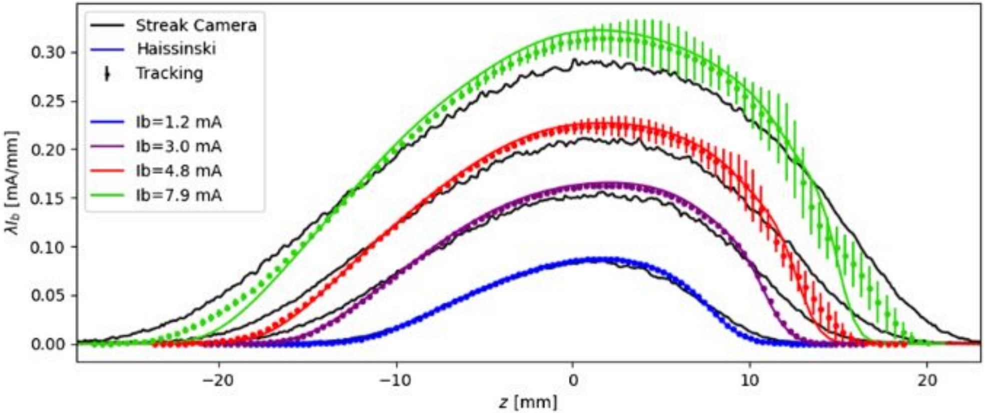
Analytical GR: 272.59s^{-1}
Simulated GR: 255.08s^{-1}

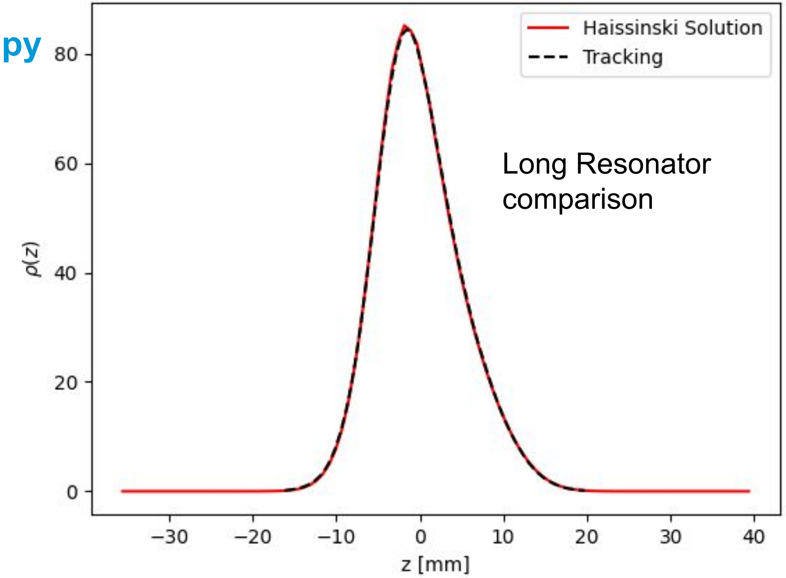
HAISSINSKI SOLVER

- **A short range Haissinski solver for arbitrary wakes has been implemented.**
  - Based on the algorithms developed by Warnock and Bane*.

- **Good agreement with tracking, comparisons with measurements made for the EBS*.**

- **An example file can be found in: at/pyat/examples/CollectiveEffects/LongDistribution.py**





Long Resonator comparison

*R. Warnock and K. Bane, ‘Numerical solution of the Haïssinski equation for the equilibrium state of a stored electron beam’, Phys. Rev. Accel. Beams 21, 124401

*L. R. Carver et al, ‘Beam based characterization of the European Synchrotron Radiation Facility Extremely Brilliant Source short range wakefield model’, Phys. Rev. Accel. Beams 26, 044402

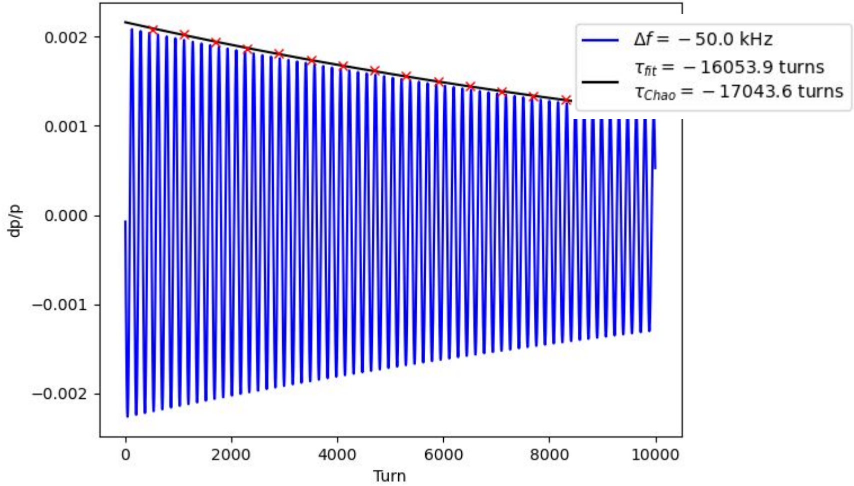
BEAMLOADING PASSMETHOD

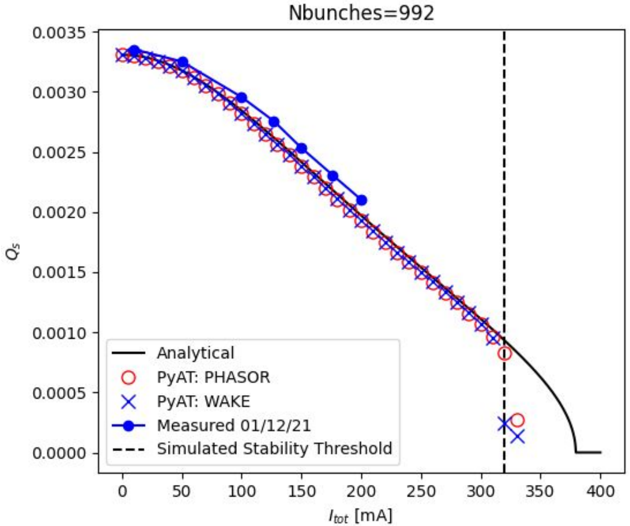
- **Active beam loading has been added.**
- **Beam is sliced longitudinally and the induced voltage is computed by summing voltage contribution from each slice.**

- **Two methods available**
    - PHASOR (where a running total is kept)
    - WAKE (where a turn history is kept and it is recomputed each time).

- **Generator voltage and phase is computed and applied (with a gain factor).**

Full details in IPAC23 proceedings (not yet released):
L.R. Carver et al, ‘Beam loading simulations in PyAT for the ESRF’





FUTURE DEVELOPMENTS

- **Harmonic cavity with passive beamloading has been implemented but not yet merged with the master.**
  - Planned benchmarking with SLS2 case in the pipeline. Once I can show it is working well, it will be merged.
  - Waiting for a few developments (BeamMoments slicing and SimpleRing [now merged]).

- **Possibility to generate a beam that is matched to the longitudinal bucket including impedance sources.**
  - Some work on this has started. Needs to be properly integrated and generalised.

- **Multi-bunch haissinski solvers**
  - I have a solver that approximately works but very far from being integrated into AT

- **Ion effects?**
  - Big project.

- **Improved unit testing of multi-bunch tracking and collective effects**
  - Example files are not sufficient.
  - MPI tests are possible?