CERN

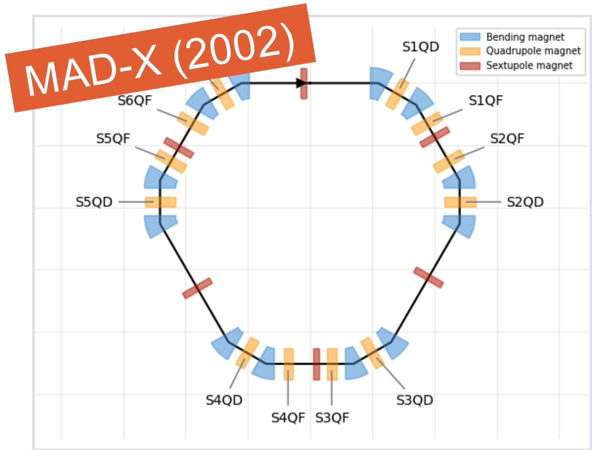
Xsuite:
A Modular Accelerator Simulation Framework

Riccardo De Maria, Giovanni Iadarola, Szymon Łopaciuk, Frederik F. Van der Veken, CERN, Geneva, Switzerland

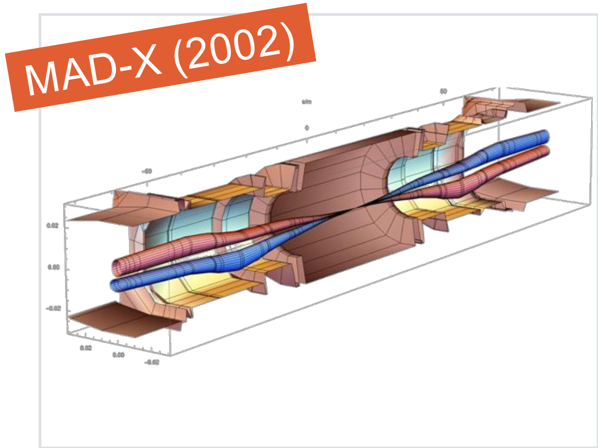
We sincerely thank the Xsuite users and contributors for their enthusiasm and invaluable input.

Beam dynamics software landscape at CERN

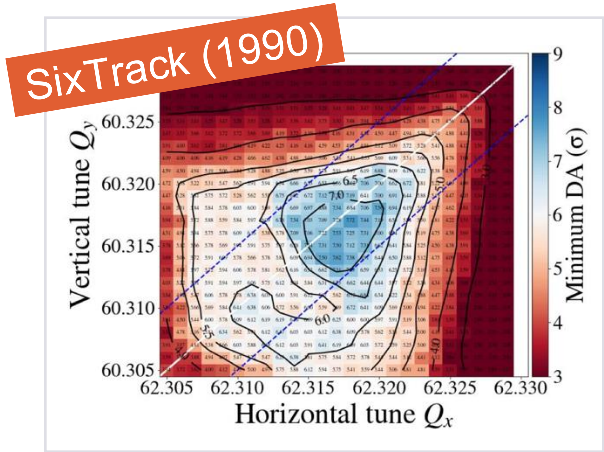
CERN has a long history of **powerful software tools for beam physics** applications, typical examples:



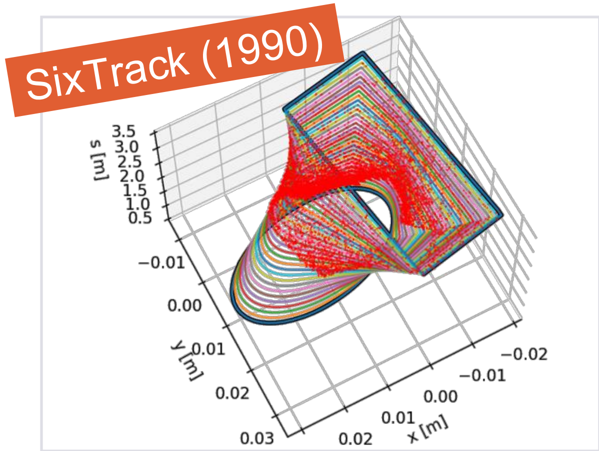
Lattice Design



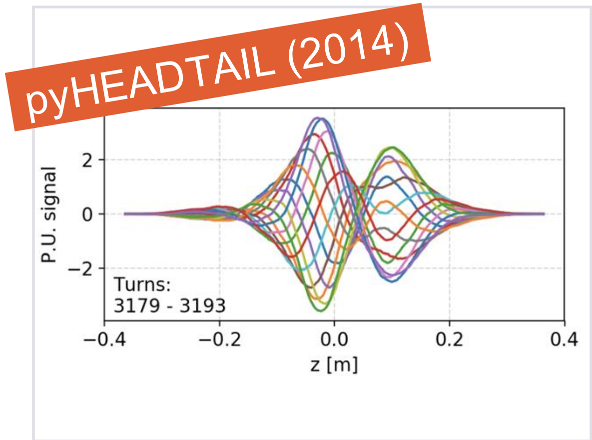
Optics (calculation & design)



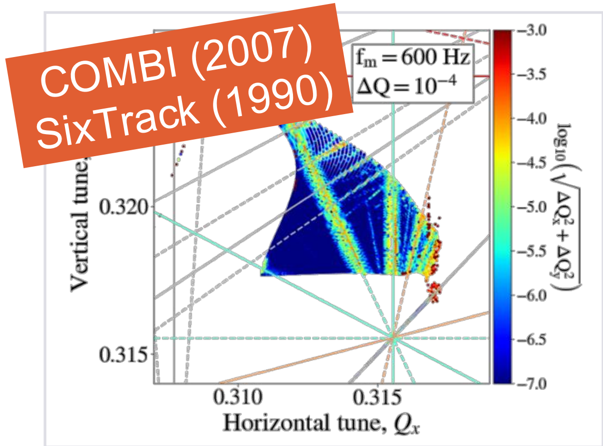
Tracking (dynamic aperture)



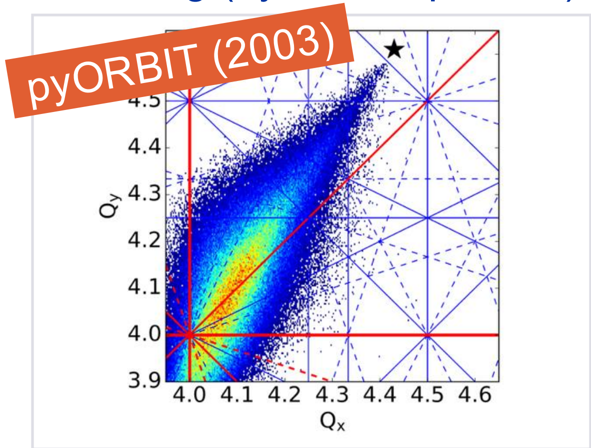
Collimation



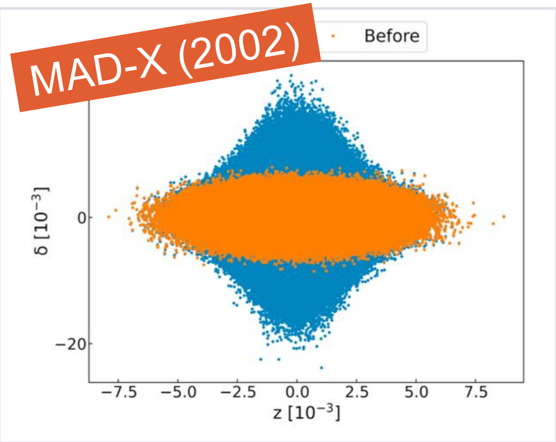
Impedances



Beam-Beam Effects



Space Charge



Intra-Beam Scattering

Requirements:

– One toolkit for many applications: from low-energy hadron rings to high-energy lepton colliders.

– Heterogeneous simulations made natural, as opposed to having to deal with ad-hoc scripts or model translations between tools.

– Extendable by default, while legacy tools are much harder to extend. They weren't designed for that. Lack of expertise to do so anymore.

– Modern user-interface: Python, with its ecosystem of scientific computing tools. No need to maintain own plotting tools, or scripting languages anymore – saves effort, less complexity.

– GPU support built-in, in addition to single- and multi-threaded CPU.

Xsuite project launched in 2021 to address issues arising from the fragmented landscape, using the know-how acquired in development of the earlier tools.

Goals: 1st class Python, 1st class collective, 1st class GPU.

References for a full list of features:

- *Documentation: xsuite.web.cern.ch.*
- *"Xsuite: a multiplatform toolbox for optics design, fast tracking, collimation and collective effects," ICAP'24.*
- *"Xsuite: a multiplatform Python toolkit for beam dynamics," ATS Seminar (2025).*
- *"Empowering a broad and diverse community in beam dynamics simulations with Xsuite." IPAC'25.*

Development approach

– **Orthogonal architecture:** split the software into independent functional blocks at all levels.

  ⇒ Well-defined, cleaner interfaces

  ⇒ Lower codebase complexity

  ⇒ Better scaling, easier to understand

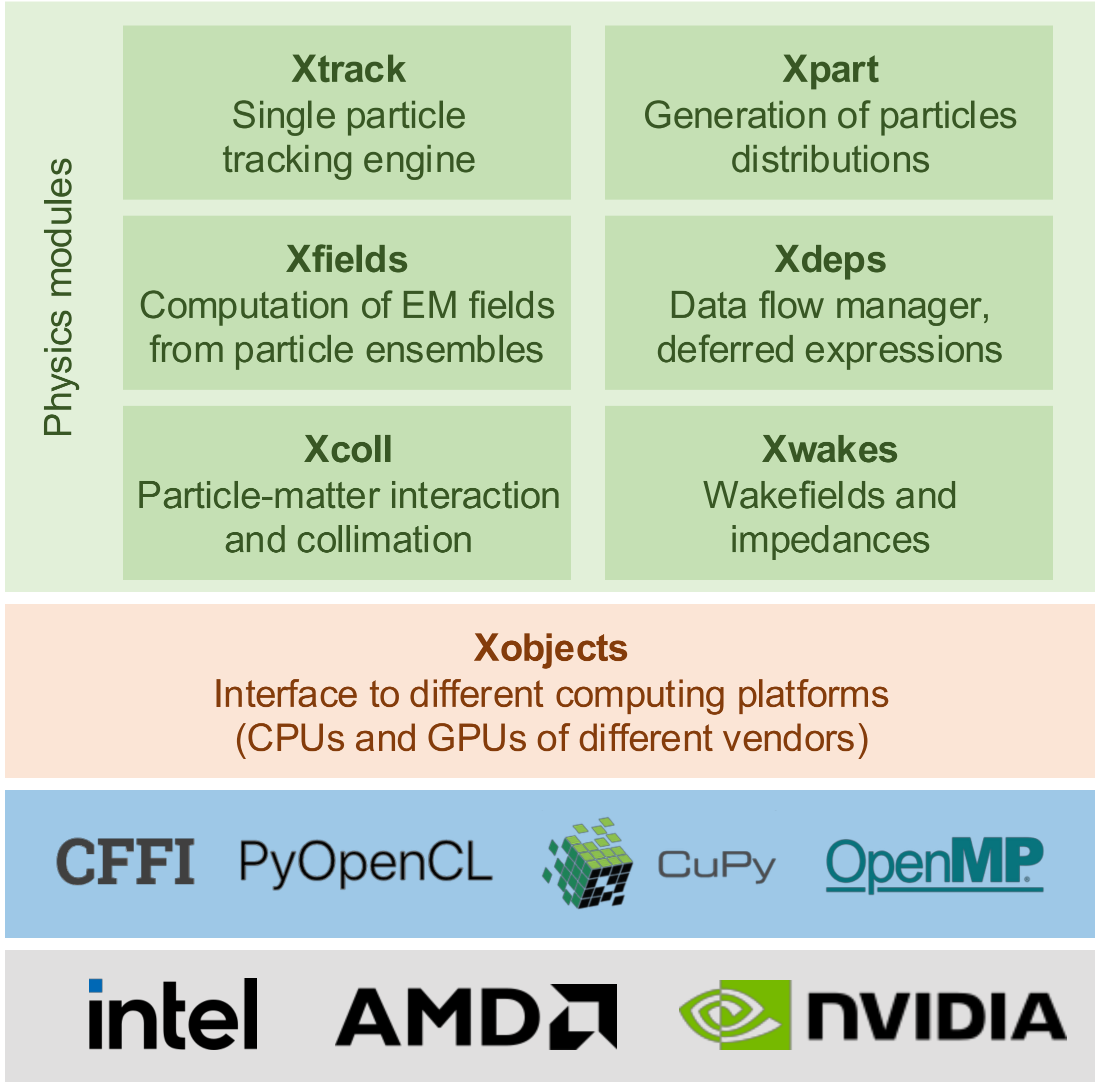
  ⇒ Lower learning curve for users & developers

  ⇒ Users can (and do!) become developers.

– **Agile development:**

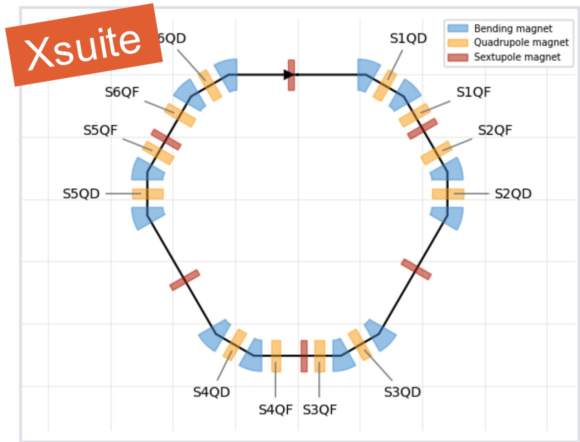
  – From the beginning big effort to support users: user feedback and involvement visibly increases the quality of the package.

  – Thanks to investment in continuous testing, we can afford a fast release cycle with new versions coming out multiple times a month, incorporating modifications/extensions based on user feedback.

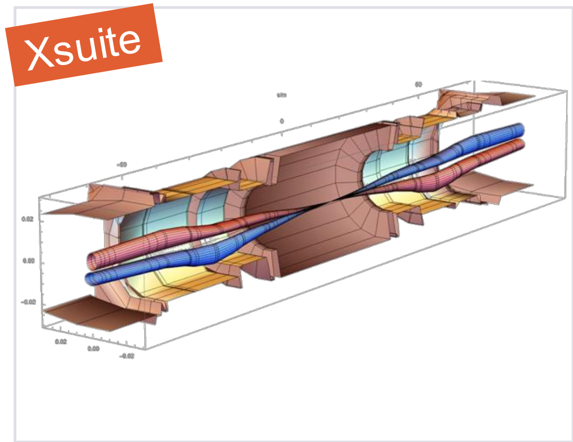


Adoption

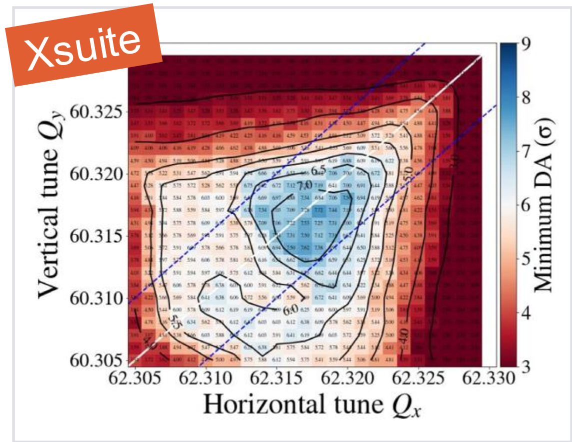
By now Xsuite became a production tool for many beam dynamics studies, allowing development discontinuation for many legacy tools.



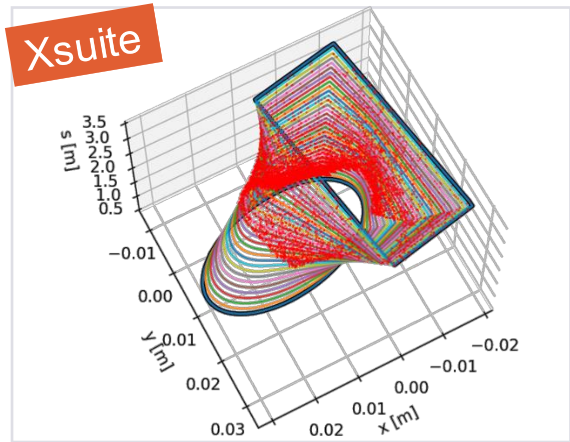
Lattice Design



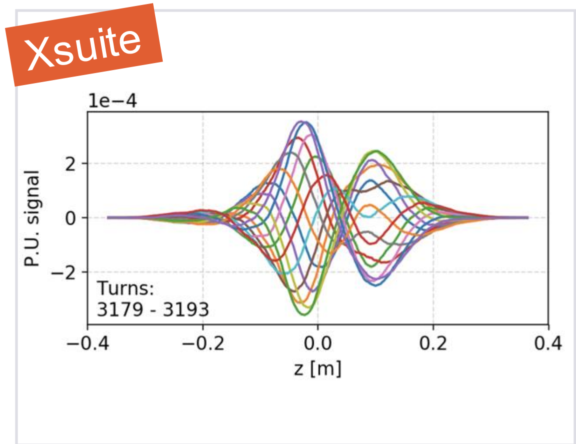
Optics (calculation & design)



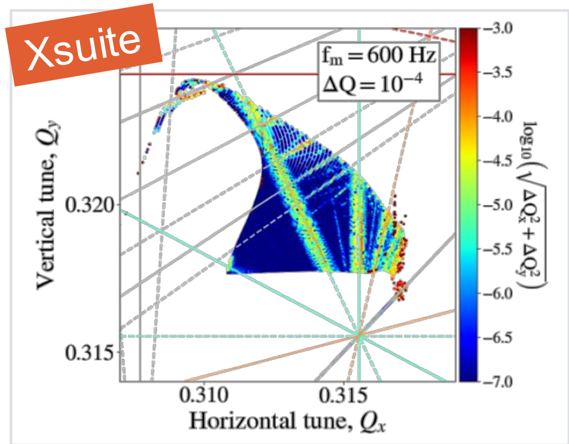
Tracking (dynamic aperture)



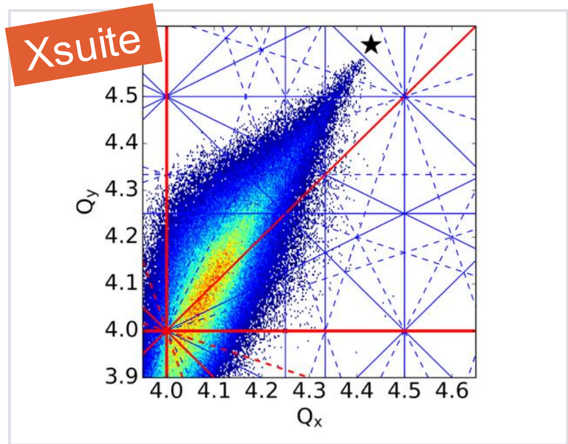
Collimation



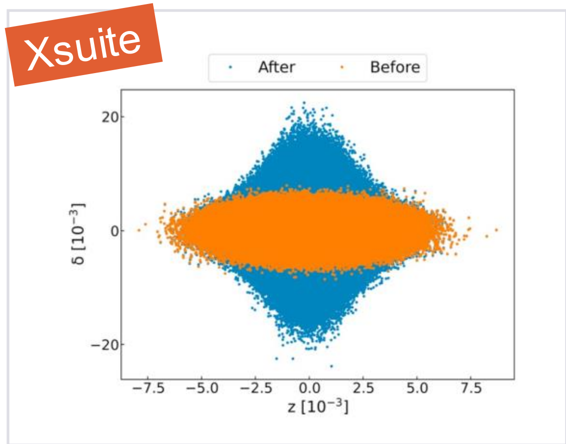
Impedances



Beam-Beam Effects



Space Charge



Intra-Beam Scattering

Lively user base

Users' and developers' response well beyond our expectations!

>30 contributors
>150 active users worldwide
>10% of all IPAC25 proceedings mention Xsuite
Accelerator schools (USPAS, CAS, JUAS) now use Xsuite in tutorials.

A lively community providing mutual support, advice, and lots of feedback to developers!

CERN
– AD
– ELENA
– LEIR
– PSB
– PS
– SPS, TI2, TI8
– LHC
– FCC-ee, FCC-hh
– Muon Collider

Fermilab
– Main injector
– Recycler
– Booster
– IOTA

GSI
– SIS-18
– SIS-100

BNL
– RHIC
– Booster
– EIC

J-PARC
– Main Ring
– KEK
– SuperKEKB

Medical facilities
– HIT (Heidelberg)
– MEDAUSTRON
– PIMMS
– NIMMS

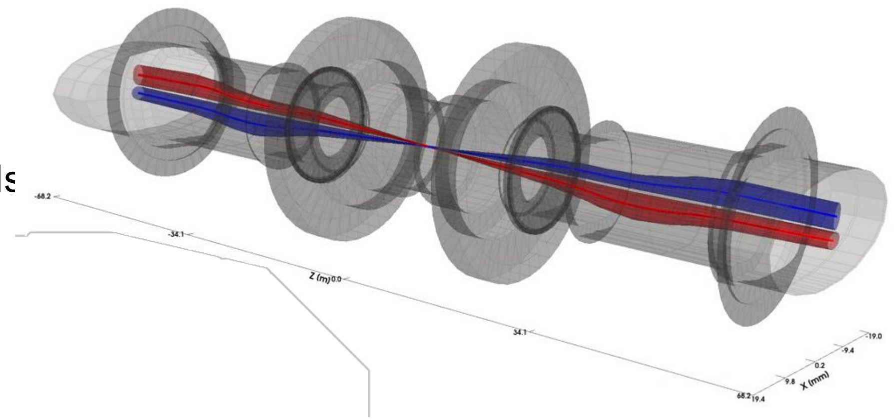
Light sources/damping rings:
– PETRA
– DESY injector ring
– ELETTRA
– BESSY III
– PSI SLS 2.0
– Canadian Light Source
– CLIC-DR

and more...

Xsuite capabilities overview

- **Lattice design, optics design and calculation**
- **Field imperfections and misalignments**
- **Fringe fields, combined function-magnets, solenoids** with overlapping multipolar fields
- **Orbit and trajectory correction**
- MAD-NG interface: compute **non-linear maps, normal forms, RDTs**
- RF-Track interface: track through **field maps**

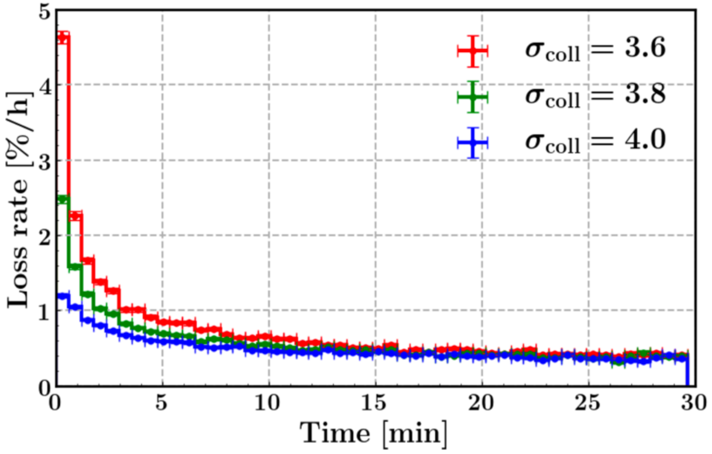
LHC Beam Envelopes at IP1 (Atlas) (3σ, β*=30cm)



- **Particle-matter interaction** (internal Everest engine + interfaces to FLUKA & GEANT4)
- **Synchrotron Radiation:** mean & stochastic models
- **Spin tracking and polarisation estimates**

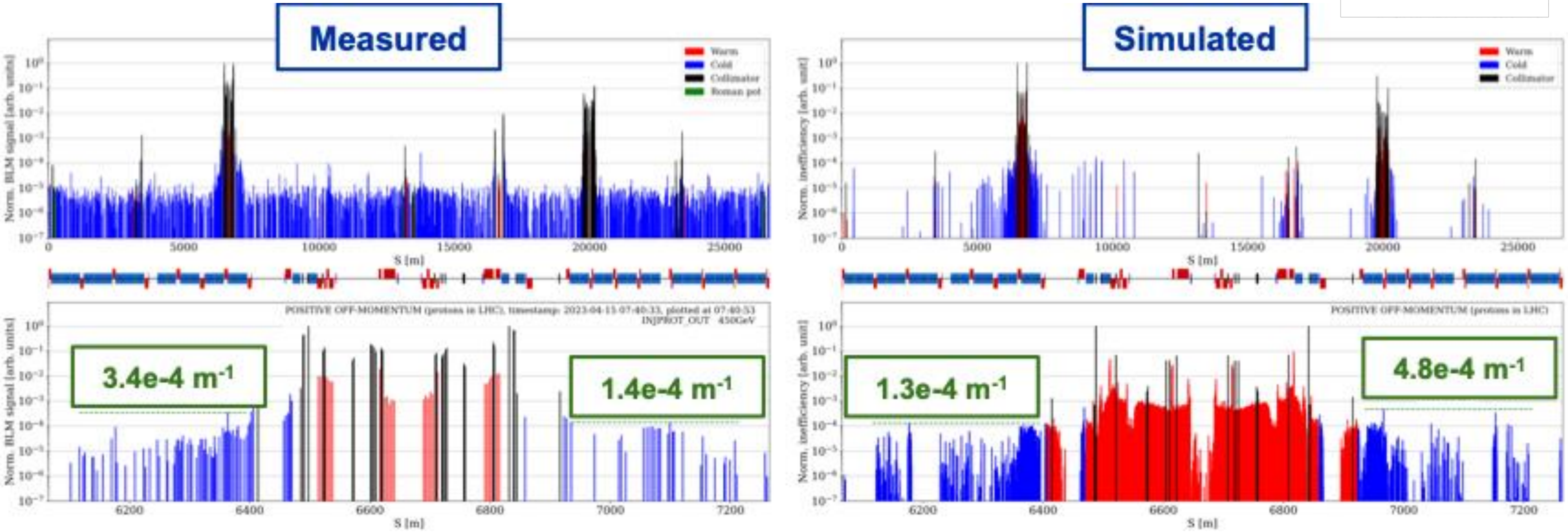
- **Space charge:** frozen and self-consistent PIC
- **Wakefields:** di- and quadrupolar, single- & multi bunch, multi-turn effects
- **Beam-beam effects:** weak- & strong-strong, analytic or PIC (new!), Beamstrahlung, Bhabha scattering
- **Intra-Beam Scattering**

Element-by-element simulation of the full LHC for 20 **M turns**:
under 3 days simulation time on GPUs



Example: Off-momentum loss map in the LHC

- **Loss maps** are performed regularly in the LHC to assess **the efficiency of the collimation system**
- **Off-momentum** loss maps are a special type, where the frequency of the RF cavities is swept
- Outcome is a **longitudinal distribution of losses** along the ring
- The simulation needs to **combine many things:** particle tracking through the lattice, while the RF frequency is **dynamically adapted**, taking care of **particle-matter interactions** whenever a particle hits a collimator



Demo 1 – Lattice Design

In addition to being feature-complete, Xsuite's major benefit is its user-friendly interface.

To showcase this, we have prepared a set of demos.

These demo files are adapted from https://github.com/xsuite/tutorial_cern_seminar and are based on the PIMMS lattice (see CERN-PS-99-010-DI).

Lattice design with Xsuite

We will uses as example the PIMM lattice developed by the TERA collaboration for proton and ion therapy (CERN/PS 99-010) and implemented in the CNAO and MEDAUSTRON synchrotrons.

[ ]:
```
import matplotlib.pyplot as plt
import xtrack as xt
import numpy as np
```

Build Xsuite environment and define a reference particle

[ ]:
```
env = xt.Environment()
env.particle_ref = xt.Particles(kinetic_energy0=200e6)
env.vars.default_to_zero = True # Undefined variables in env are automatically
                                # set to zero.
```

Define elements

[ ]:
```
# Element geometry
n_bends = 16
env['ang_mb'] = 2 * np.pi/n_bends
env['l_mb'] = 1.65
env['l_mq'] = 0.35

env.new('mb', xt.RBend, length='l_mb', angle='ang_mb', k0_from_h=True)
env.new('mq', xt.Quadrupole, length='l_mq');
```

[ ]:
```
# Quadrupole families with different strengths
env.new('qfa', 'mq', k1= 'kqfa')
env.new('qfb', 'mq', k1= 'kqfb')
env.new('qd',  'mq', k1= 'kqd');
```

Lattice design with Xsuite

We will uses as example the PIMM lattice developed by the TERA collaboration for proton and ion therapy (CERN/PS 99-010) and implemented in the CNAO and MEDAUSTRON synchrotrons.

[1]:
```
import matplotlib.pyplot as plt
import xtrack as xt
import numpy as np
```

Build Xsuite environment and define a reference particle

[ ]:
```
env = xt.Environment()
env.particle_ref = xt.Particles(kinetic_energy0=200e6)
env.vars.default_to_zero = True # Undefined variables in env are automatically
                                # set to zero.
```

Define elements

[ ]:
```
# Element geometry
n_bends = 16
env['ang_mb'] = 2 * np.pi/n_bends
env['l_mb'] = 1.65
env['l_mq'] = 0.35

env.new('mb', xt.RBend, length='l_mb', angle='ang_mb', k0_from_h=True)
env.new('mq', xt.Quadrupole, length='l_mq');
```

[ ]:
```
# Quadrupole families with different strengths
env.new('qfa', 'mq', k1= 'kqfa')
env.new('qfb', 'mq', k1= 'kqfb')
env.new('qd',  'mq', k1= 'kqd');
```

Lattice design with Xsuite

We will uses as example the PIMM lattice developed by the TERA collaboration for proton and ion therapy (CERN/PS 99-010) and implemented in the CNAO and MEDAUSTRON synchrotrons.

[1]:
```
import matplotlib.pyplot as plt
import xtrack as xt
import numpy as np
```

Build Xsuite environment and define a reference particle

[2]:
```
env = xt.Environment()
env.particle_ref = xt.Particles(kinetic_energy0=200e6)
env.vars.default_to_zero = True # Undefined variables in env are automatically
                                # set to zero.
```

Define elements

[ ]:
```
# Element geometry
n_bends = 16
env['ang_mb'] = 2 * np.pi/n_bends
env['l_mb'] = 1.65
env['l_mq'] = 0.35

env.new('mb', xt.RBend, length='l_mb', angle='ang_mb', k0_from_h=True)
env.new('mq', xt.Quadrupole, length='l_mq');
```

[ ]:
```
# Quadrupole families with different strengths
env.new('qfa', 'mq', k1= 'kqfa')
env.new('qfb', 'mq', k1= 'kqfb')
env.new('qd',  'mq', k1= 'kqd');
```

Lattice design with Xsuite

We will uses as example the PIMM lattice developed by the TERA collaboration for proton and ion therapy (CERN/PS 99-010) and implemented in the CNAO and MEDAUSTRON synchrotrons.

[1]:
```
import matplotlib.pyplot as plt
import xtrack as xt
import numpy as np
```

Build Xsuite environment and define a reference particle

[2]:
```
env = xt.Environment()
env.particle_ref = xt.Particles(kinetic_energy0=200e6)
env.vars.default_to_zero = True # Undefined variables in env are automatically
                                # set to zero.
```

Define elements

[3]:
```
# Element geometry
n_bends = 16
env['ang_mb'] = 2 * np.pi/n_bends
env['l_mb'] = 1.65
env['l_mq'] = 0.35

env.new('mb', xt.RBend, length='l_mb', angle='ang_mb', k0_from_h=True)
env.new('mq', xt.Quadrupole, length='l_mq');
```

[ ]:
```
# Quadrupole families with different strengths
env.new('qfa', 'mq', k1= 'kqfa')
env.new('qfb', 'mq', k1= 'kqfb')
env.new('qd',  'mq', k1= 'kqd');
```

Build lattice cells

```
cell_a = env.new_line(length=7.405, components=[
    env.place('qfa', at=0.3875),
    env.place('mb', at=1.8125),
    env.place('qd', at=3.2925),
    env.place('mb', at=5.0475),
    env.place('qfa', at=6.3275),
])
cell_a.survey().plot()
```

```
cell_b = env.new_line(name='cell_b', length=8.405, components=[
    env.place('qfb', at=1.2725),
    env.place('mb', at= 2.7275),
    env.place('qd', at=4.8575),
    env.place('mb', at=6.5125),
    env.place('qfb', at=7.7925),
])
cell_b.survey().plot()
```

Build an arc

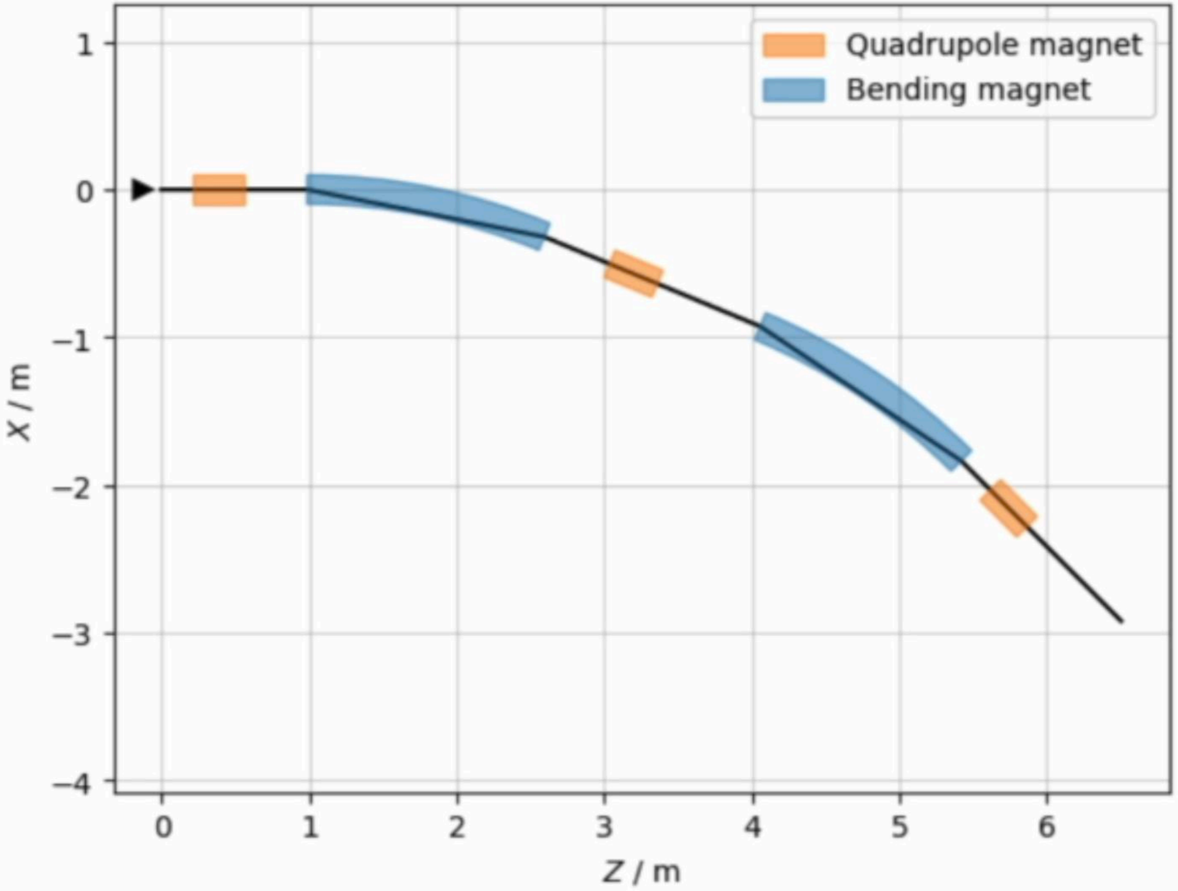
```
# Concatenate the two cells
arc = cell_a + cell_b
arc.survey().plot()
```

Build straight sections

```
long_straight = env.new_line(length=2., components=[
    env.new('mid.lss', xt.Marker, at=1.)
])
short_straight = env.new_line(length=1., components=[
    env.new('mid.sss', xt.Marker, at=0.5)
])
```

Build lattice cells

[5]:
```
cell_a = env.new_line(length=7.405, components=[
    env.place('qfa', at=0.3875),
    env.place('mb', at=1.8125),
    env.place('qd', at=3.2925),
    env.place('mb', at=5.0475),
    env.place('qfa', at=6.3275),
])
cell_a.survey().plot()
```



[ ]:
```
cell_b = env.new_line(name='cell_b', length=8.405, components=[
    env.place('qfb', at=1.2725),
```

```
[ ]: cell_b = env.new_line(name='cell_b', length=8.405, components=[
         env.place('qfb', at=1.2725),
         env.place('mb', at= 2.7275),
         env.place('qd', at=4.8575),
         env.place('mb', at=6.5125),
         env.place('qfb', at=7.7925),
     ])
     cell_b.survey().plot()
```

Build an arc

```
[ ]: # Concatenate the two cells
     arc = cell_a + cell_b
     arc.survey().plot()
```

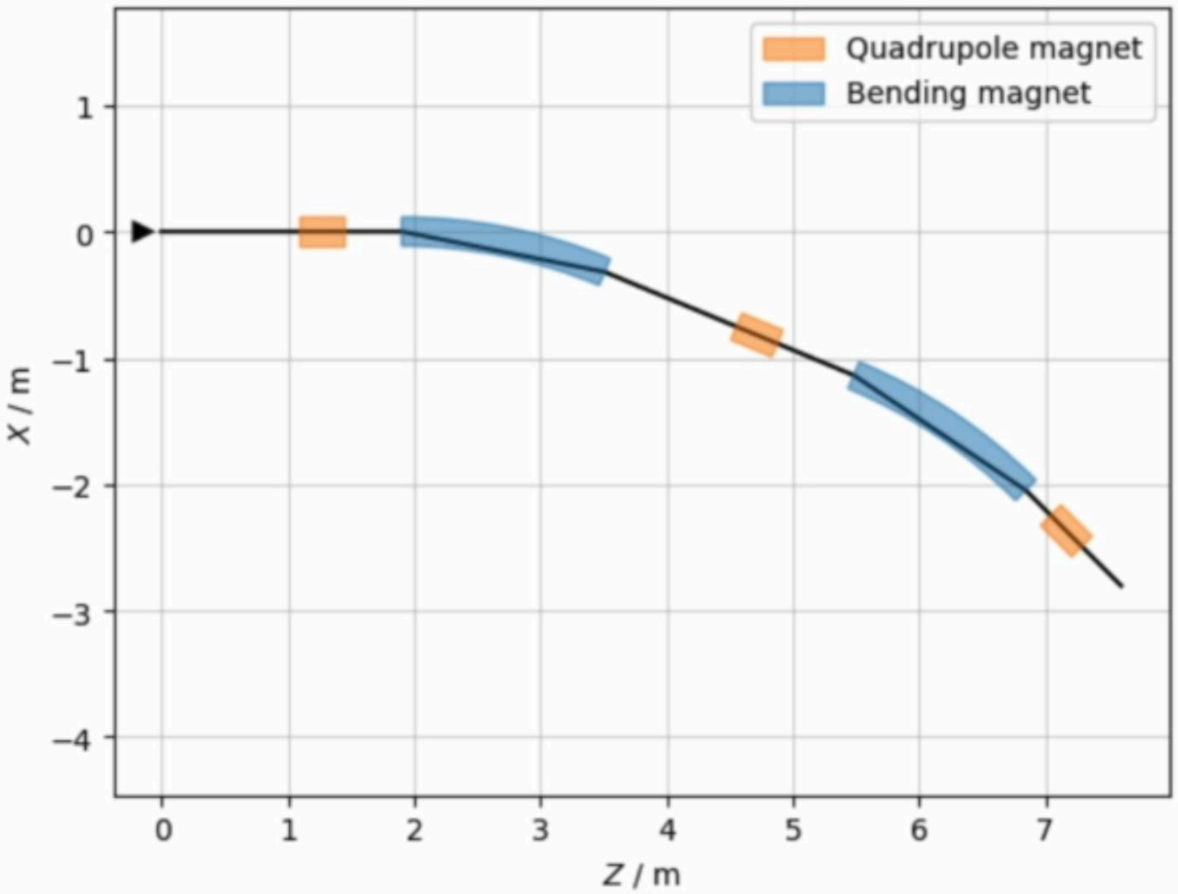
Build straight sections

```
[ ]: long_straight = env.new_line(length=2., components=[
         env.new('mid.lss', xt.Marker, at=1.)
     ])
     short_straight = env.new_line(length=1., components=[
         env.new('mid.sss', xt.Marker, at=0.5)
     ])
```

Assemble the ring

```
[ ]: half_ring = (
         long_straight
         + arc
         + short_straight
         - arc # mirror symmetric lattice
     )
     half_ring.survey().plot()
```

[6]:
```python
cell_b = env.new_line(name='cell_b', length=8.405, components=[
    env.place('qfb', at=1.2725),
    env.place('mb', at= 2.7275),
    env.place('qd', at=4.8575),
    env.place('mb', at=6.5125),
    env.place('qfb', at=7.7925),
])
cell_b.survey().plot()
```



Build an arc

Build an arc

```
[ ]: # Concatenate the two cells
arc = cell_a + cell_b
arc.survey().plot()
```

Build straight sections

```
[ ]: long_straight = env.new_line(length=2., components=[
         env.new('mid.lss', xt.Marker, at=1.)
     ])
     short_straight = env.new_line(length=1., components=[
         env.new('mid.sss', xt.Marker, at=0.5)
     ])
```

Assemble the ring

```
[ ]: half_ring = (
         long_straight
         + arc
         + short_straight
         - arc # mirror symmetric lattice
     )
     half_ring.survey().plot()
```

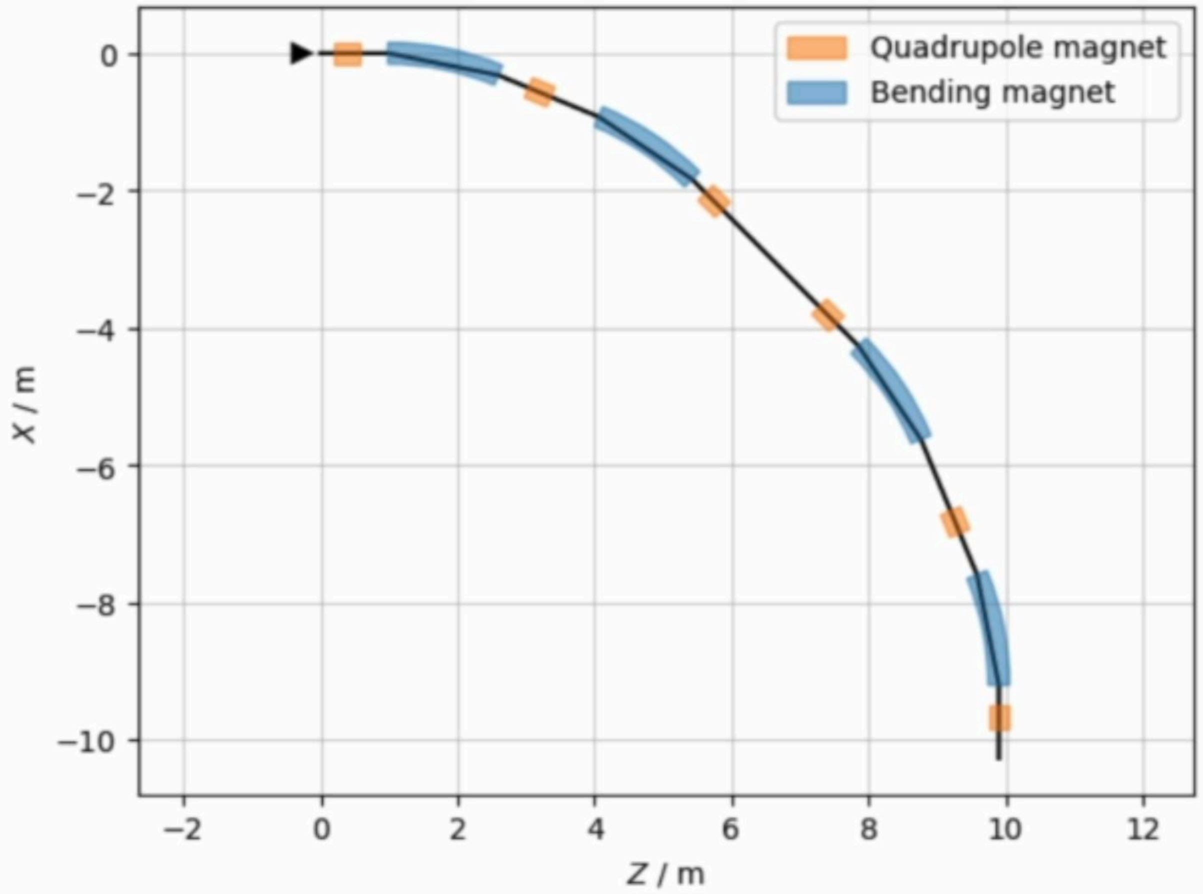
```
[ ]: ring = 2 * half_ring
     ring.survey().plot()
```

Replace repeated elements

```
[ ]: ring.replace_all_repeated_elements() # give all elements unique names
```

Build an arc

[7]:
```python
# Concatenate the two cells
arc = cell_a + cell_b
arc.survey().plot()
```



Build straight sections

[ ]:
```python
long_straight = env.new_line(length=2., components=[
    env.new('mid.lss', xt.Marker, at=1.)
```

Build straight sections

```
[ ]: long_straight = env.new_line(length=2., components=[
         env.new('mid.lss', xt.Marker, at=1.)
     ])
     short_straight = env.new_line(length=1., components=[
         env.new('mid.sss', xt.Marker, at=0.5)
     ])
```

Assemble the ring

```
[ ]: half_ring = (
         long_straight
         + arc
         + short_straight
         - arc # mirror symmetric lattice
     )
     half_ring.survey().plot()
```

```
[ ]: ring = 2 * half_ring
     ring.survey().plot()
```

Replace repeated elements

```
[ ]: ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

```
[ ]: tt = ring.get_table()
     tt.cols['element_type s_start s_center s_end']
```

```
[ ]: # Inspect all quadrupoles
```

Build straight sections

[8]:
```
long_straight = env.new_line(length=2., components=[
    env.new('mid.lss', xt.Marker, at=1.)
])
short_straight = env.new_line(length=1., components=[
    env.new('mid.sss', xt.Marker, at=0.5)
])
```

Assemble the ring

[ ]:
```
half_ring = (
    long_straight
    + arc
    + short_straight
    - arc # mirror symmetric lattice
)
half_ring.survey().plot()
```

[ ]:
```
ring = 2 * half_ring
ring.survey().plot()
```

Replace repeated elements

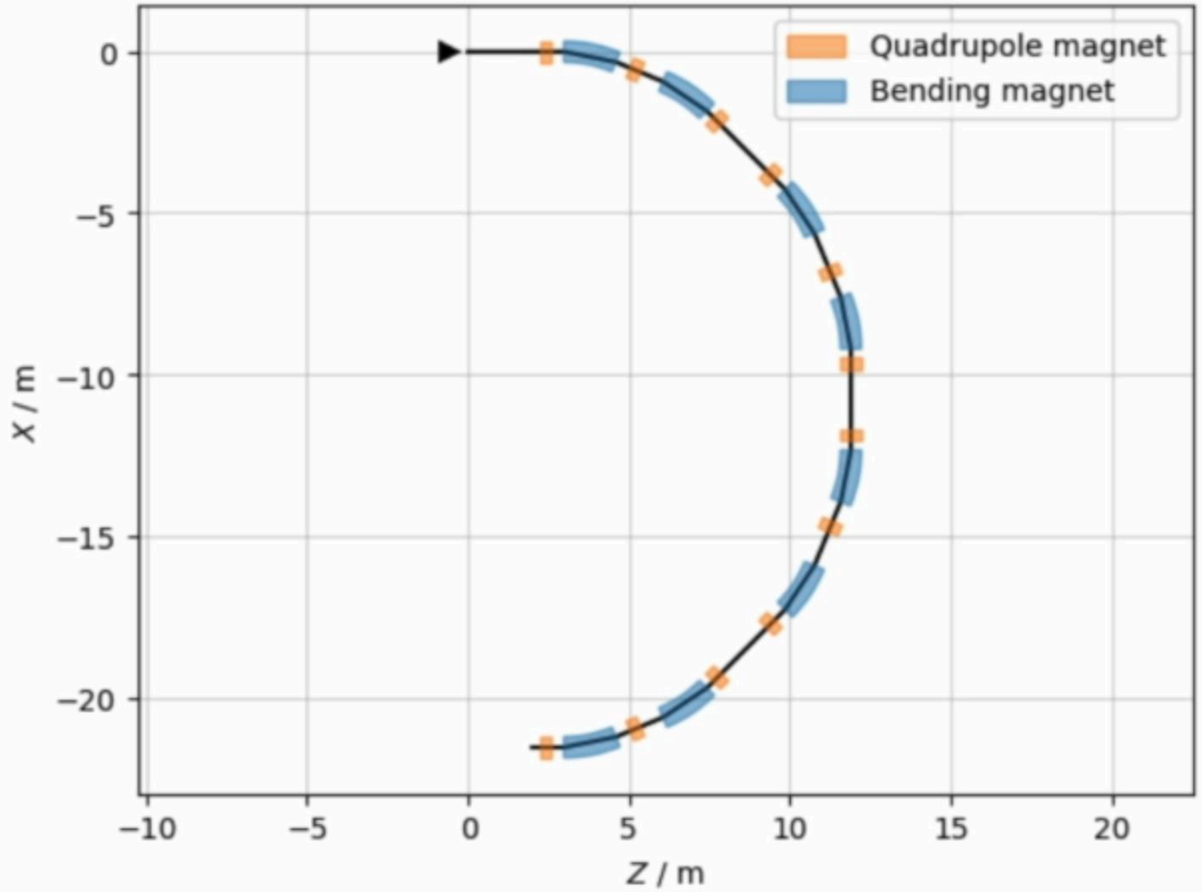
[ ]:
```
ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

[ ]:
```
tt = ring.get_table()
tt.cols['element_type s_start s_center s_end']
```

[ ]:
```
# Inspect all quadrupoles
```

```
)
half_ring.survey().plot()
```




```
[ ]: ring = 2 * half_ring
     ring.survey().plot()
```

Replace repeated elements

```
[ ]: ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

```
[ ]: ring = 2 * half_ring
     ring.survey().plot()
```

Replace repeated elements

```
[ ]: ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

```
[ ]: tt = ring.get_table()
     tt.cols['element_type s_start s_center s_end']
```

```
[ ]: # Inspect all quadrupoles
     tt_quad = tt.rows[tt.element_type=='Quadrupole']
     tt_quad.cols['s_start s_center s_end']
```

```
[ ]: # Tag all quadrupoles in survey plot
     sv = ring.survey()
     sv.plot(labels=tt_quad.name);
```

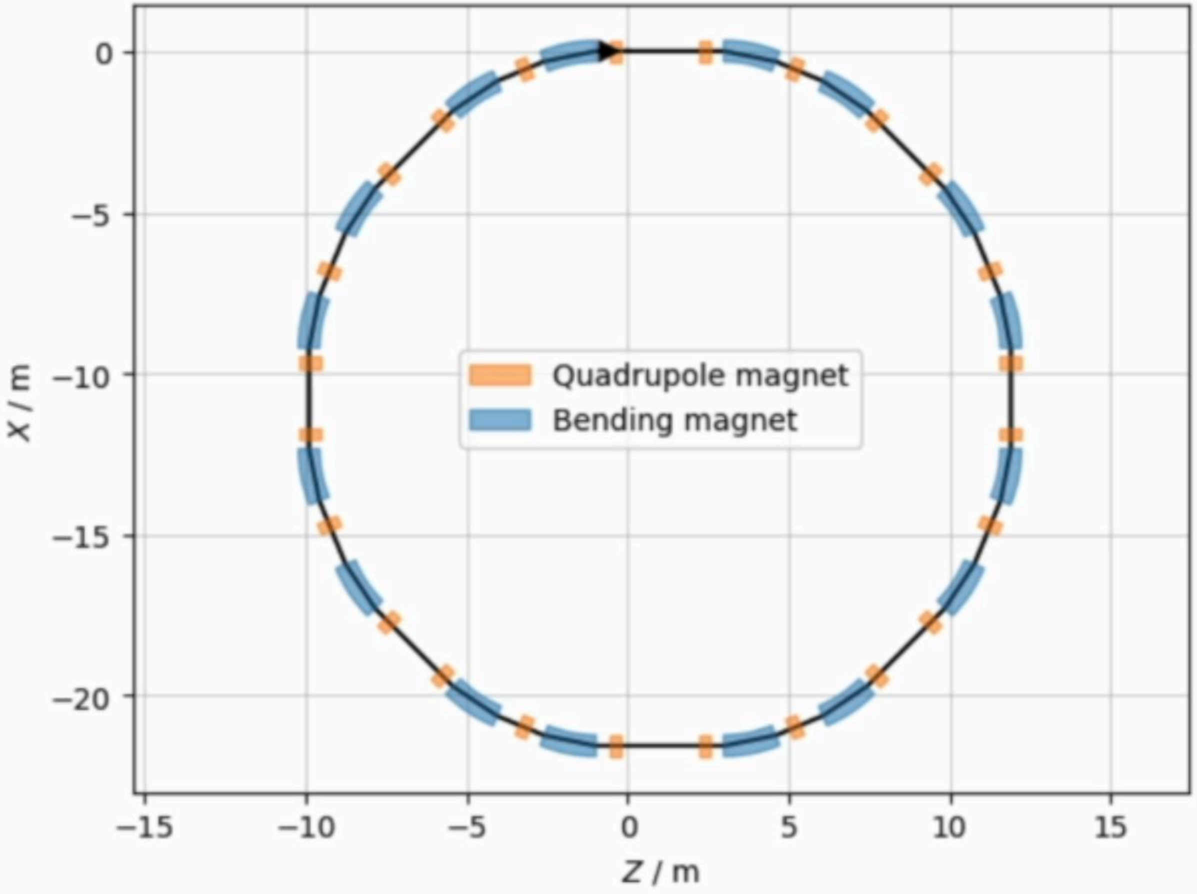
Define and install sextupoles

```
[ ]: # Magnet type
     env.new('ms', xt.Sextupole, length=0.2)

     # Magnet instances
     env.new('msf.1', 'ms', k2='ksf')
     env.new('msf.2', 'ms', k2='ksf')
     env.new('msd.1', 'ms', k2='ksd')
     env.new('msd.2', 'ms', k2='ksd')
     env.new('mse',   'ms', k2='kse');
```

```
[ ]: ring.insert([
         env.place('msf.1', at=-0.2, from_='qfb.0@start'),
```

[10]:
```python
ring = 2 * half_ring
ring.survey().plot()
```

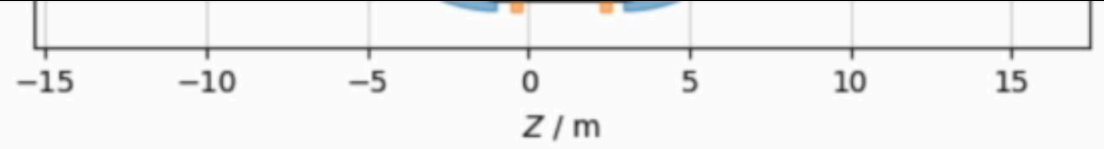


Replace repeated elements

[ ]:
```python
ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

[ ]:
```python
tt = ring.get_table()
tt.cols['element_type s_start s_center s_end']
```



Replace repeated elements

```
[ ]: ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

```
[ ]: tt = ring.get_table()
     tt.cols['element_type s_start s_center s_end']
```

```
[ ]: # Inspect all quadrupoles
     tt_quad = tt.rows[tt.element_type=='Quadrupole']
     tt_quad.cols['s_start s_center s_end']
```

```
[ ]: # Tag all quadrupoles in survey plot
     sv = ring.survey()
     sv.plot(labels=tt_quad.name);
```

Define and install sextupoles

```
[ ]: # Magnet type
     env.new('ms', xt.Sextupole, length=0.2)

     # Magnet instances
     env.new('msf.1', 'ms', k2='ksf')
     env.new('msf.2', 'ms', k2='ksf')
     env.new('msd.1', 'ms', k2='ksd')
     env.new('msd.2', 'ms', k2='ksd')
     env.new('mse',   'ms', k2='kse');
```

[11]:
```python
ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

[ ]:
```python
tt = ring.get_table()
tt.cols['element_type s_start s_center s_end']
```

[ ]:
```python
# Inspect all quadrupoles
tt_quad = tt.rows[tt.element_type=='Quadrupole']
tt_quad.cols['s_start s_center s_end']
```

[ ]:
```python
# Tag all quadrupoles in survey plot
sv = ring.survey()
sv.plot(labels=tt_quad.name);
```

Define and install sextupoles

[ ]:
```python
# Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
env.new('msf.1', 'ms', k2='ksf')
env.new('msf.2', 'ms', k2='ksf')
env.new('msd.1', 'ms', k2='ksd')
env.new('msd.2', 'ms', k2='ksd')
env.new('mse',   'ms', k2='kse');
```

[ ]:
```python
ring.insert([
    env.place('msf.1', at=-0.2, from_='qfb.0@start'),
    env.place('msf.2', at=-0.2, from_='qfb.4@start'),
    env.place('msd.1', at=0.3,  from_='qd.2@end'),
    env.place('msd.2', at=0.3,  from_='qd.6@end'),
    env.place('mse',   at=-0.3, from_='qfa.4@start')
])
```

[11]:
```
ring.replace_all_repeated_elements() # give all elements unique names
```

Inspect beamline table

[*]:
```
tt = ring.get_table()
tt.cols['element_type s_start s_center s_end']
```

[ ]:
```
# Inspect all quadrupoles
tt_quad = tt.rows[tt.element_type=='Quadrupole']
tt_quad.cols['s_start s_center s_end']
```

[ ]:
```
# Tag all quadrupoles in survey plot
sv = ring.survey()
sv.plot(labels=tt_quad.name);
```

Define and install sextupoles

[ ]:
```
# Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
env.new('msf.1', 'ms', k2='ksf')
env.new('msf.2', 'ms', k2='ksf')
env.new('msd.1', 'ms', k2='ksd')
env.new('msd.2', 'ms', k2='ksd')
env.new('mse',   'ms', k2='kse');
```

[ ]:
```
ring.insert([
    env.place('msf.1', at=-0.2, from_='qfb.0@start'),
    env.place('msf.2', at=-0.2, from_='qfb.4@start'),
    env.place('msd.1', at=0.3,  from_='qd.2@end'),
    env.place('msd.2', at=0.3,  from_='qd.6@end'),
    env.place('mse',   at=-0.3, from_='qfa.4@start')
])
```

```python
# Inspect all quadrupoles
tt_quad = tt.rows[tt.element_type=='Quadrupole']
tt_quad.cols['s_start s_center s_end']
```

```python
# Tag all quadrupoles in survey plot
sv = ring.survey()
sv.plot(labels=tt_quad.name);
```

Define and install sextupoles

```python
# Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
env.new('msf.1', 'ms', k2='ksf')
env.new('msf.2', 'ms', k2='ksf')
env.new('msd.1', 'ms', k2='ksd')
env.new('msd.2', 'ms', k2='ksd')
env.new('mse',   'ms', k2='kse');
```

```python
ring.insert([
    env.place('msf.1', at=-0.2, from_='qfb.0@start'),
    env.place('msf.2', at=-0.2, from_='qfb.4@start'),
    env.place('msd.1', at=0.3,  from_='qd.2@end'),
    env.place('msd.2', at=0.3,  from_='qd.6@end'),
    env.place('mse',   at=-0.3, from_='qfa.4@start')
])
```

```python
# Inspect sextupoles in the survey
sv = ring.survey()
sv.plot(labels=['msf.1', 'msf.2', 'msd.1', 'msd.2', 'mse'])
```

Inspect circuit structure

[13]:
```
# Inspect all quadrupoles
tt_quad = tt.rows[tt.element_type=='Quadrupole']
tt_quad.cols['s_start s_center s_end']
```

[13]:
```
Table: 24 rows, 4 cols
name          s_start     s_center        s_end
qfa.0          2.2125       2.3875       2.5625
qd.0           5.1175       5.2925       5.4675
qfa.1          8.1525       8.3275       8.5025
qfb.0         10.5025      10.6775      10.8525
qd.1          14.0875      14.2625      14.4375
qfb.1         17.0225      17.1975      17.3725
qfb.2         19.2475      19.4225      19.5975
qd.2          22.1825      22.3575      22.5325
qfb.3         25.7675      25.9425      26.1175
qfa.2         28.1175      28.2925      28.4675
qd.3          31.1525      31.3275      31.5025
qfa.3         34.0575      34.2325      34.4075
qfa.4         36.8325      37.0075      37.1825
qd.4          39.7375      39.9125      40.0875
qfa.5         42.7725      42.9475      43.1225
qfb.4         45.1225      45.2975      45.4725
qd.5          48.7075      48.8825      49.0575
qfb.5         51.6425      51.8175      51.9925
qfb.6         53.8675      54.0425      54.2175
qd.6          56.8025      56.9775      57.1525
qfb.7         60.3875      60.5625      60.7375
qfa.6         62.7375      62.9125      63.0875
qd.7          65.7725      65.9475      66.1225
qfa.7         68.6775      68.8525      69.0275
```

[ ]:
```
# Tag all quadrupoles in survey plot
sv = ring.survey()
sv.plot(labels=tt_quad.name);
```

Define and install sextupoles

```
qfa.7        68.6775       68.8525       69.0275
```

[ ]:
```
# Tag all quadrupoles in survey plot
sv = ring.survey()
sv.plot(labels=tt_quad.name);
```

Define and install sextupoles

[ ]:
```
# Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
env.new('msf.1', 'ms', k2='ksf')
env.new('msf.2', 'ms', k2='ksf')
env.new('msd.1', 'ms', k2='ksd')
env.new('msd.2', 'ms', k2='ksd')
env.new('mse',   'ms', k2='kse');
```

[ ]:
```
ring.insert([
    env.place('msf.1', at=-0.2, from_='qfb.0@start'),
    env.place('msf.2', at=-0.2, from_='qfb.4@start'),
    env.place('msd.1', at=0.3,  from_='qd.2@end'),
    env.place('msd.2', at=0.3,  from_='qd.6@end'),
    env.place('mse',   at=-0.3, from_='qfa.4@start')
])
```

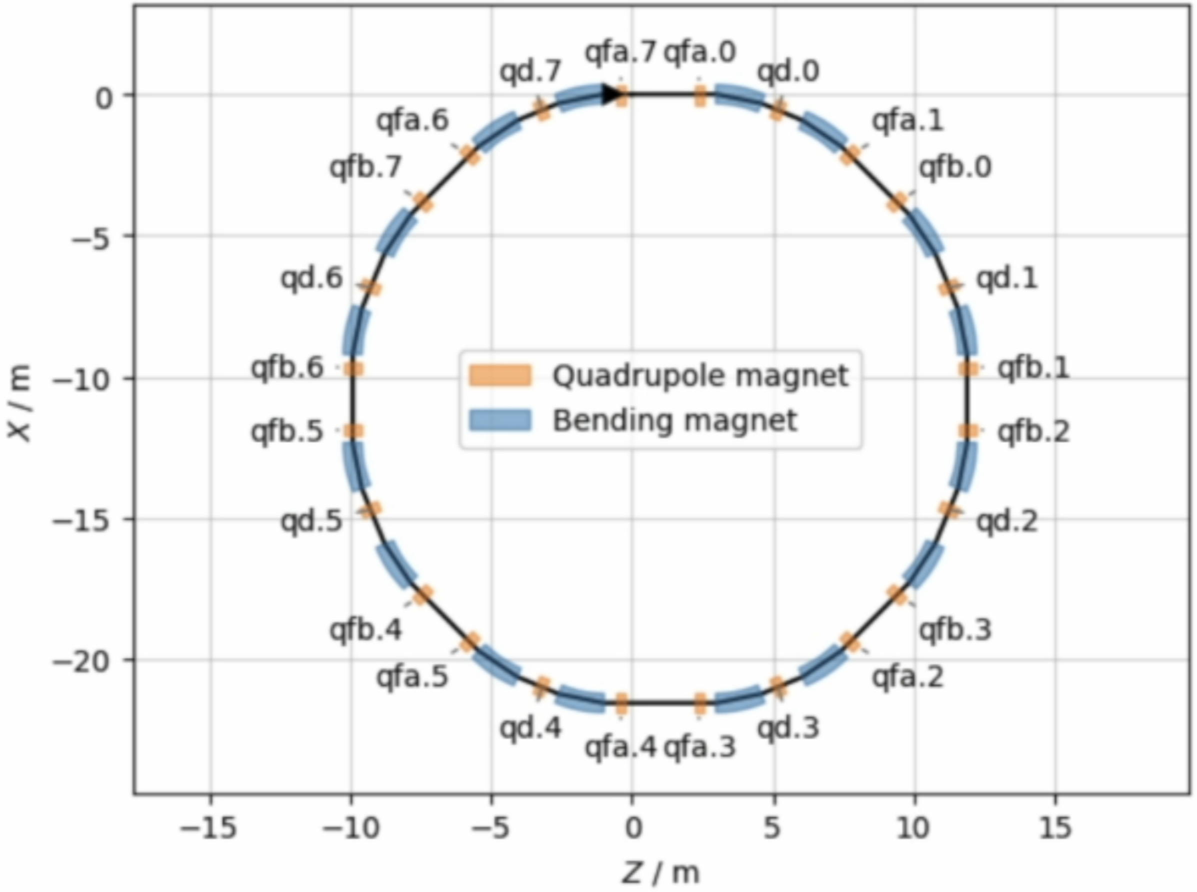
[ ]:
```
# Inspect sextupoles in the survey
sv = ring.survey()
sv.plot(labels=['msf.1', 'msf.2', 'msd.1', 'msd.2', 'mse'])
```

Inspect circuit structure

[ ]:
```
# Entities controlled by one knob
env.info('ksf')
```

```
qfa.7        68.6775        68.8525        69.0275
```

[14]:
```
# Tag all quadrupoles in survey plot
sv = ring.survey()
sv.plot(labels=tt_quad.name);
```



Define and install sextupoles

[ ]:
```
# Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
```

Define and install sextupoles

```
[ ]: # Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
env.new('msf.1', 'ms', k2='ksf')
env.new('msf.2', 'ms', k2='ksf')
env.new('msd.1', 'ms', k2='ksd')
env.new('msd.2', 'ms', k2='ksd')
env.new('mse',   'ms', k2='kse');
```

```
[ ]: ring.insert([
    env.place('msf.1', at=-0.2, from_='qfb.0@start'),
    env.place('msf.2', at=-0.2, from_='qfb.4@start'),
    env.place('msd.1', at=0.3,  from_='qd.2@end'),
    env.place('msd.2', at=0.3,  from_='qd.6@end'),
    env.place('mse',   at=-0.3, from_='qfa.4@start')
])
```

```
[ ]: # Inspect sextupoles in the survey
sv = ring.survey()
sv.plot(labels=['msf.1', 'msf.2', 'msd.1', 'msd.2', 'mse'])
```

Inspect circuit structure

```
[ ]: # Entities controlled by one knob
env.info('ksf')
```

```
[ ]: # Inspect knob controlling one magnet
env.info('msf.1')
```

Install RF cavity

Define and install sextupoles

[15]:
```
# Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
env.new('msf.1', 'ms', k2='ksf')
env.new('msf.2', 'ms', k2='ksf')
env.new('msd.1', 'ms', k2='ksd')
env.new('msd.2', 'ms', k2='ksd')
env.new('mse',   'ms', k2='kse');
```

[ ]:
```
ring.insert([
    env.place('msf.1', at=-0.2, from_='qfb.0@start'),
    env.place('msf.2', at=-0.2, from_='qfb.4@start'),
    env.place('msd.1', at=0.3,  from_='qd.2@end'),
    env.place('msd.2', at=0.3,  from_='qd.6@end'),
    env.place('mse',   at=-0.3, from_='qfa.4@start')
])
```

[ ]:
```
# Inspect sextupoles in the survey
sv = ring.survey()
sv.plot(labels=['msf.1', 'msf.2', 'msd.1', 'msd.2', 'mse'])
```

Inspect circuit structure

[ ]:
```
# Entities controlled by one knob
env.info('ksf')
```

[ ]:
```
# Inspect knob controlling one magnet
env.info('msf.1')
```

Install RF cavity

Define and install sextupoles

[15]:
```
# Magnet type
env.new('ms', xt.Sextupole, length=0.2)

# Magnet instances
env.new('msf.1', 'ms', k2='ksf')
env.new('msf.2', 'ms', k2='ksf')
env.new('msd.1', 'ms', k2='ksd')
env.new('msd.2', 'ms', k2='ksd')
env.new('mse',   'ms', k2='kse');
```

[16]:
```
ring.insert([
    env.place('msf.1', at=-0.2, from_='qfb.0@start'),
    env.place('msf.2', at=-0.2, from_='qfb.4@start'),
    env.place('msd.1', at=0.3,  from_='qd.2@end'),
    env.place('msd.2', at=0.3,  from_='qd.6@end'),
    env.place('mse',   at=-0.3, from_='qfa.4@start')
])
```

Slicing line: 100% 100/100 [00:00<00:00, 10277.64it/s]

[ ]:
```
# Inspect sextupoles in the survey
sv = ring.survey()
sv.plot(labels=['msf.1', 'msf.2', 'msd.1', 'msd.2', 'mse'])
```

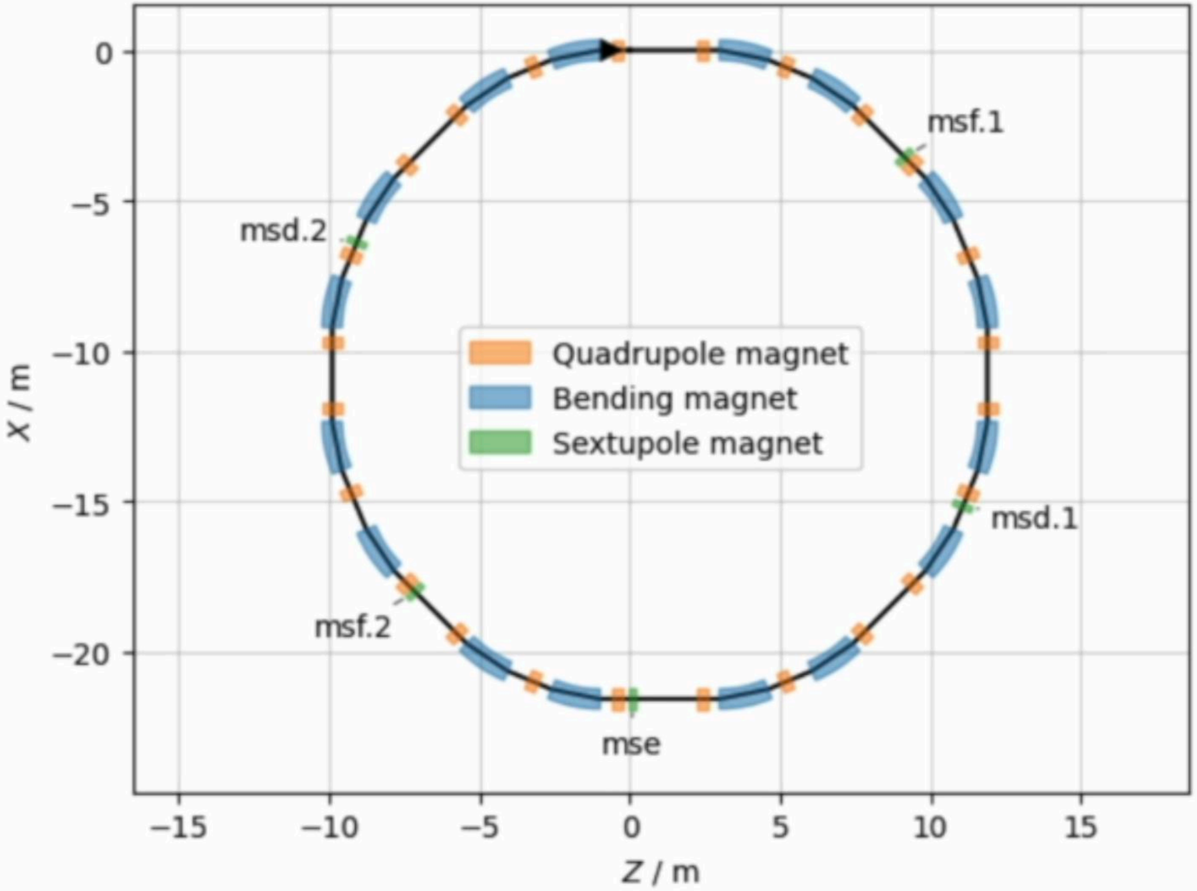
Inspect circuit structure

[ ]:
```
# Entities controlled by one knob
env.info('ksf')
```

[ ]:
```
# Inspect knob controlling one magnet
env.info('msf.1')
```

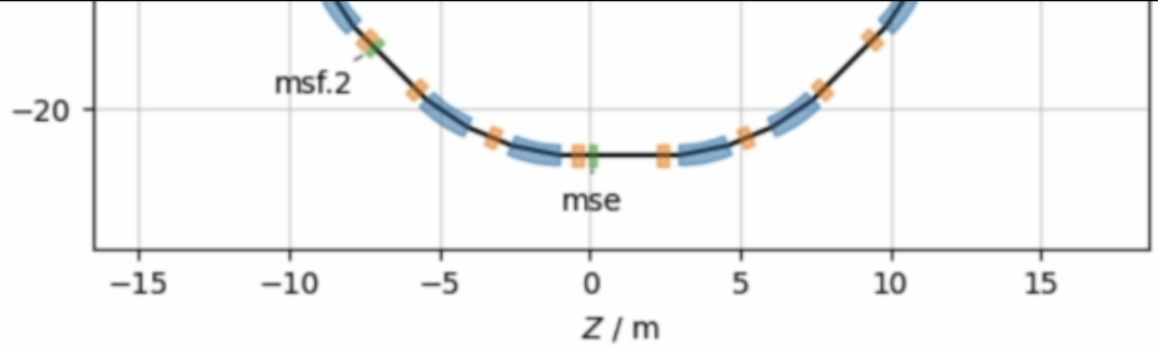
Slicing line: 100% 100/100 [00:00<00:00, 10277.64it/s]

[17]:
```python
# Inspect sextupoles in the survey
sv = ring.survey()
sv.plot(labels=['msf.1', 'msf.2', 'msd.1', 'msd.2', 'mse'])
```



Inspect circuit structure

[ ]:
```python
# Entities controlled by one knob
env.info('ksf')
```



Inspect circuit structure

```
[ ]: # Entities controlled by one knob
     env.info('ksf')
```

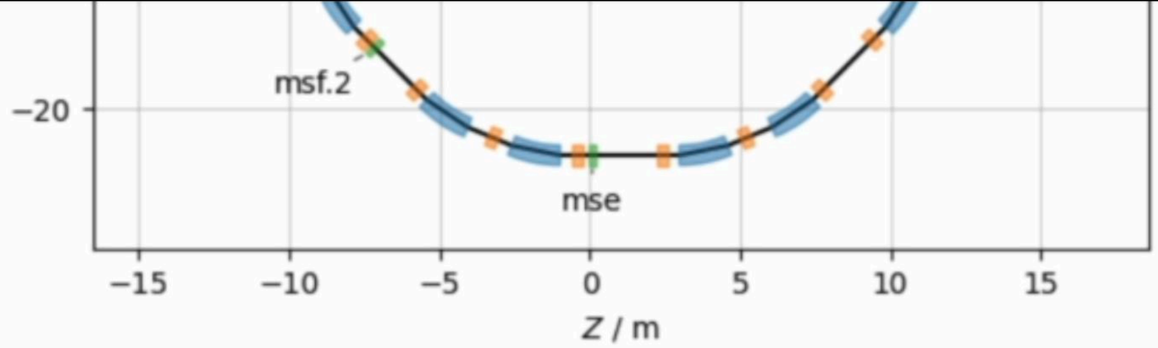
```
[ ]: # Inspect knob controlling one magnet
     env.info('msf.1')
```

Install RF cavity

```
[ ]: env.new('rf1', xt.Cavity, voltage='vrf', frequency='frf')
     ring.insert('rf1', at=0.5, from_='qfa.3@start')
     sv = ring.survey()
     sv.plot(labels=['rf1'])
```

Save lattice to json file

```
[ ]: env['ring'] = ring
     env.to_json('pimm.json')
```



Inspect circuit structure

[18]:
```
# Entities controlled by one knob
env.info('ksf')
```

```
#  vars['ksf']._get_value()
   vars['ksf'] = 0.0

#  vars['ksf']._expr is None

#  vars['ksf']._find_dependant_targets()
   element_refs['msf.2'].k2
   element_refs['msf.1'].k2
```

[ ]:
```
# Inspect knob controlling one magnet
env.info('msf.1')
```

Install RF cavity

[ ]:
```
env.new('rf1', xt.Cavity, voltage='vrf', frequency='frf')
ring.insert('rf1', at=0.5, from_='qfa.3@start')
sv = ring.survey()
sv.plot(labels=['rf1'])
```

```
[18]: # Entities controlled by one knob
env.info('ksf')
```

```
#  vars['ksf']._get_value()
   vars['ksf'] = 0.0

#  vars['ksf']._expr is None

#  vars['ksf']._find_dependant_targets()
   element_refs['msf.2'].k2
   element_refs['msf.1'].k2
```

```
[19]: # Inspect knob controlling one magnet
env.info('msf.1')
```

```
Element of type:  Sextupole
name                value                                                  expr
k2                  0.0                                                    vars['ksf']
k2s                 0.0                                                    None
length              0.2                                                    None
order               5                                                      None
inv_factorial_order 0.008333333333333333                                   None
knl                 [0. 0. 0. 0. 0. 0.]                                    None
ksl                 [0. 0. 0. 0. 0. 0.]                                    None
edge_entry_active   0                                                      None
edge_exit_active    0                                                      None
num_multipole_kicks 0                                                      None
model               adaptive                                               None
integrator          adaptive                                               None
radiation_flag      0                                                      None
delta_taper         0.0                                                    None
_sin_rot_s          -999.0                                                 None
_cos_rot_s          -999.0                                                 None
_shift_x            0.0                                                    None
_shift_y            0.0                                                    None
_shift_s            0.0                                                    None
_internal_record_id RecordIdentifier(buffer_id=np.int64(0), offset=np.int64(0)) None
```

Install RF cavity

```
k2                  0.0                                                          vars['ksf']
k2s                 0.0                                                          None
length              0.2                                                          None
order               5                                                            None
inv_factorial_order 0.008333333333333333                                         None
knl                 [0. 0. 0. 0. 0. 0.]                                          None
ksl                 [0. 0. 0. 0. 0. 0.]                                          None
edge_entry_active   0                                                            None
edge_exit_active    0                                                            None
num_multipole_kicks 0                                                            None
model               adaptive                                                     None
integrator          adaptive                                                     None
radiation_flag      0                                                            None
delta_taper         0.0                                                          None
_sin_rot_s          -999.0                                                       None
_cos_rot_s          -999.0                                                       None
_shift_x            0.0                                                          None
_shift_y            0.0                                                          None
_shift_s            0.0                                                          None
_internal_record_id RecordIdentifier(buffer_id=np.int64(0), offset=np.int64(0)) None
```

Install RF cavity

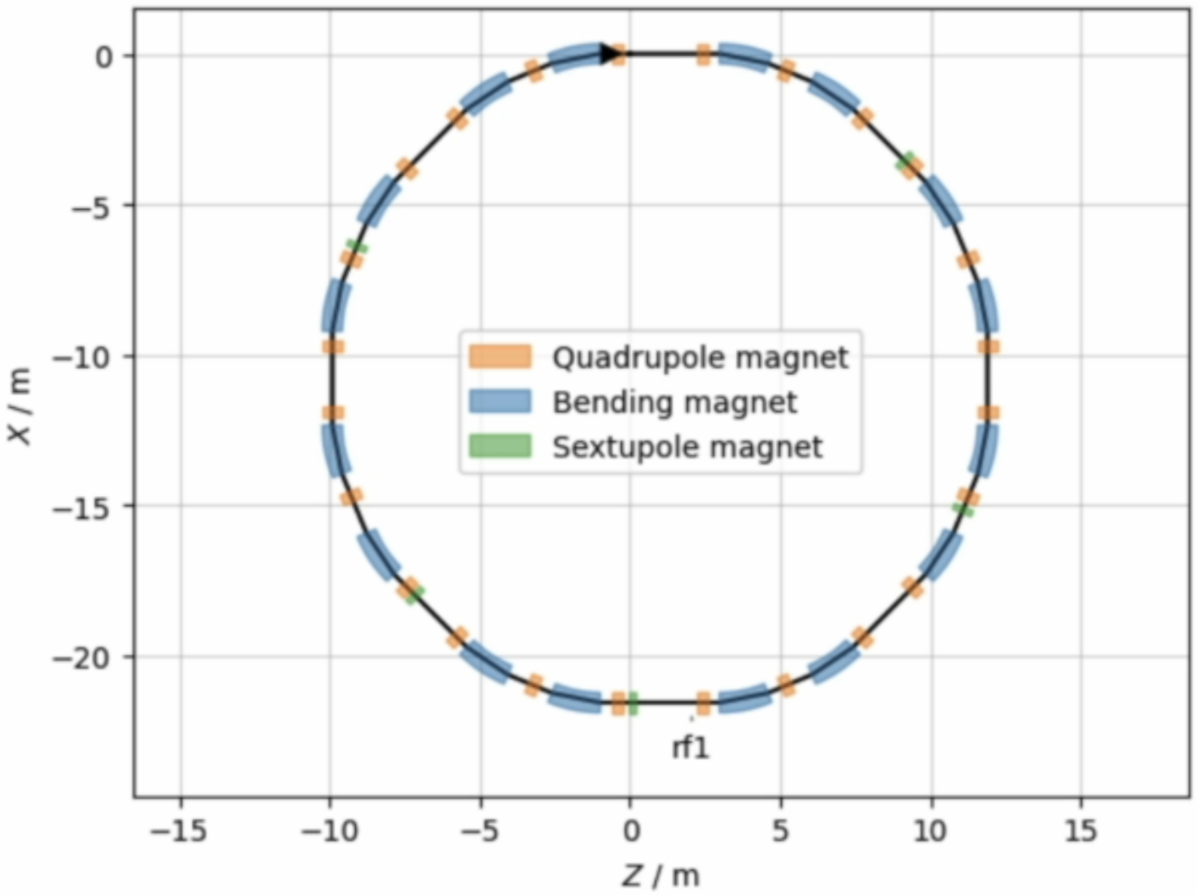
[ ]:
```
env.new('rf1', xt.Cavity, voltage='vrf', frequency='frf')
ring.insert('rf1', at=0.5, from_='qfa.3@start')
sv = ring.survey()
sv.plot(labels=['rf1'])
```

Save lattice to json file

[ ]:
```
env['ring'] = ring
env.to_json('pimm.json')
```

```
sv = ring.survey()
sv.plot(labels=['rf1'])
```

Slicing line: 100% 109/109 [00:00<00:00, 16174.17it/s]

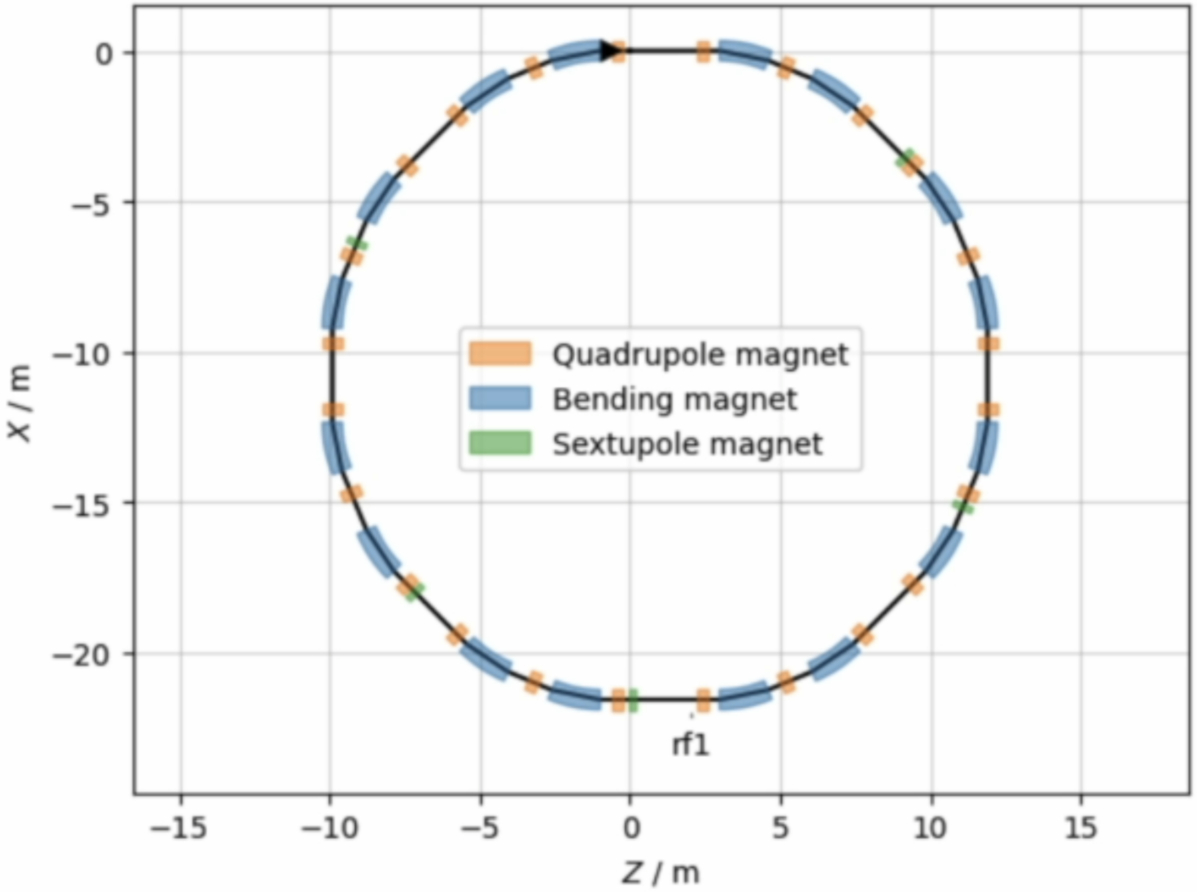


Save lattice to json file

```
[ ]: env['ring'] = ring
     env.to_json('pimm.json')
```

```
sv = ring.survey()
sv.plot(labels=['rf1'])
```

Slicing line: 100% 109/109 [00:00<00:00, 16174.17it/s]



Save lattice to json file

[21]:
```
env['ring'] = ring
env.to_json('pimm.json')
```

[ ]:

Demo 2 – Optics for slow extraction

These demo files are adapted from https://github.com/xsuite/tutorial_cern_seminar.

Optics matching & slow extraction

```
import xtrack as xt
import numpy as np
import matplotlib.pyplot as plt
```

Load model

```
env = xt.Environment.from_json('./pimm.json')
ring = env['ring']
```

Set some initial strengths to get a first twiss

```
env['kqfa'] =  0.1
env['kqfb'] =  0.1
env['kqd']  = -0.2
```

```
tw = ring.twiss4d(compute_chromatic_properties=False)
tw.plot()
```

Match the tunes and dispersion in the straight sections

```
opt_tune = ring.match(
    solve=False, # <- prepare the match without running it
    compute_chromatic_properties=False,
    method='4d',
    vary=[
        xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
        xt.Vary('kqfb', limits=(0, 10),  step=1e-5),
        xt.Vary('kqd', limits=(-10, 0), step=1e-5),
    ],
```

Optics matching & slow extraction

[1]:
```python
import xtrack as xt
import numpy as np
import matplotlib.pyplot as plt
```

Load model

[ ]:
```python
env = xt.Environment.from_json('./pimm.json')
ring = env['ring']
```

Set some initial strengths to get a first twiss

[ ]:
```python
env['kqfa'] =  0.1
env['kqfb'] =  0.1
env['kqd']  = -0.2
```

[ ]:
```python
tw = ring.twiss4d(compute_chromatic_properties=False)
tw.plot()
```

Match the tunes and dispersion in the straight sections

[ ]:
```python
opt_tune = ring.match(
    solve=False, # <- prepare the match without running it
    compute_chromatic_properties=False,
    method='4d',
    vary=[
        xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
        xt.Vary('kqfb', limits=(0, 10),  step=1e-5),
        xt.Vary('kqd', limits=(-10, 0), step=1e-5),
    ],
```

Optics matching & slow extraction

[1]:
```
import xtrack as xt
import numpy as np
import matplotlib.pyplot as plt
```

Load model

[2]:
```
env = xt.Environment.from_json('./pimm.json')
ring = env['ring']
```

Loading line from dict: 100% 148/148 [00:00<00:00, 9707.67it/s]

```
Done loading line from dict.
```

Set some initial strengths to get a first twiss

[ ]:
```
env['kqfa'] =  0.1
env['kqfb'] =  0.1
env['kqd']  = -0.2
```

[ ]:
```
tw = ring.twiss4d(compute_chromatic_properties=False)
tw.plot()
```

Match the tunes and dispersion in the straight sections

[ ]:
```
opt_tune = ring.match(
    solve=False, # <- prepare the match without running it
    compute_chromatic_properties=False,
    method='4d',
    vary=[
        xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
```

Optics matching & slow extraction

[1]:
```
import xtrack as xt
import numpy as np
import matplotlib.pyplot as plt
```

Load model

[2]:
```
env = xt.Environment.from_json('./pimm.json')
ring = env['ring']
```

Loading line from dict: 100% 148/148 [00:00<00:00, 9707.67it/s]

```
Done loading line from dict.
```

Set some initial strengths to get a first twiss

[3]:
```
env['kqfa'] =  0.1
env['kqfb'] =  0.1
env['kqd']  = -0.2
```

[ ]:
```
tw = ring.twiss4d(compute_chromatic_properties=False)
tw.plot()
```

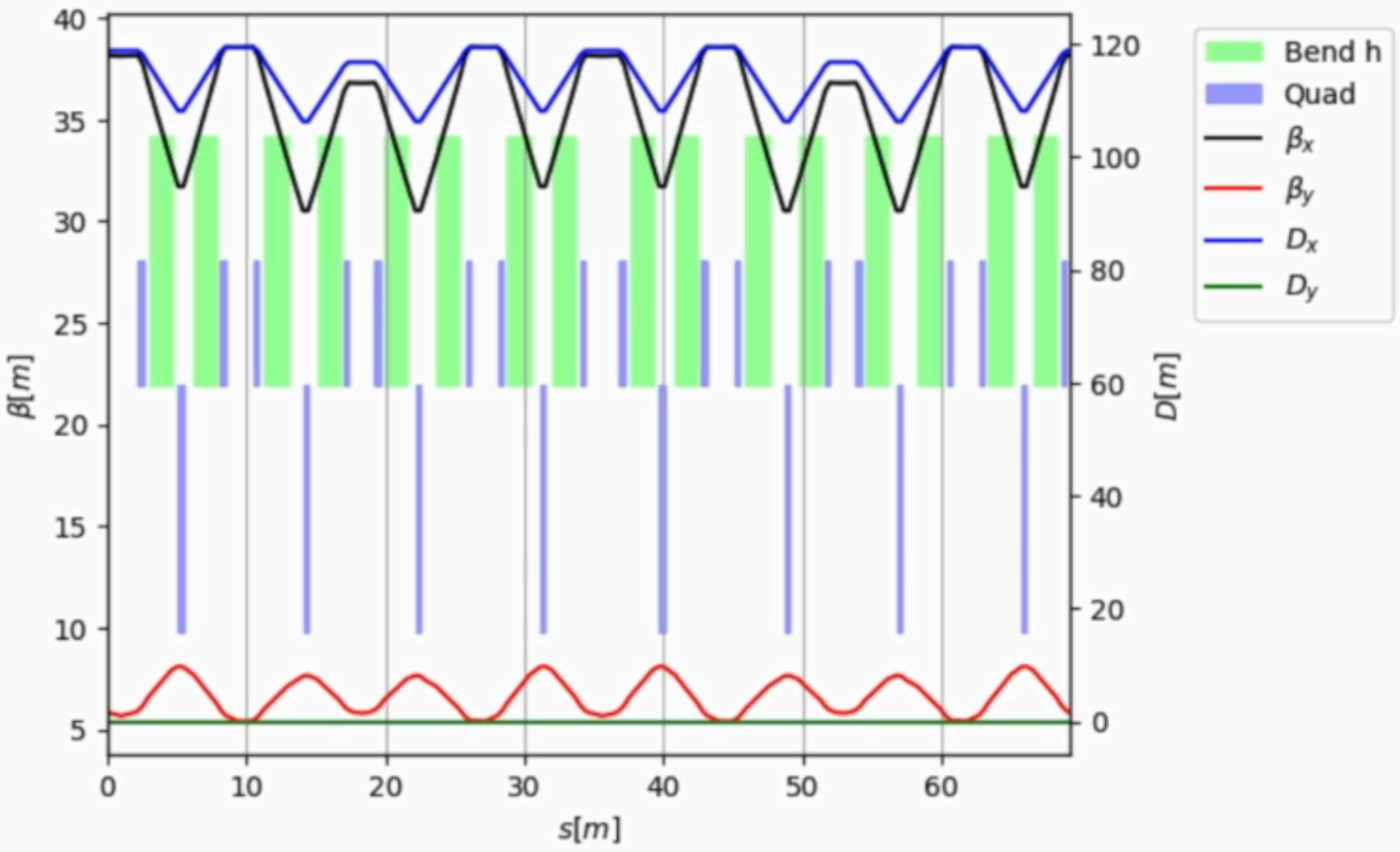
Match the tunes and dispersion in the straight sections

[ ]:
```
opt_tune = ring.match(
    solve=False, # <- prepare the match without running it
    compute_chromatic_properties=False,
    method='4d',
    vary=[
        xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
```

[3]:
```python
env['kqfa'] =  0.1
env['kqfb'] =  0.1
env['kqd']  = -0.2
```

[4]:
```python
tw = ring.twiss4d(compute_chromatic_properties=False)
tw.plot()
```

[4]: <xtrack.twissplot.TwissPlot object at 0x14daf3da0>



Match the tunes and dispersion in the straight sections

Match the tunes and dispersion in the straight sections

```
[ ]: opt_tune = ring.match(
         solve=False, # <- prepare the match without running it
         compute_chromatic_properties=False,
         method='4d',
         vary=[
             xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
             xt.Vary('kqfb', limits=(0, 10),  step=1e-5),
             xt.Vary('kqd', limits=(-10, 0), step=1e-5),
         ],
         targets=[
             # Horizontal tune close to 3rd order resonance
             xt.TargetSet(qx=1.665, qy=1.72, tol=1e-4),
             # Set horizontal dispersion to zero in the middle
             # of the long straight sections
             xt.TargetSet(dx=0, at='mid.lss.0'),
             xt.TargetSet(dx=0, at='mid.lss.1'),
         ]
     )
```

```
[ ]: opt_tune.target_status()
```

```
[ ]: opt_tune.run_jacobian(20)
```

```
[ ]: opt_tune.target_status()
```

```
[ ]: opt_tune.vary_status()
```

```
[ ]: tw = ring.twiss4d()
     pl = tw.plot()
     pl.ylim(left_hi=40, right_lo=-20, right_hi=20, lattice_hi=1.5, lattice_lo=-7)
```

Correct chromaticity

Match the tunes and dispersion in the straight sections

[5]:
```
opt_tune = ring.match(
    solve=False, # <- prepare the match without running it
    compute_chromatic_properties=False,
    method='4d',
    vary=[
        xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
        xt.Vary('kqfb', limits=(0, 10),  step=1e-5),
        xt.Vary('kqd', limits=(-10, 0), step=1e-5),
    ],
    targets=[
        # Horizontal tune close to 3rd order resonance
        xt.TargetSet(qx=1.665, qy=1.72, tol=1e-4),
        # Set horizontal dispersion to zero in the middle
        # of the long straight sections
        xt.TargetSet(dx=0, at='mid.lss.0'),
        xt.TargetSet(dx=0, at='mid.lss.1'),
    ]
)
```

[ ]:
```
opt_tune.target_status()
```

[ ]:
```
opt_tune.run_jacobian(20)
```

[ ]:
```
opt_tune.target_status()
```

[ ]:
```
opt_tune.vary_status()
```

[ ]:
```
tw = ring.twiss4d()
pl = tw.plot()
pl.ylim(left_hi=40, right_lo=-20, right_hi=20, lattice_hi=1.5, lattice_lo=-7)
```

Correct chromaticity

Match the tunes and dispersion in the straight sections

[5]:
```
opt_tune = ring.match(
    solve=False, # <- prepare the match without running it
    compute_chromatic_properties=False,
    method='4d',
    vary=[
        xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
        xt.Vary('kqfb', limits=(0, 10),  step=1e-5),
        xt.Vary('kqd', limits=(-10, 0), step=1e-5),
    ],
    targets=[
        # Horizontal tune close to 3rd order resonance
        xt.TargetSet(qx=1.665, qy=1.72, tol=1e-4),
        # Set horizontal dispersion to zero in the middle
        # of the long straight sections
        xt.TargetSet(dx=0, at='mid.lss.0'),
        xt.TargetSet(dx=0, at='mid.lss.1'),
    ]
)
```

[6]:
```
opt_tune.target_status()
```

```
Target status:               alty = 1.6806e+03
id state tag           tol_met       residue   current_val    target_val description
0  ON    qx              False      -1.35461      0.310395         1.665 'qx', val=1.665, tol=0.0001, weight=10
1  ON    qy              False    -0.0237697       1.69623          1.72 'qy', val=1.72, tol=0.0001, weight=10
2  ON    mid.lss.0_dx    False        118.83        118.83             0 ('dx', 'mid.lss.0'), val=0, tol=1e-09, w ...
3  ON    mid.lss.1_dx    False        118.83        118.83             0 ('dx', 'mid.lss.1'), val=0, tol=1e-09, w ...
```

[ ]:
```
opt_tune.run_jacobian(20)
```

[ ]:
```
opt_tune.target_status()
```

[ ]:
```
opt_tune.vary_status()
```

Match the tunes and dispersion in the straight sections

[5]:
```
opt_tune = ring.match(
    solve=False, # <- prepare the match without running it
    compute_chromatic_properties=False,
    method='4d',
    vary=[
        xt.Vary('kqfa', limits=(0, 10),  step=1e-5),
        xt.Vary('kqfb', limits=(0, 10),  step=1e-5),
        xt.Vary('kqd', limits=(-10, 0), step=1e-5),
    ],
    targets=[
        # Horizontal tune close to 3rd order resonance
        xt.TargetSet(qx=1.665, qy=1.72, tol=1e-4),
        # Set horizontal dispersion to zero in the middle
        # of the long straight sections
        xt.TargetSet(dx=0, at='mid.lss.0'),
        xt.TargetSet(dx=0, at='mid.lss.1'),
    ]
)
```

[6]:
```
opt_tune.target_status()
```

```
Target status:               alty = 1.6806e+03
id state tag           tol_met      residue  current_val    target_val description
0  ON    qx              False     -1.35461     0.310395         1.665 'qx', val=1.665, tol=0.0001, weight=10
1  ON    qy              False   -0.0237697      1.69623          1.72 'qy', val=1.72, tol=0.0001, weight=10
2  ON    mid.lss.0_dx    False       118.83       118.83             0 ('dx', 'mid.lss.0'), val=0, tol=1e-09, w ...
3  ON    mid.lss.1_dx    False       118.83       118.83             0 ('dx', 'mid.lss.1'), val=0, tol=1e-09, w ...
```

[7]:
```
opt_tune.run_jacobian(20)
```

```
Optimize - start penalty: 1681
Matching: model call n. 69 penalty = 6.6246e-10
Optimize - end penalty:  6.62464e-10
```

[ ]:
```
opt_tune.target_status()
```

```
optimize - end penalty:  6.62464e-10
```

```
[ ]: opt_tune.target_status()
```

```
[ ]: opt_tune.vary_status()
```

```
[ ]: tw = ring.twiss4d()
     pl = tw.plot()
     pl.ylim(left_hi=40, right_lo=-20, right_hi=20, lattice_hi=1.5, lattice_lo=-7)
```

Correct chromaticity

```
[ ]: opt_chrom = ring.match(
         solve=False,
         method='4d',
         vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
         targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

```
[ ]: opt_chrom.run_ls_dogbox(30)
```

```
[ ]: opt_chrom.target_status()
```

Observe deformation of phase space when exciting the resonance

```
[ ]: # Generate 20 particles on the x axis
     x_gen = np.linspace(0, 0.5e-2, 20)
     p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

```
[ ]: # Set extraction sextupole
     env['kse'] = 0

     # Track 1000 turns
     ring.track(p0.copy(), num_turns=1000, turn_by_turn_monitor=True)
     rec = ring.record_last_track
```

```
Optimize - end penalty: 6.62464e-10
```

[8]:
```
opt_tune.target_status()
```

```
Target status:               nalty = 6.6246e-10
id state tag            tol_met      residue   current_val    target_val description
0  ON    qx                True -1.88027e-12         1.665         1.665 'qx', val=1.665, tol=0.0001, weight=10
1  ON    qy                True   -2.085e-13          1.72          1.72 'qy', val=1.72, tol=0.0001, weight=10
2  ON    mid.lss.0_dx      True  4.38136e-11   4.38136e-11             0 ('dx', 'mid.lss.0'), val=0, tol=1e-09, w ...
3  ON    mid.lss.1_dx      True  4.96525e-11   4.96525e-11             0 ('dx', 'mid.lss.1'), val=0, tol=1e-09, w ...
```

[ ]:
```
opt_tune.vary_status()
```

[ ]:
```
tw = ring.twiss4d()
pl = tw.plot()
pl.ylim(left_hi=40, right_lo=-20, right_hi=20, lattice_hi=1.5, lattice_lo=-7)
```

Correct chromaticity

[ ]:
```
opt_chrom = ring.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

[ ]:
```
opt_chrom.run_ls_dogbox(30)
```

[ ]:
```
opt_chrom.target_status()
```

Observe deformation of phase space when exciting the resonance

[ ]:
```
# Generate 20 particles on the x axis
x_gen = np.linspace(0, 0.5e-2, 20)
p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

[ ]:
```
# Set extraction sextupole
```

```
Optimize - end penalty:  6.62464e-10
```

[8]:
```
opt_tune.target_status()
```

```
Target status:              nalty = 6.6246e-10
id state tag          tol_met      residue   current_val    target_val description
0  ON    qx              True -1.88027e-12         1.665         1.665 'qx', val=1.665, tol=0.0001, weight=10
1  ON    qy              True   -2.085e-13          1.72          1.72 'qy', val=1.72, tol=0.0001, weight=10
2  ON    mid.lss.0_dx    True  4.38136e-11   4.38136e-11             0 ('dx', 'mid.lss.0'), val=0, tol=1e-09, w ...
3  ON    mid.lss.1_dx    True  4.96525e-11   4.96525e-11             0 ('dx', 'mid.lss.1'), val=0, tol=1e-09, w ...
```

[ ]:
```
opt_tune.vary_status()
```

[ ]:
```
tw = ring.twiss4d()
pl = tw.plot()
pl.ylim(left_hi=40, right_lo=-20, right_hi=20, lattice_hi=1.5, lattice_lo=-7)
```

Correct chromaticity

[ ]:
```
opt_chrom = ring.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

[ ]:
```
opt_chrom.run_ls_dogbox(30)
```

[ ]:
```
opt_chrom.target_status()
```

Observe deformation of phase space when exciting the resonance

[ ]:
```
# Generate 20 particles on the x axis
x_gen = np.linspace(0, 0.5e-2, 20)
p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

[ ]:
```
# Set extraction sextupole
```

```
Optimize - end penalty: 6.62464e-10
```

[8]:
```
opt_tune.target_status()
```

```
Target status:              nalty = 6.6246e-10
id state tag          tol_met     residue  current_val     target_val description
0  ON    qx             True -1.88027e-12        1.665          1.665 'qx', val=1.665, tol=0.0001, weight=10
1  ON    qy             True   -2.085e-13         1.72           1.72 'qy', val=1.72, tol=0.0001, weight=10
2  ON    mid.lss.0_dx   True  4.38136e-11  4.38136e-11              0 ('dx', 'mid.lss.0'), val=0, tol=1e-09, w ...
3  ON    mid.lss.1_dx   True  4.96525e-11  4.96525e-11              0 ('dx', 'mid.lss.1'), val=0, tol=1e-09, w ...
```

[9]:
```
opt_tune.vary_status()
```

```
Vary status:
id state tag met name lower_limit   current_val upper_limit val_at_iter_0         step       weight
0  ON        OK  kqfa           0      0.347931          10           0.1        1e-05            1
1  ON        OK  kqfb           0      0.547781          10           0.1        1e-05            1
2  ON        OK  kqd          -10     -0.592287           0          -0.2        1e-05            1
```

[ ]:
```
tw = ring.twiss4d()
pl = tw.plot()
pl.ylim(left_hi=40, right_lo=-20, right_hi=20, lattice_hi=1.5, lattice_lo=-7)
```

Correct chromaticity

[ ]:
```
opt_chrom = ring.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

[ ]:
```
opt_chrom.run_ls_dogbox(30)
```

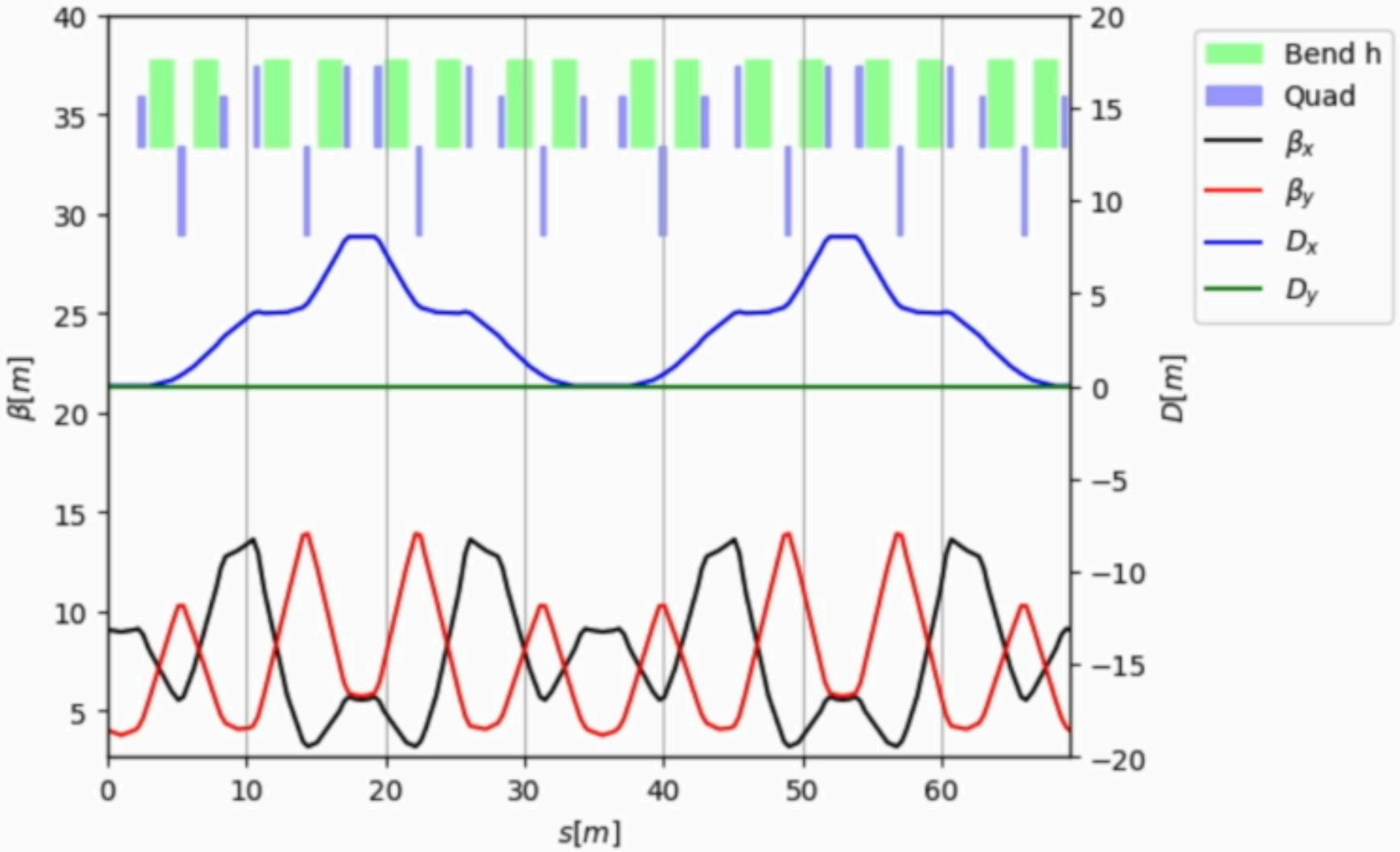
[ ]:
```
opt_chrom.target_status()
```

Observe deformation of phase space when exciting the resonance

2 ON OK kqd -10 -0.592287 0 -0.2 1e-05 1

[10]:
```
tw = ring.twiss4d()
pl = tw.plot()
pl.ylim(left_hi=40, right_lo=-20, right_hi=20, lattice_hi=1.5, lattice_lo=-7)
```

[10]: <xtrack.twissplot.TwissPlot object at 0x14da3b470>



Correct chromaticity

[ ]:
```
opt_chrom = ring.match(
    solve=False,
```

Correct chromaticity

```
[ ]: opt_chrom = ring.match(
         solve=False,
         method='4d',
         vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
         targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

```
[ ]: opt_chrom.run_ls_dogbox(30)
```

```
[ ]: opt_chrom.target_status()
```

Observe deformation of phase space when exciting the resonance

```
[ ]: # Generate 20 particles on the x axis
     x_gen = np.linspace(0, 0.5e-2, 20)
     p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

```
[ ]: # Set extraction sextupole
     env['kse'] = 0

     # Track 1000 turns
     ring.track(p0.copy(), num_turns=1000, turn_by_turn_monitor=True)
     rec = ring.record_last_track

     # Plot turn-by-turn data
     plt.figure()
     plt.plot(rec.x.T, rec.px.T, '.', markersize=1, color='C0')
     plt.xlabel(r'$x$ [m]'); plt.ylabel(r'$p_x$'), plt.xlim(-2e-2, 2e-2); plt.ylim(-2e-3, 2e-3)
     plt.subplots_adjust(left=.15)
```

```
[ ]:
```

Correct chromaticity

[11]:
```
opt_chrom = ring.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

[ ]:
```
opt_chrom.run_ls_dogbox(30)
```

[ ]:
```
opt_chrom.target_status()
```

Observe deformation of phase space when exciting the resonance

[ ]:
```
# Generate 20 particles on the x axis
x_gen = np.linspace(0, 0.5e-2, 20)
p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

[ ]:
```
# Set extraction sextupole
env['kse'] = 0

# Track 1000 turns
ring.track(p0.copy(), num_turns=1000, turn_by_turn_monitor=True)
rec = ring.record_last_track

# Plot turn-by-turn data
plt.figure()
plt.plot(rec.x.T, rec.px.T, '.', markersize=1, color='C0')
plt.xlabel(r'$x$ [m]'); plt.ylabel(r'$p_x$'), plt.xlim(-2e-2, 2e-2); plt.ylim(-2e-3, 2e-3)
plt.subplots_adjust(left=.15)
```

[ ]:

Correct chromaticity

[11]:
```
opt_chrom = ring.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

[12]:
```
opt_chrom.run_ls_dogbox(30)
```

```
Matching: model call n. 22 penalty = 7.9208e-11
```

[ ]:
```
opt_chrom.target_status()
```

Observe deformation of phase space when exciting the resonance

[ ]:
```
# Generate 20 particles on the x axis
x_gen = np.linspace(0, 0.5e-2, 20)
p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

[ ]:
```
# Set extraction sextupole
env['kse'] = 0

# Track 1000 turns
ring.track(p0.copy(), num_turns=1000, turn_by_turn_monitor=True)
rec = ring.record_last_track

# Plot turn-by-turn data
plt.figure()
plt.plot(rec.x.T, rec.px.T, '.', markersize=1, color='C0')
plt.xlabel(r'$x$ [m]'); plt.ylabel(r'$p_x$'), plt.xlim(-2e-2, 2e-2); plt.ylim(-2e-3, 2e-3)
plt.subplots_adjust(left=.15)
```

[ ]:

```
    targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

[12]:
```
opt_chrom.run_ls_dogbox(30)
```

```
Matching: model call n. 22 penalty = 7.9208e-11
```

[13]:
```
opt_chrom.target_status()
```

```
Target status:               nalty = 7.9208e-11
id state tag tol_met        residue   current_val    target_val description
0  ON    dqx    True   7.85383e-11         -0.01         -0.01 'dqx', val=-0.01, tol=0.001, weight=1
1  ON    dqy    True  -1.02796e-11         -0.01         -0.01 'dqy', val=-0.01, tol=0.001, weight=1
```

Observe deformation of phase space when exciting the resonance

[ ]:
```
# Generate 20 particles on the x axis
x_gen = np.linspace(0, 0.5e-2, 20)
p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

[ ]:
```
# Set extraction sextupole
env['kse'] = 0

# Track 1000 turns
ring.track(p0.copy(), num_turns=1000, turn_by_turn_monitor=True)
rec = ring.record_last_track

# Plot turn-by-turn data
plt.figure()
plt.plot(rec.x.T, rec.px.T, '.', markersize=1, color='C0')
plt.xlabel(r'$x$ [m]'); plt.ylabel(r'$p_x$'), plt.xlim(-2e-2, 2e-2); plt.ylim(-2e-3, 2e-3)
plt.subplots_adjust(left=.15)
```

[ ]:

```
    targets=xt.TargetSet(dqx=-0.01, dqy=-0.01, tol=1e-3))
```

[12]:
```
opt_chrom.run_ls_dogbox(30)
```

```
Matching: model call n. 22 penalty = 7.9208e-11
```

[13]:
```
opt_chrom.target_status()
```

```
Target status:               nalty = 7.9208e-11
id state tag tol_met       residue   current_val    target_val description
0  ON    dqx   True   7.85383e-11         -0.01         -0.01 'dqx', val=-0.01, tol=0.001, weight=1
1  ON    dqy   True  -1.02796e-11         -0.01         -0.01 'dqy', val=-0.01, tol=0.001, weight=1
```

Observe deformation of phase space when exciting the resonance

[14]:
```
# Generate 20 particles on the x axis
x_gen = np.linspace(0, 0.5e-2, 20)
p0 = ring.build_particles(x=x_gen, px=0, y=0, py=0, zeta=0, delta=0)
```

[ ]:
```
# Set extraction sextupole
env['kse'] = 0

# Track 1000 turns
ring.track(p0.copy(), num_turns=1000, turn_by_turn_monitor=True)
rec = ring.record_last_track

# Plot turn-by-turn data
plt.figure()
plt.plot(rec.x.T, rec.px.T, '.', markersize=1, color='C0')
plt.xlabel(r'$x$ [m]'); plt.ylabel(r'$p_x$'), plt.xlim(-2e-2, 2e-2); plt.ylim(-2e-3, 2e-3)
plt.subplots_adjust(left=.15)
```

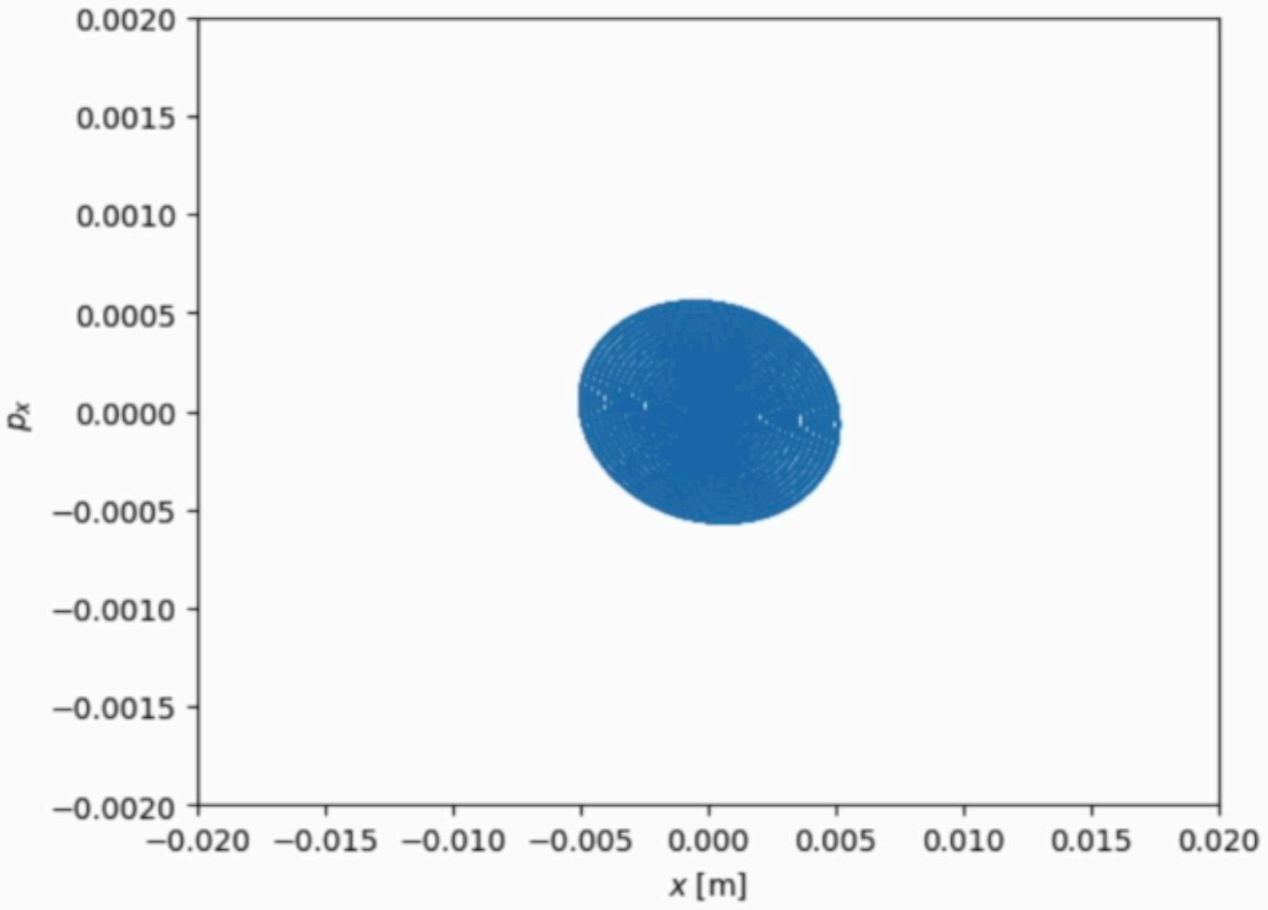
[ ]:

[15]:

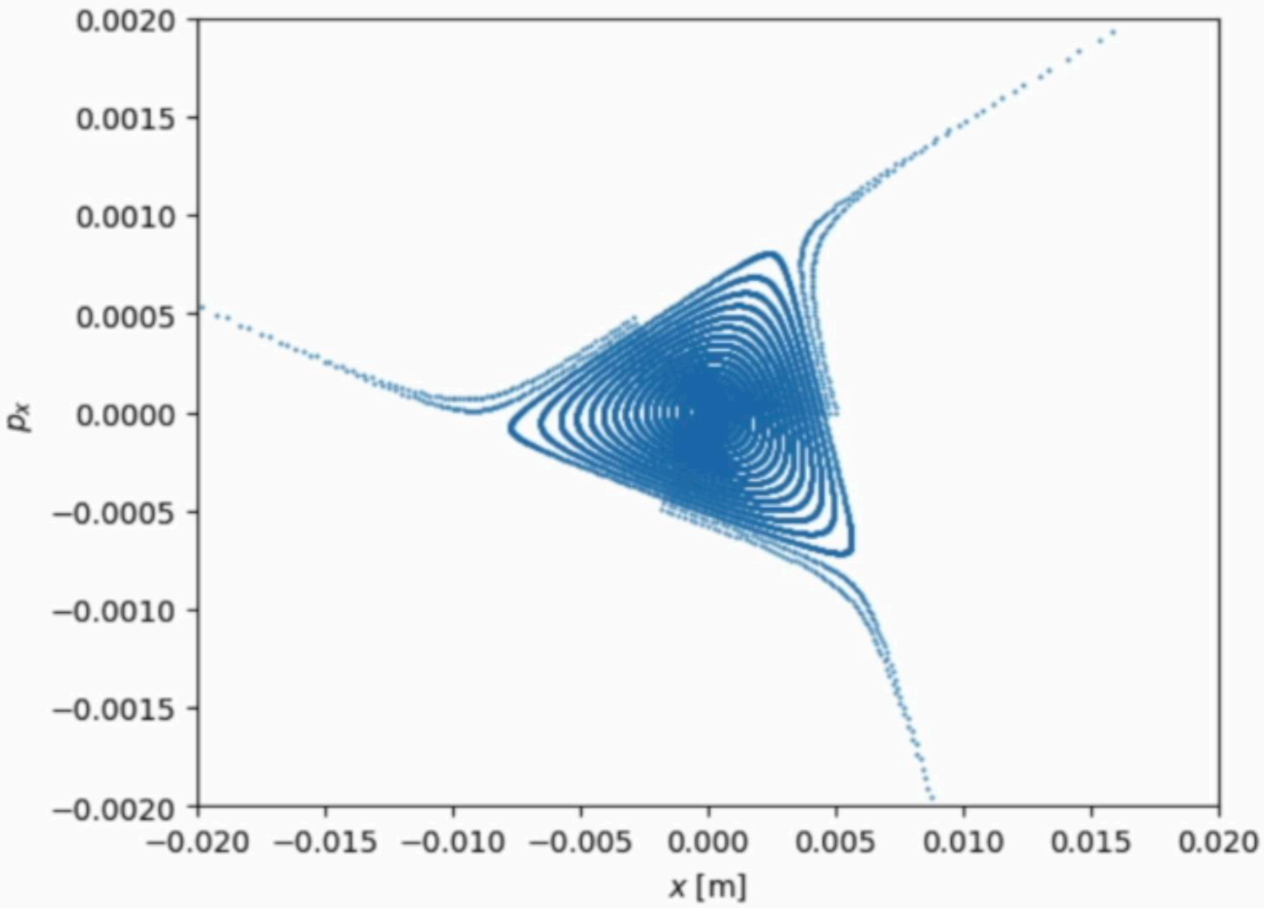
```python
# Set extraction sextupole
env['kse'] = 0

# Track 1000 turns
ring.track(p0.copy(), num_turns=1000, turn_by_turn_monitor=True)
rec = ring.record_last_track

# Plot turn-by-turn data
plt.figure()
plt.plot(rec.x.T, rec.px.T, '.', markersize=1, color='C0')
plt.xlabel(r'$x$ [m]'); plt.ylabel(r'$p_x$'), plt.xlim(-2e-2, 2e-2); plt.ylim(-2e-3, 2e-3)
plt.subplots_adjust(left=.15)
```



```
# Plot turn-by-turn data
plt.figure()
plt.plot(rec.x.T, rec.px.T, '.', markersize=1, color='C0')
plt.xlabel(r'$x$ [m]'); plt.ylabel(r'$p_x$'), plt.xlim(-2e-2, 2e-2); plt.ylim(-2e-3, 2e-3)
plt.subplots_adjust(left=.15)
```



[ ]:

Demo 3 – Instability

These demo files are adapted from https://github.com/xsuite/tutorial_cern_seminar.

Instability simulation

```
import matplotlib.pyplot as plt
import numpy as np
import xtrack as xt
import xpart as xp
import xwakes as xw
```

Import machine and twiss

```
# Import PIMM injection model (7 MeV; RF configured)
env = xt.Environment.from_json('./pimm.json')
env.vars.load_json('./pimm_strengths.json')

line = env.ring
line.configure_bend_model(num_multipole_kicks=5)

tw = line.twiss(method='4d')

# Enable RF
line['vrf'] = 10e3  # V
line['frf'] = 1 / tw.T_rev0
```

```
# Insert aperture limitation to see losses
line.discard_tracker()
line.append_element(name='aperture', element=xt.LimitRect(min_y=-0.03, max_y=0.03))
```

Install transverse wakefields

```
wf = xw.WakeResonator(kind='dipolar_y',
                      r=200e6, # Shunt impedance [Ohm/m]
                      f_r=1.3e6,
```

Instability simulation

```
[1]: import matplotlib.pyplot as plt
import numpy as np
import xtrack as xt
import xpart as xp
import xwakes as xw
```

Import machine and twiss

```
[ ]: # Import PIMM injection model (7 MeV; RF configured)
env = xt.Environment.from_json('./pimm.json')
env.vars.load_json('./pimm_strengths.json')

line = env.ring
line.configure_bend_model(num_multipole_kicks=5)

tw = line.twiss(method='4d')

# Enable RF
line['vrf'] = 10e3  # V
line['frf'] = 1 / tw.T_rev0
```

```
[ ]: # Insert aperture limitation to see losses
line.discard_tracker()
line.append_element(name='aperture', element=xt.LimitRect(min_y=-0.03, max_y=0.03))
```

Install transverse wakefields

```
[ ]: wf = xw.WakeResonator(kind='dipolar_y',
                       r=200e6, # Shunt impedance [Ohm/m]
                       f_r=1.3e6,
```

Instability simulation

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import xtrack as xt
import xpart as xp
import xwakes as xw
```

Import machine and twiss

[2]:
```
# Import PIMM injection model (7 MeV; RF configured)
env = xt.Environment.from_json('./pimm.json')
env.vars.load_json('./pimm_strengths.json')

line = env.ring
line.configure_bend_model(num_multipole_kicks=5)

tw = line.twiss(method='4d')

# Enable RF
line['vrf'] = 10e3  # V
line['frf'] = 1 / tw.T_rev0
```

Loading line from dict: 100% 148/148 [00:00<00:00, 7572.33it/s]

```
Done loading line from dict.
```

[ ]:
```
# Insert aperture limitation to see losses
line.discard_tracker()
line.append_element(name='aperture', element=xt.LimitRect(min_y=-0.03, max_y=0.03))
```

Install transverse wakefields

Install transverse wakefields

```
[ ]: wf = xw.WakeResonator(kind='dipolar_y',
                          r=200e6, # Shunt impedance [Ohm/m]
                          f_r=1.3e6,
                          q=1.)
     wf.configure_for_tracking(zeta_range=(-20, 20), num_slices=20)
```

```
[ ]: # Install wakefield in the Xsuite ring
     line.append('wf', obj=wf)
```

Generate a bunch matched to the bucket and to the optics

```
[ ]: %%capture
     line.build_tracker()

     bunch = xp.generate_matched_gaussian_bunch(line=line, num_particles=1000,
         total_intensity_particles=1e11, nemitt_x=1e-6, nemitt_y=1e-6, sigma_z=10.)

     # Initial kick of 1 mm
     bunch.y += 1e-3

     # Keep initial state
     bunch0 = bunch.copy()
```

Simulate 1000 turns

```
[ ]: # Define quantities to be logged during tracking
     def compute_y_mean(line, particles):
         bunch.hide_lost_particles()
         y_ave = np.mean(particles.y)
         bunch.unhide_lost_particles()
         return y_ave
```

Install transverse wakefields

```
wf = xw.WakeResonator(kind='dipolar_y',
                      r=200e6, # Shunt impedance [Ohm/m]
                      f_r=1.3e6,
                      q=1.)
wf.configure_for_tracking(zeta_range=(-20, 20), num_slices=20)
```

```
# Install wakefield in the Xsuite ring
line.append('wf', obj=wf)
```

Generate a bunch matched to the bucket and to the optics

```
%%capture
line.build_tracker()

bunch = xp.generate_matched_gaussian_bunch(line=line, num_particles=1000,
    total_intensity_particles=1e11, nemitt_x=1e-6, nemitt_y=1e-6, sigma_z=10.)

# Initial kick of 1 mm
bunch.y += 1e-3

# Keep initial state
bunch0 = bunch.copy()
```

Simulate 1000 turns

```
# Define quantities to be logged during tracking
def compute_y_mean(line, particles):
    bunch.hide_lost_particles()
    y_ave = np.mean(particles.y)
    bunch.unhide_lost_particles()
    return y_ave
```

Install transverse wakefields

[4]:
```
wf = xw.WakeResonator(kind='dipolar_y',
                      r=200e6, # Shunt impedance [Ohm/m]
                      f_r=1.3e6,
                      q=1.)
wf.configure_for_tracking(zeta_range=(-20, 20), num_slices=20)
```

[5]:
```
# Install wakefield in the Xsuite ring
line.append('wf', obj=wf)
```

Generate a bunch matched to the bucket and to the optics

[ ]:
```
%%capture
line.build_tracker()

bunch = xp.generate_matched_gaussian_bunch(line=line, num_particles=1000,
    total_intensity_particles=1e11, nemitt_x=1e-6, nemitt_y=1e-6, sigma_z=10.)

# Initial kick of 1 mm
bunch.y += 1e-3

# Keep initial state
bunch0 = bunch.copy()
```

Simulate 1000 turns

[ ]:
```
# Define quantities to be logged during tracking
def compute_y_mean(line, particles):
    bunch.hide_lost_particles()
    y_ave = np.mean(particles.y)
    bunch.unhide_lost_particles()
    return y_ave
```

Simulate 1000 turns

[ ]:
```
# Define quantities to be logged during tracking
def compute_y_mean(line, particles):
    bunch.hide_lost_particles()
    y_ave = np.mean(particles.y)
    bunch.unhide_lost_particles()
    return y_ave

def compute_intensity(line, particles):
    bunch.hide_lost_particles()
    inten = np.sum(particles.weight)
    bunch.unhide_lost_particles()
    return inten

track_log = xt.Log(y_mean=compute_y_mean, intensity=compute_intensity)
```

[ ]:
```
# Track!
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

[ ]:
```
# Plot logged data
y_mean = line.log_last_track['y_mean']
intensity = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean)
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity)
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')

plt.subplots_adjust(left=0.2, hspace=0.3, top=0.9)
```

Simulate 1000 turns

[7]:
```python
# Define quantities to be logged during tracking
def compute_y_mean(line, particles):
    bunch.hide_lost_particles()
    y_ave = np.mean(particles.y)
    bunch.unhide_lost_particles()
    return y_ave

def compute_intensity(line, particles):
    bunch.hide_lost_particles()
    inten = np.sum(particles.weight)
    bunch.unhide_lost_particles()
    return inten

track_log = xt.Log(y_mean=compute_y_mean, intensity=compute_intensity)
```

[ ]:
```python
# Track!
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

[ ]:
```python
# Plot logged data
y_mean = line.log_last_track['y_mean']
intensity = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean)
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity)
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')

plt.subplots_adjust(left=0.2, hspace=0.3, top=0.9)
```

[ ]:
```python
# Plot logged data
y_mean = line.log_last_track['y_mean']
intensity = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean)
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity)
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')

plt.subplots_adjust(left=0.2, hspace=0.3, top=0.9)
```

Adjust the sextupoles to change the chromaticity

[ ]:
```python
%%capture
opt = line.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-6., dqy=-6., tol=1e-3, tag="chrom")
)
opt.solve()
```

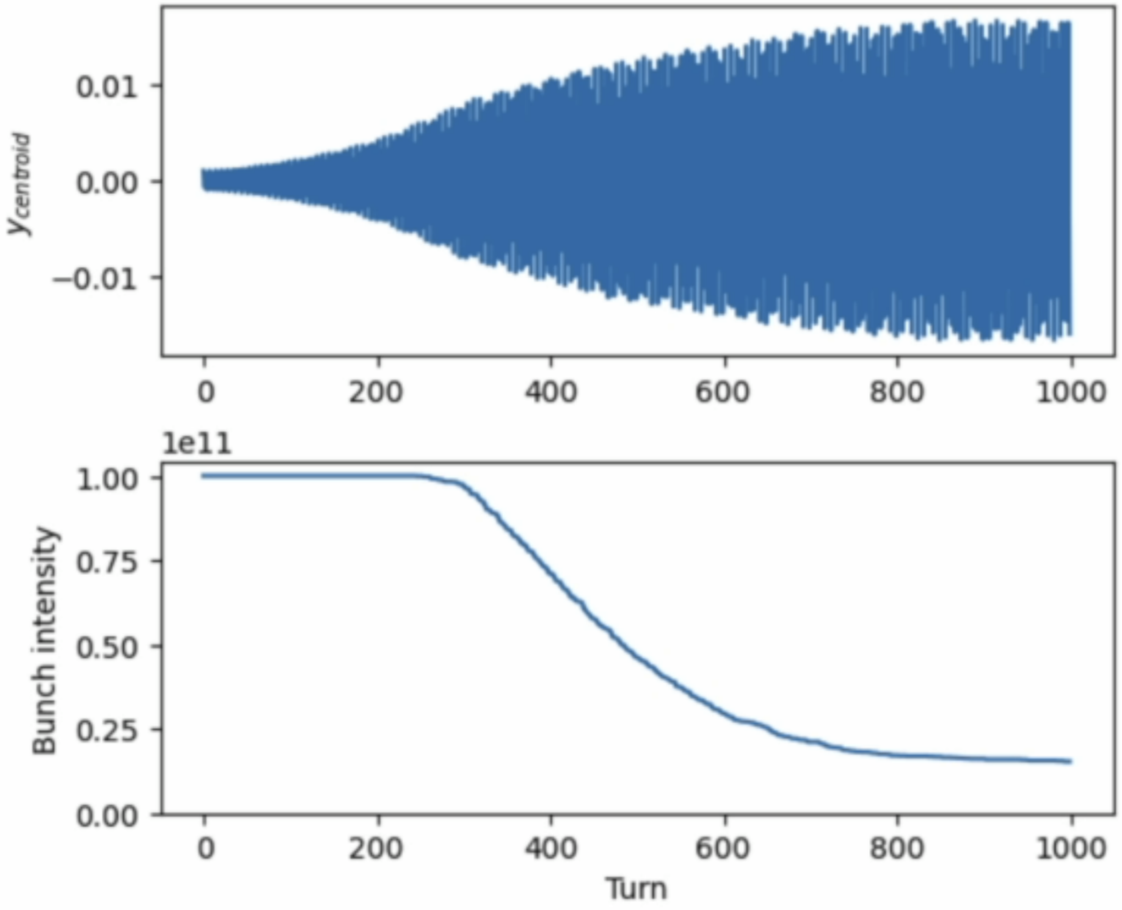
Repeat the simulation

[ ]:
```python
bunch = bunch0.copy()
bunch.x += 1e-3
```

[ ]:
```python
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

```
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean)
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity)
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')

plt.subplots_adjust(left=0.2, hspace=0.3, top=0.9)
```



Adjust the sextupoles to change the chromaticity

Adjust the sextupoles to change the chromaticity

```
%%capture
opt = line.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-6., dqy=-6., tol=1e-3, tag="chrom")
)
opt.solve()
```

Repeat the simulation

```
bunch = bunch0.copy()
bunch.x += 1e-3
```

```
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

```
# Plot logged data
y_mean_high_chroma = line.log_last_track['y_mean']
intensity_high_chroma = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean, label="Q' = -0.1")
plt.plot(y_mean_high_chroma, label="Q' = -2.0")
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity, label="Q' = -0.1")
plt.plot(intensity_high_chroma, label="Q' = -2.0")
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')
plt.legend()
```

Adjust the sextupoles to change the chromaticity

[10]:
```
%%capture
opt = line.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-6., dqy=-6., tol=1e-3, tag="chrom")
)
opt.solve()
```

Repeat the simulation

[ ]:
```
bunch = bunch0.copy()
bunch.x += 1e-3
```

[ ]:
```
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

[ ]:
```
# Plot logged data
y_mean_high_chroma = line.log_last_track['y_mean']
intensity_high_chroma = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean, label="Q' = -0.1")
plt.plot(y_mean_high_chroma, label="Q' = -2.0")
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity, label="Q' = -0.1")
plt.plot(intensity_high_chroma, label="Q' = -2.0")
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')
plt.legend()
```

Adjust the sextupoles to change the chromaticity

[10]:
```
%%capture
opt = line.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-6., dqy=-6., tol=1e-3, tag="chrom")
)
opt.solve()
```

Repeat the simulation

[11]:
```
bunch = bunch0.copy()
bunch.x += 1e-3
```

[ ]:
```
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

[ ]:
```
# Plot logged data
y_mean_high_chroma = line.log_last_track['y_mean']
intensity_high_chroma = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean, label="Q' = -0.1")
plt.plot(y_mean_high_chroma, label="Q' = -2.0")
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity, label="Q' = -0.1")
plt.plot(intensity_high_chroma, label="Q' = -2.0")
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')
plt.legend()
```

Adjust the sextupoles to change the chromaticity

[10]:
```
%%capture
opt = line.match(
    solve=False,
    method='4d',
    vary=xt.VaryList(['ksf', 'ksd'], step=1e-3),
    targets=xt.TargetSet(dqx=-6., dqy=-6., tol=1e-3, tag="chrom")
)
opt.solve()
```

Repeat the simulation

[11]:
```
bunch = bunch0.copy()
bunch.x += 1e-3
```

[12]:
```
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

Tracking: 100% 1000/1000 [00:18<00:00, 52.96it/s]

[ ]:
```
# Plot logged data
y_mean_high_chroma = line.log_last_track['y_mean']
intensity_high_chroma = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean, label="Q' = -0.1")
plt.plot(y_mean_high_chroma, label="Q' = -2.0")
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity, label="Q' = -0.1")
plt.plot(intensity_high_chroma, label="Q' = -2.0")
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
```

```
    targets=xt.TargetSet(dqx=-6., dqy=-6., tol=1e-3, tag="chrom")
)
opt.solve()
```

Repeat the simulation

[11]:
```
bunch = bunch0.copy()
bunch.x += 1e-3
```

[12]:
```
line.track(bunch, log=track_log, num_turns=1000, with_progress=10)
```

Tracking: 100% 1000/1000 [00:18<00:00, 52.96it/s]

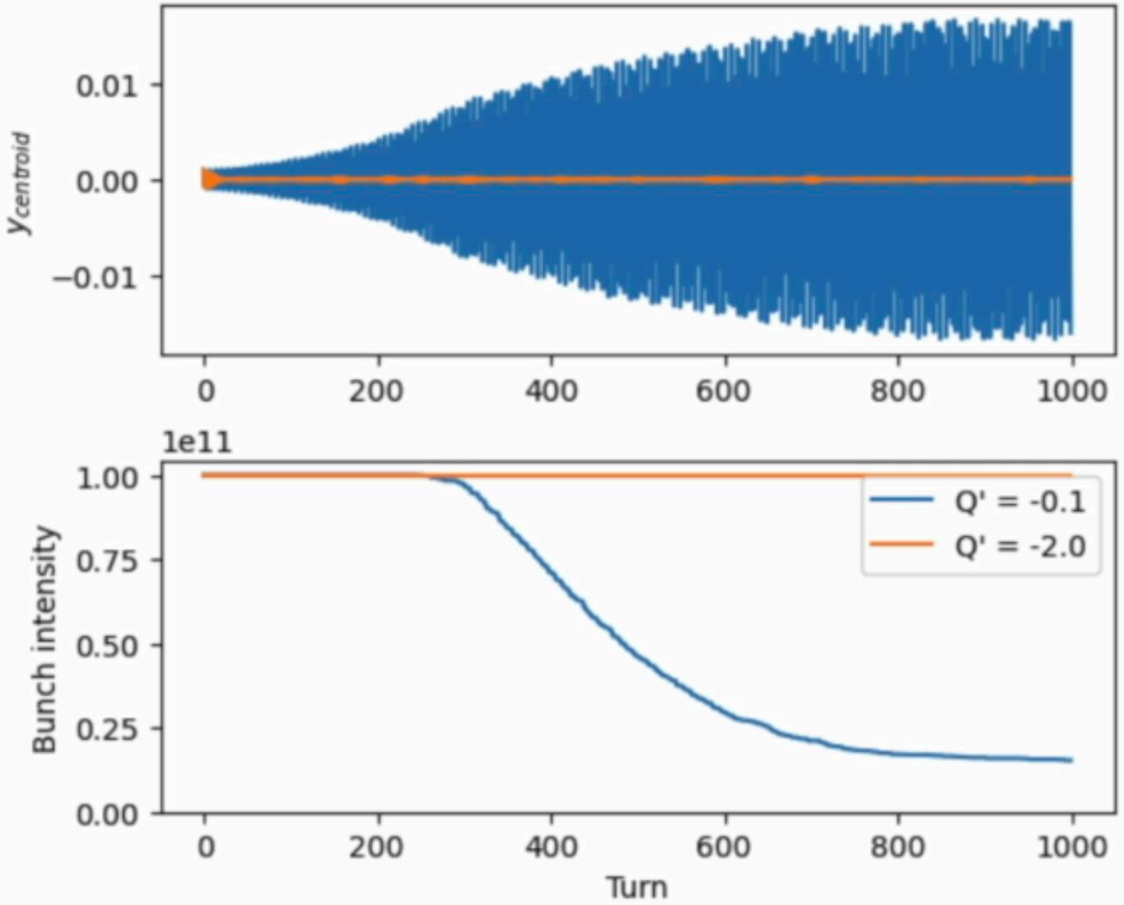
[ ]:
```
# Plot logged data
y_mean_high_chroma = line.log_last_track['y_mean']
intensity_high_chroma = line.log_last_track['intensity']

plt.figure()
ax1 = plt.subplot(2,1,1)
plt.ylabel(r'$y_{centroid}$')
plt.plot(y_mean, label="Q' = -0.1")
plt.plot(y_mean_high_chroma, label="Q' = -2.0")
ax2 = plt.subplot(2,1,2, sharex=ax1)
plt.plot(intensity, label="Q' = -0.1")
plt.plot(intensity_high_chroma, label="Q' = -2.0")
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')
plt.legend()

plt.subplots_adjust(left=0.2, hspace=0.3, top=0.9)
```

```python
plt.plot(intensity, label="Q' = -0.1")
plt.plot(intensity_high_chroma, label="Q' = -2.0")
plt.ylim(bottom=0)
plt.ylabel('Bunch intensity')
plt.xlabel('Turn')
plt.legend()

plt.subplots_adjust(left=0.2, hspace=0.3, top=0.9)
```



[ ]:

Thank you!

xsuite.web.cern.ch