RF-Track
Beam tracking in complex accelerator scenarios

Andrea Latina
andrea.latina@cern.ch

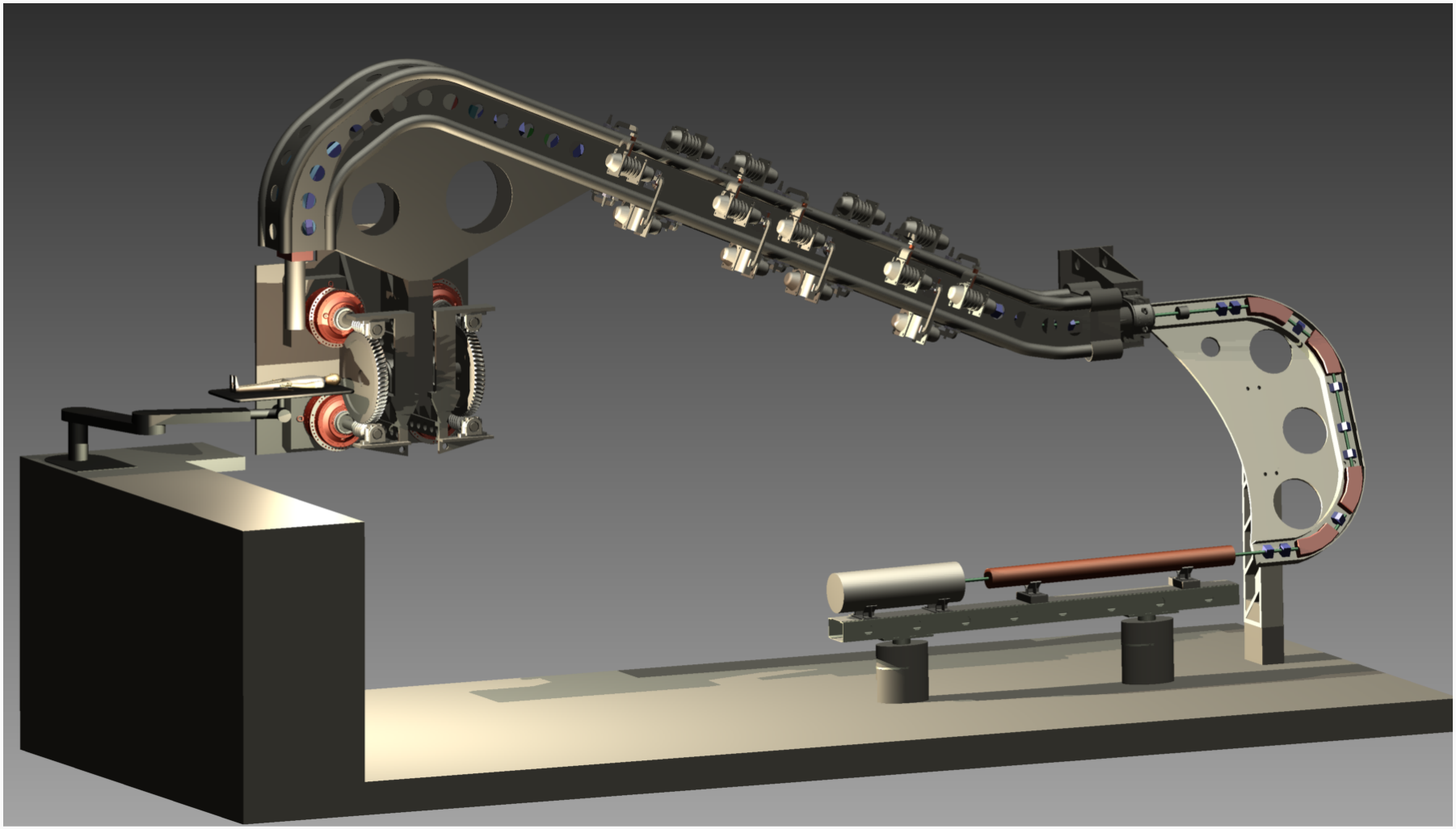
Muon Collider Design meeting - 17 October 2022

Table of Contents

Introduction

Motivation for developing RF-Track

The TULIP Project: High-gradient proton linac for proton therapy



S. Benedetti, A. Grudiev, and A. Latina, Phys. Rev. Accel. Beams 20, 040101

Motivations for RF-Track: simulation requirements

To design and optimise the TULIP linac, which used a new high-gradient (50 MV/m) S-band backward travelling-wave accelerating structure with initial $\beta = 0.38$:

- We needed 6-D tracking in 3-D electromagnetic field maps of **backward traveling wave** structures
- We needed the possibility to maximise the transmission changing **any** parameter: **RF input power**, quadrupole strengths, quadrupole positions, input distribution, etc.
- We needed to track **protons** as well as **carbon ions**
- We needed something **flexible** and **fast** to enable non trivial optimisations
- We decided to develop a new code from scratch

RF-Track highlights

- It handles $\mathbb{C}$omplex 3-D field maps of oscillating radio-frequency electro-magnetic fields:
  - with the capability of simulating of Backward and Forward travelling waves, as well as static fields
- It's fully relativistic
  - no approximations are made, like $\beta \simeq 1$ or $\gamma \gg 1$
  - Can simulated any particle, and was successfully used with: electrons, positrons, protons, antiprotons, ions, at various energies, recently photons and muons
- Implements high-order integration algorithms
- Implements space-charge effects, wakefields
- Tracks mixed-species beams, with collective effects
- It's flexible and fast

RF-Track internals

- RF-Track was written in modern C++, as a fast, parallel module to be loaded and used from within:

RF-Track internals

- RF-Track was written in modern C++, as a fast, parallel module to be loaded and used from within:

  Octave, a "programming language for scientific computing". It is software featuring a high-level programming language, primarily intended for numerical computations. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB.
  `https://www.octave.org`
  `https://octave.sourceforge.io`

RF-Track internals

- RF-Track was written in modern C++, as a fast, parallel module to be loaded and used from within:

  Octave, a "programming language for scientific computing". It is software featuring a high-level programming language, primarily intended for numerical computations. Octave helps in solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB.
  `https://www.octave.org`
  `https://octave.sourceforge.io`

  or

  Python, a general purpose programming language created by Guido Van Rossum. Using libraries such as numpy, matplotlib, pandas offer many functionalities that make it similar to MATLAB and Octave.
  `https://www.python.org`

RF-Track: minimalistic and physics-oriented

RF-Track only contains C++ code that provides accelerator-physics concepts:

- Flexible accelerator and beam models
- Accurate integration of the equations of motion
- Interpolation of field maps which is Maxwell's-equations-compliant
- Collective effects

For "all the rest" (e.g., integration algorithms, random number generation, etc.), it relies on two robust and renowned open-source libraries:

- GSL, "Gnu Scientific Library", provides a wide range of mathematical routines such as high-quality random number generators, ODE integrators, linear algebra, and much more
- FFTW, "Fastest Fourier Transform in the West", arguably the fastest open-source library to compute discrete Fourier transforms

RF-Track example, in Octave

```
% load RF-Track
RF_Track;

% create a bunch from phase-space matrix
B0 = Bunch6d(electronmass, 200 * pC, -1, phase_space_matrix);

% create a lattice (1 FODO cell)
Lq = 0.4; % m
Ld = 0.6; % m
G = 1.2; % T/m

FODO = Lattice();
FODO.append (Quadrupole (Lq, G));
FODO.append (Drift (Ld));
FODO.append (Quadrupole (Lq, -G));
FODO.append (Drift (Ld));

% track the beam
B1 = FODO.track(B0);

% plot the phase space
T1 = B1.get_phase_space("%x %xp %y %yp");
scatter (T1(:,1), T1(:,2), "*");
xlabel ("x [mm]");
ylabel ("x' [mrad]");
```

Beam models: tracking in space and in time

1. Beam moving in space: `Bunch6d()`
   - All particles have the same S position
   - Each particle's phase space is

$$(x \text{ [mm]},\ x' \text{ [mrad]},\ y \text{ [mm]},\ y' \text{ [mrad]},\ t \text{ [mm/c]},\ P \text{ [MeV/c]})$$

   Integrates the equations of motion in dS:

$$S \rightarrow S + dS$$

   (moves the beam element by element)

Beam models: tracking in space and in time

1. Beam moving in space: `Bunch6d()`
   - All particles have the same S position
   - Each particle's phase space is

$$(x \text{ [mm]},\ x' \text{ [mrad]},\ y \text{ [mm]},\ y' \text{ [mrad]},\ t \text{ [mm/c]},\ P \text{ [MeV/c]})$$

   Integrates the equations of motion in dS:

$$S \rightarrow S + dS$$

   (moves the beam element by element)

2. Beam moving in time: `Bunch6dT()`
   - All particles are considered at same time t
   - Each particle's phase space is

$$(X \text{ [mm]},\ Y \text{ [mm]},\ S \text{ [mm]},\ P_x \text{ [MeV/c]},\ P_y \text{ [MeV/c]},\ P_z \text{ [MeV/c]})$$

   - Handles particles with $P_z < 0$ or even $P_z = 0$: particles can move backward
   - Integrates the equations of motion in dt:

$$t \rightarrow t + dt$$

Beam models: tracking in space and in time

1. Beam moving in space: `Bunch6d()`
   - All particles have the same S position
   - Each particle's phase space is

$$(x \text{ [mm]},\ x' \text{ [mrad]},\ y \text{ [mm]},\ y' \text{ [mrad]},\ t \text{ [mm/c]},\ P \text{ [MeV/c]})$$

   Integrates the equations of motion in dS:

$$S \to S + dS$$

   (moves the beam element by element)

2. Beam moving in time: `Bunch6dT()`
   - All particles are considered at same time t
   - Each particle's phase space is

$$(X \text{ [mm]},\ Y \text{ [mm]},\ S \text{ [mm]},\ P_x \text{ [MeV/c]},\ P_y \text{ [MeV/c]},\ P_z \text{ [MeV/c]})$$

   - Handles particles with $P_z < 0$ or even $P_z = 0$: particles can move backward
   - Integrates the equations of motion in dt:

$$t \to t + dt$$

- Each particle also stores

$$m: \text{ mass [MeV/c}^2\text{]}, \quad Q: \text{ charge } [e^+]$$

$$N: \text{ nb of particles / macroparticle}, \quad t_0: \text{ creation time}^{(\star)} \qquad \tau: \text{ lifetime}$$

  $^{(\star)}$ only for beams moving in time

- **Note:** RF-Track can simulate mixed-specie beams and particle's creation

Tracking: Integration algorithms

- The default is: "**leapfrog**": fast, second-order accurate, "symplectic"

- Higher-order, adaptive algorithms provided by GSL:
  - ⋆"**rk2**" Runge-Kutta (2, 3)
  - ⋆"**rk4**" 4th order (classical) Runge-Kutta
  - ⋆"**rkf45**" Runge-Kutta-Fehlberg (4, 5)
  - ⋆"**rkck**" Runge-Kutta Cash-Karp (4, 5)
  - ⋆"**rk8pd**" Runge-Kutta Prince-Dormand (8, 9)
  - ⋆"**msadams**" multistep Adams in Nordsieck form; order varies dynamically between 1 and 12

- Analytic algorithm:
  - ⋆"**analytic**" integration of the equations of motion assuming a locally-constant EM field.

The beam can be tracked backward in time. Including collective effects.

User interface: flexible input / output

- Input / output:
  - Input / output through Octave or Python: ASCII files, binary files, HDF5
  - Specific to RF-Track: can save beam data in DST (PlotWin) and SDDS formats
  - Automatic compression / decompression of files (e.g. field maps)

- Retrieving particles phase space with flexibility:
  - PLACET style

```
T1 = B1.get_phase_space("%E %x %y %dt %xp %yp");
```

  - MAD-X style

```
T1 = B1.get_phase_space("%x %px %y %py %Z %pt");
```

  - TRANSPORT style

```
T1 = B1.get_phase_space("%x %xp %y %yp %dt %d");
```

User interface: inquiring the phase space

One can retrieve the phase space with great flexibility: e.g.

```
P1 = B1.get_phase_space(%x %Px %y %Py %deg@750 %K);
```

%x	horiz. position at S	mm	%X	horiz. position at $t=t_0$	mm
%y	vert. position at S	mm	%Y	vert. position at $t=t_0$	mm
%xp	horiz. angle	mrad	%t	proper time	mm/c
%yp	vert. angle	mrad	%dt	delay = $t - t_0$	mm/c
%Vx	velocity	c	%z	$S/beta_0 - c.t = c(t_0 - t)$	mm
%Vy	velocity	c	%Z	–%dt * %Vz	mm
%Vz	velocity	c	%S	S + %Z	m
%Px	momentum	MeV/c	%deg@f	degrees @*freq* [MHz]	deg
%Py	momentum	MeV/c	%d	relative momentum	per mille
%Pz	momentum	MeV/c	%pt	$(\%E - E_0) / P_0c$	per mille
%px	$\%Px/P_0$	mrad	%P	total momentum	MeV/c
%py	$\%Py/P_0$	mrad	%E	total energy	MeV
%pz	$\%Pz/P_0$	mrad	%K	kinetic energy	MeV

All relative quantities use the first particle in the particles array as reference, or the average particle if the first is lost.

Overview: tracking global beam properties

After tracking over a lattice "L", one can retrieve the average beam properties:

```
T1 = L.get_transport_table("%S %beta_x %beta_y %mean_K %N");
```

%sigma_x|xp|y|yp|t|P	Std(#)	various
%sigma_xxp|yyp|tP	Cov(#,#)	various
%mean_x|xp|y|yp|t|P|K|E	Mean(#)	various
%alpha_x|y|z	Twiss	1
%beta_x|y|z	Twiss	see below
%rmax	envelope	mm
%emitt_x|y|z	RMS emittance	see below
%S	position	m
%N	Transmission	percent

β_x [m/rad]	$\epsilon_x = \epsilon_{x,\,\mathrm{geom}}\beta_{\mathrm{rel}}\gamma_{\mathrm{rel}}$ [mm.mrad]	$\epsilon_{x,\,\mathrm{geom}} = \sigma_x\sigma_{x'}$	$\sigma_x = \sqrt{\epsilon_{x,\,\mathrm{geom}}\beta_x}$ [mm]
β_y [m/rad]	$\epsilon_y = \epsilon_{y,\,\mathrm{geom}}\beta_{\mathrm{rel}}\gamma_{\mathrm{rel}}$ [mm.mrad]	$\epsilon_{y,\,\mathrm{geom}} = \sigma_y\sigma_{y'}$	$\sigma_y = \sqrt{\epsilon_{y,\,\mathrm{geom}}\beta_y}$ [mm]
β_z [m]	$\epsilon_z = \epsilon_{z,\,\mathrm{geom}}\beta_{\mathrm{rel}}\gamma_{\mathrm{rel}}$ [mm.permil]	$\epsilon_{z,\,\mathrm{geom}} = \sigma_Z\sigma_\delta$	$\sigma_z = \sqrt{\epsilon_{z,\,\mathrm{geom}}\beta_z}$ [mm]

Beam Tracking

Tracking environments

RF-Track offers two tracking environments, "**Volume**" and "**Lattice**".

Lattice: a list of elements

- It uses Bunch6d to track the particles element-by-element, *along S*
- Like in textbooks, suitable for matrix-based tracking

Volume: a portion of 3-D space

- It uses Bunch6dT to track the particles *in time*
- Elements can be placed at arbitrary locations in the volume
  - position + Euler angles (raw, pitch, yaw)
- Allows element overlap
- Allows particles creation
  - Can simulate cathodes and field emission
  - Considers the effect of mirror charges at the cathode

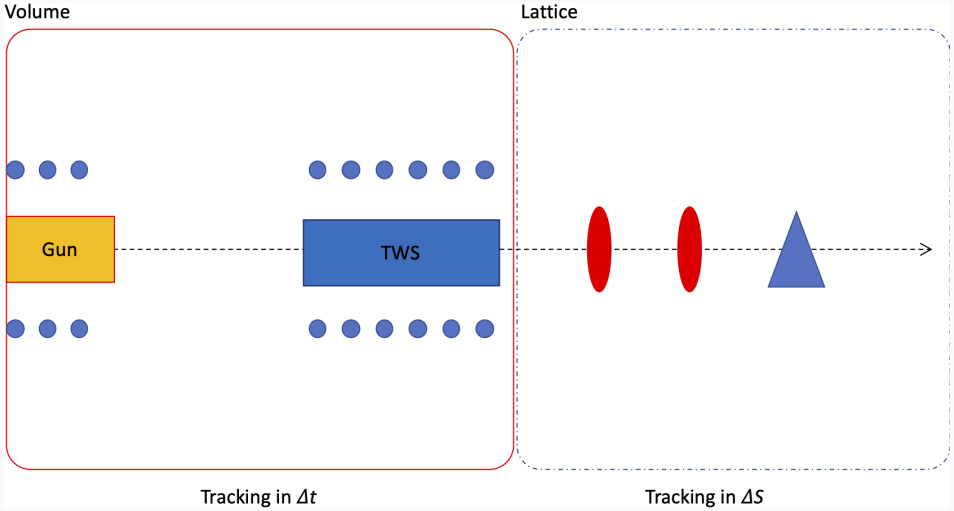
Tracking with Lattice()

Each element:

- Can be tracked in several steps, to capture phase space evolution along the element
  - Twiss parameters, emittances, and average quantities are tracked
  - Phase space is saved as 'gzipped' files
- Can have an aperture, which is checked at each integration step
  - The user can retrieve the full phase space coordinates if each lost particle, as well as the 3-D position and time at which it was lost
- One can identify what particles of the initial distribution are lost

Tracking with Volume()

Lattice and Volume are meant to be used together:



(SC-dominated regimes)

Example:

```
V = Volume();
V.add (Element, X, Y, Z, roll, pitch, yaw, "reference");
```

- X, Y, Z: arbitrary position in the 3-D space
- roll, pitch, yaw: Euler angles
- reference point: "center", "entrance", "exit"

Volume's TrackingOptions

Tracking is performed via the command:

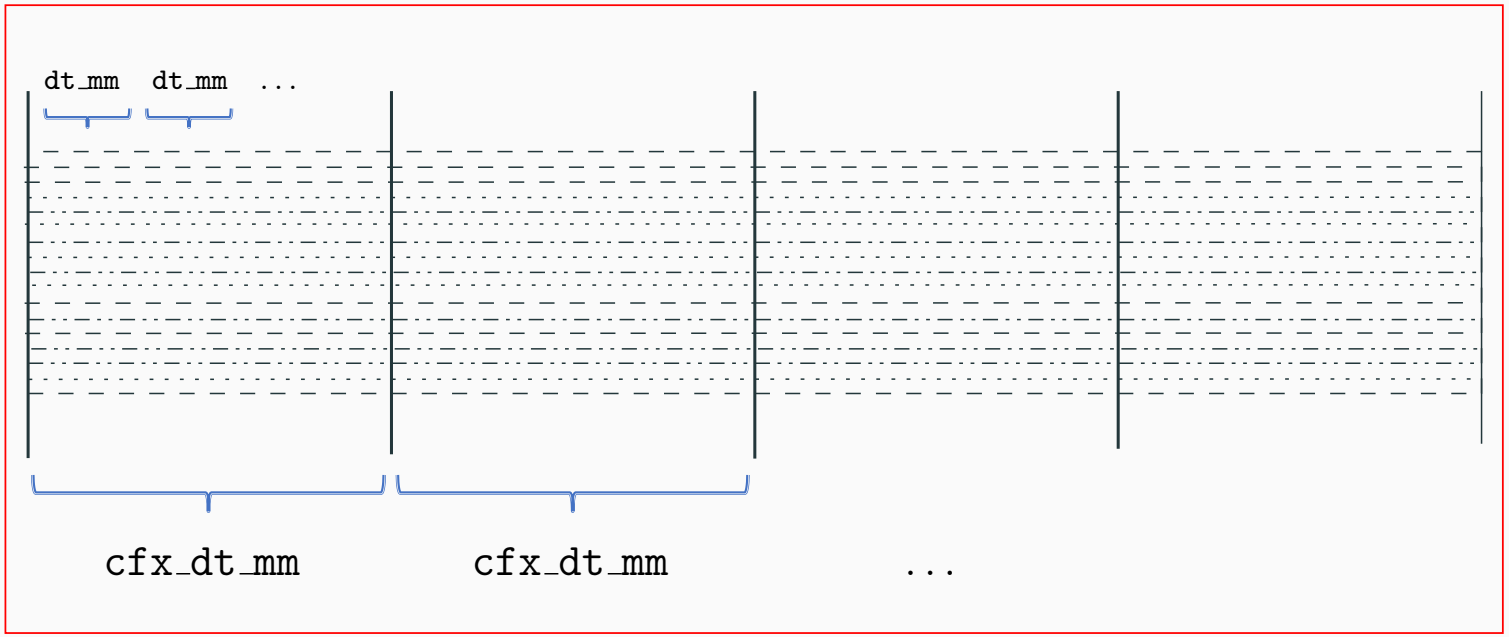
```
B1 = V.track(B0, O);
```

where "O" are tracking options:

```
O = TrackingOptions();
O.odeint_algorithm = "leapfrog"; % built for speed
O.odeint_epsabs = 1.0; % applies only to GSL algorithms
O.odeint_epsrel = 0.0; % applies only to GSL algorithms
O.dt_mm = Inf; % mm/c, integration step (Inf means: set automatically)
O.t_max_mm = Inf; % track until t<t_max_mm, if '+inf' tracks all the way to the end
O.t_min_mm = -Inf; % track until t>t_min_mm, if '-inf' tracks all the way to the beginning
O.sc_dt_mm = 0.0; % apply one space-charge kick every sc_dt_mm
O.cfx_dt_mm = 0.0; % apply collective effects every cfx_dt_mm
O.tt_dt_mm = 0.0; % tabulate the bunch parameters every tt_dt_mm
O.wp_dt_mm = 0.0; % (watch point) save the beam on a file every wp_dt_mm
O.wp_basename = "watch_beam"; % save the beam distributions with name watch_beam.0000n.txt
O.wp_gzip = false; % gzip files
O.verbosity = 0; % verbosity level, 0 = silent
```

Volume: Staggered integration

Beam is tracked in parallel between consecutive kicks of collective effects:



Volume

Staggered integration:

- Fine: Integration in 'dt_mm' is performed in parallel using high-order adaptive integration: 'rk2', 'rk4', 'rkf45', 'rk8pd', 'msadams', 'leapfrog'
- Coarse: Integration in 'cfx_dt_mm' is performed on the whole beam, using fixed-step-size leapfrog, and it's meant for collective effects

Beam line elements

1. **Standard** set of matrix-based symplectic elements:
   - Sector Bends (standard matrix-based)
   - Quadrupole (standard matrix-based)
   - Drift (with an optional constant electric and magnetic fields, can be used to simulate e.g., rbends, or solenoids)

Beam line elements

1. **Standard** set of matrix-based symplectic elements:
   - Sector Bends (standard matrix-based)
   - Quadrupole (standard matrix-based)
   - Drift (with an optional constant electric and magnetic fields, can be used to simulate e.g., rbends, or solenoids)

2. **Field maps** (see next slides)

Beam line elements

1. **Standard** set of matrix-based symplectic elements:
   - Sector Bends (standard matrix-based)
   - Quadrupole (standard matrix-based)
   - Drift (with an optional constant electric and magnetic fields, can be used to simulate e.g., rbends, or solenoids)

2. **Field maps** (see next slides)

3. **Special elements**:
   - 3-D Analytic Coils
   - 3-D Analytic Solenoid
   - 3-D Analytic Standing wave and Traveling wave structures
   - Adiabatic matching devices, Toroidal Harmonics, LaserBeams (ICS)
   - Twiss table: tracks through an arbitrary lattice, given a table of Twiss parameters, phase advances, momentum compaction, 1^{st} and 2^{nd} order chromaticity

Field maps

RF-Track supports several types of field maps:

- 1-D (on-axis) field maps of oscillating traveling-wave electric fields
  - It uses the Maxwell's equations to reconstruct the 3-D fields: E_r, E_ϕ as well as B_r and B_ϕ in points off-axis
- 2-D fields maps: field on a plane and applies cylindrical symmetry
- 3-D field maps of oscillating electro-magnetic fields
  - It accepts 3-D meshes of complex numbers (see next slide for more details)
- StaticElectric and StaticMagnetic fields
  - Dedicated implementation to provide curl-free (electric) and divergence-free (magnetic) interpolation of the field map

3-D field maps

- Complex EM maps of RF fields
  - forward traveling / backward traveling / static fields
  - trilinear / tricubic interpolation
- Accepts quarter/half field maps
  - automatic mirroring of the fields
  - accepts cartesian and cylindrical maps
- Can change dynamically input power
  - Provide P_{map}, set P_{actual}
- Not-a-number's are considered as walls
  - allows one to precisely track losses in the 3-D volume
- One can retrieve the fields at any point: e.g.
  `[E,B] = RFQ.get_field(x,y,z,t);`

Example:

```
load 'field.dat.gz';

RFQ = RF_FieldMap( ...
    Ex, Ey, Ez, ... % [V/m]
    Bx, By, Bz, ... % [T]
    xa(1), ... % x0,y0 [m]
    ya(1), ...
    hx, ... % mesh size [m]
    hy, ...
    hz, ...
    -1, ... % length [m]
    frequency, ... % [Hz]
    direction, ... % +1, -1, 0
    P_map, ... % design input power [W]
    P_actual); % actual " "

L = Lattice();
L.append(RFQ);
```

Example of Octave script implementing two coils

```
%% Load RF-Track
RF_Track;

%% Declare two coils
Cm = Coil(0.01, -1.0, 0.2);    % L length [m],
                               % B field at the center of the coil [T],
                               % R radius [m]
Cp = Coil(0.01, +1.0, 0.2);

%% Create a Volume
V = Volume();

% Add the two coils
V.add(Cm, 0, 0, -0.5);
V.add(Cp, 0, 0, 0.5);

% Set the boundaries
V.set_s0(-1.0); % -1 m
V.set_s1(+1.0); % +1 m
```

Special elements: 3-D static magnetic field maps

- When working with a field map care must be taken to interpolate the field in such a way as to ensure that $\nabla \cdot B = 0$

- In the paper:

  > Brackbill, J.U. and Barnes, D.C.,"*The effect of nonzero $\nabla \cdot B$ on the numerical solution of the magneto-hydrodynamic equations*", Journal of Computational Physics, Volume: 35 Issue: 3 Pages: 426-430, 1980

  the authors demonstrate that $\nabla \cdot \mathbf{B} \neq 0$ results in numerical errors in the solution of the equation of motion that violate Liouville's theorem (phase space volume is not preserved).

$\nabla \cdot \mathbf{B} = 0$ interpolation

- One can ensure that an interpolation of the field satisfies $\nabla \cdot \mathbf{B} = 0$
  - by working with the magnetic vector potential $\mathbf{A}$
  - by ensuring that the interpolation is C^2 continuous (the field and its first two derivatives are continuous in the entire domain)

- Then one must solve for the vector potential $\mathbf{A}$:

$$\nabla \times \mathbf{A} = \mathbf{B}$$

  which is a set of coupled partial differential equations, which can be solved in multiple ways.

- Reference:

  Mackay, F., R. Marchand, and K. Kabin, "*Divergence-free magnetic field interpolation and charged particle trajectory integration*", J. Geophys. Res., 111, A06205, 2006

Special elements: 3-D static magnetic field maps

In RF-Track, the element

```
Static_Magnetic_FieldMap ()
```

implements the field using such a reconstruction of the vector potential.

Given the field map of a static magnetic field, the vector potential is calculated

It offers two alternative methods

- Using a fast implementation of the Helmholtz decomposition based on FFT (very fast)
- Using a slower implementation based on the integration of the surface fields

Tracking is then performed in the reconstructed magnetic field

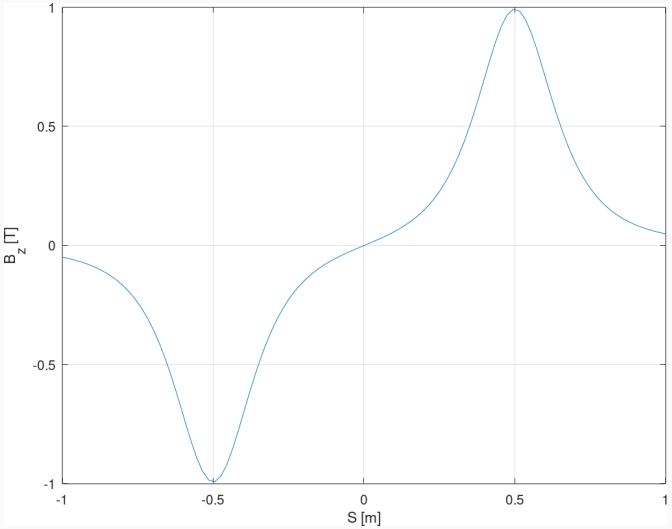
Special elements: Analytic Coil

Analytic coil equations. The coil is specified given radius and current (or alternatively radius and max field).

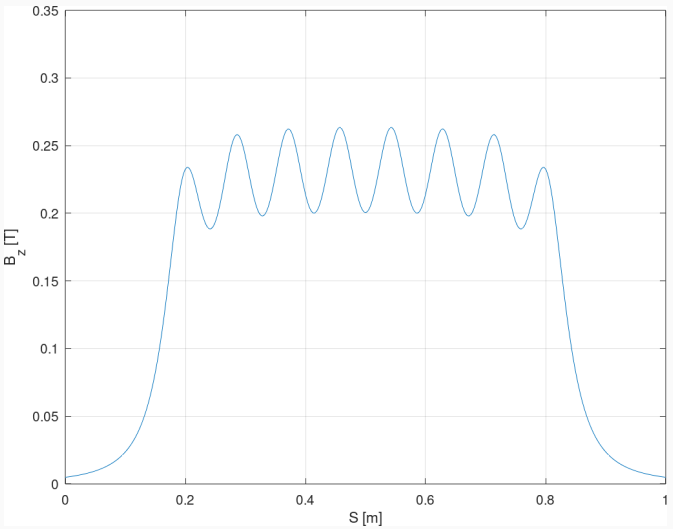
Its field extend to the whole Volume (with fringe fields)

Two examples:

2 coils:



8 coils:



Example 1: (left) On-axis field of two coils, located respectively at $S = -0.5$ m and $S = 0.5$ m, carrying an equal electric current which flows in opposite directions. The volume extends from -1 to 1 m.

Example 2: (right) On-axis field of a sequence of 8 coils to form a solenoid-like magnetic field.

Special elements: Absorber

Implements Multiple Coulomb Scattering:

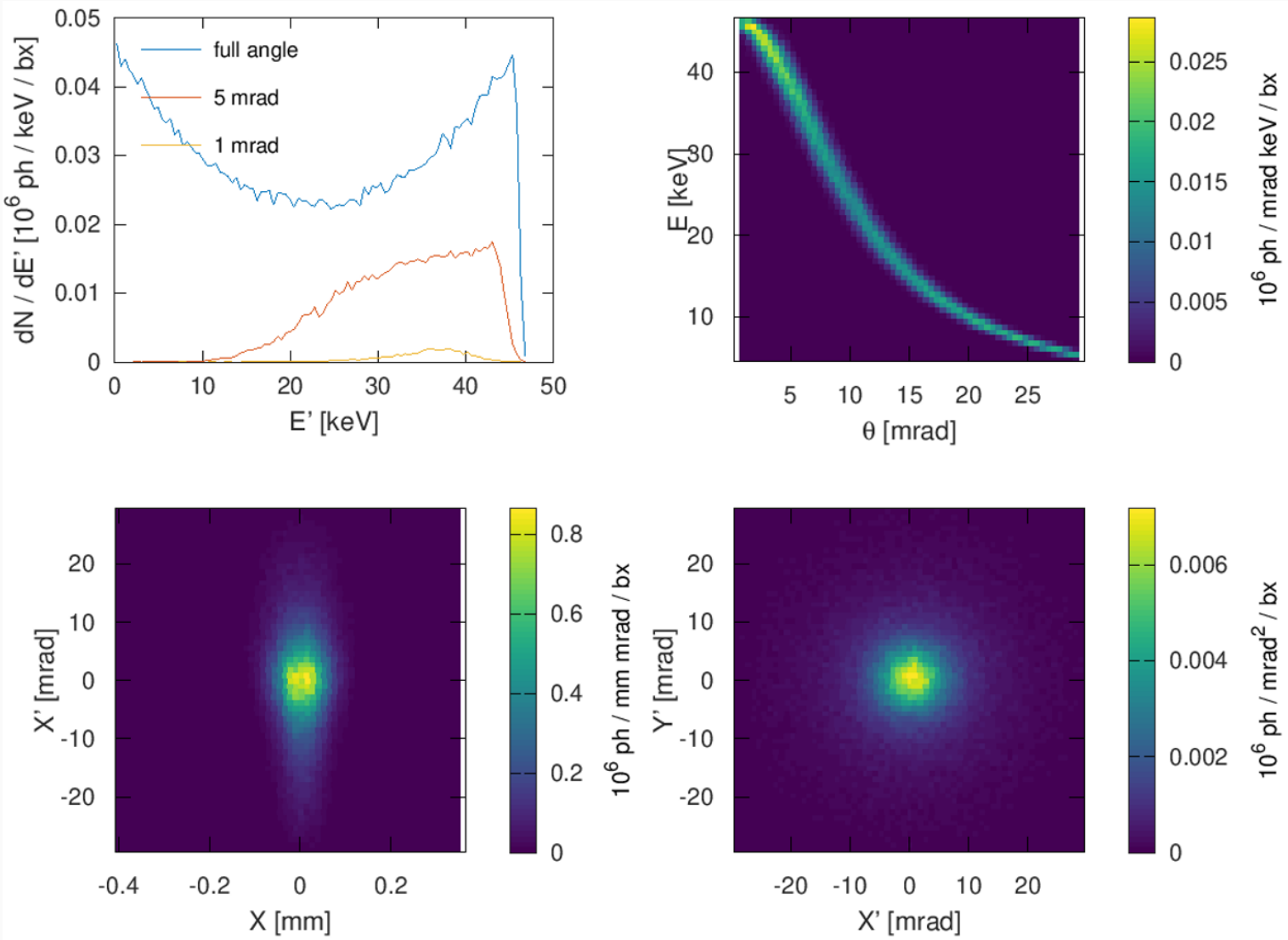
```
Absorber(length, radiation_length, Z, A, density, mean_excitation_energy);
Absorber(length, material_name );
```

Pre-defined materials include: 'air', 'water', 'beryllium', 'lithium', 'liquid_hydrogen'.

Live demo.

Special elements: LaserBeam

To simulate Inverse-Compton Scattering: (ThomX example)



Collective and Single-Particle Effects

Collective and single-particle effects

"Effects" can be attached to any element. An arbitrary number of effects can be attached.
Collective effects:

- Space-charge
  - Mirror charges at cathode
- Short-range wakefields:
  - Karl Bane's approximation
  - Arbitrary Spline
- Long-rage wakefields
  - Frequency, amplitude, and Q factor
- Self-consistent Beam-loading effect in TW structures (SW in progress)

Single-particle Effects:

- Incoherent Synchrotron Radiation (in any fields)
- Multipole kicks for magnetic-imperfection studies
- Multiple Coulomb Scattering (NEW!)

3-D Space-charge: P2P and PIC

RF-Track provides self-consistent 3-D solvers, with two optional methods:

1. Particle-2-particle:
   - computes the electromagnetic interaction between each pair of particles
   - numerically-stable summation of the forces (Kahan summation)
   - fully parallel: $O\left(N^2_{\text{particles}}\right)/N_{\text{cores}}$ operation

2. 3-D Particle-in-cell code: $\rightarrow$ fast
   - uses 3-D Integrated Green functions
   - computes E and B fields directly from ϕ and $\vec{A}$ (this ensures $\nabla \cdot \vec{B} = 0$)
   - can save E and B field maps on file, and use them for fast tracking
   - implements continuous beams
   - fully parallel: $O\left(N_{\text{mesh points}} \log N_{\text{mesh points}}\right)/N_{\text{cores}}$ operations

- No approximations such as "small velocities", or $\vec{B} \ll \vec{E}$, or rigid gaussian bunch, are made.
- It can simulate beam-beam forces

Example: RFQ Commissioning at LIGHT (ADAM)

"LIGHT is a normal conducting 230 MeV medical proton linear accelerator being constructed by ADAM.

For the commissioning, RFQ beam dynamics simulations were performed with RF-TRACK by simulating the particles through the 3-D field map."

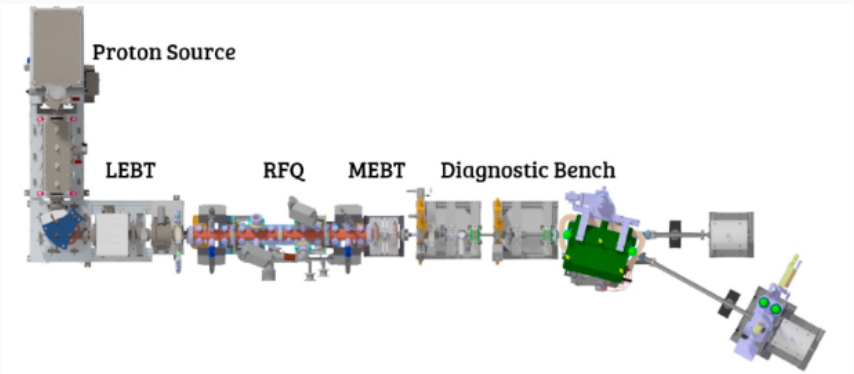


Figure 1: Layout of the LIGHT structures during the beam commissioning at 5 MeV.

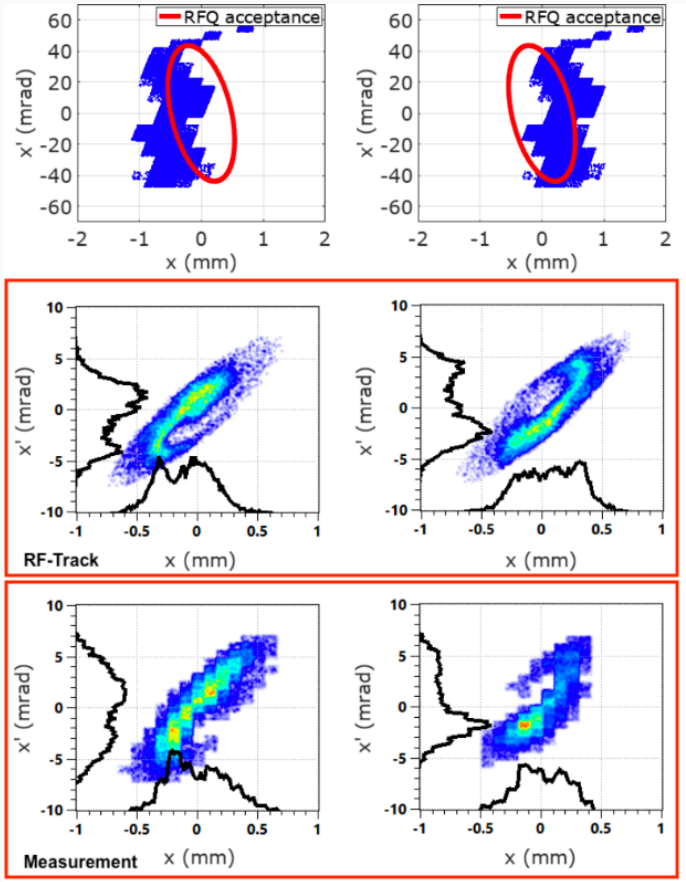


Figure 7: Horizontal phase space plots of the RFQ input beam when steered in the negative and positive x directions (first row), expected (second row) and the measured (third row) phase space plots after the RFQ for each case.

V. Dimov et al., "Beam commissioning of the 750 MHz proton RFQ for the LIGHT prototype", IPAC2018, Vancouver, BC, Canada, TUPAF002

How to get it

How to get it

RF-Track:

- A new code: minimalistic, parallel, fast
- Friendly and flexible, it uses Octave and Python as user interfaces
- It can track any particles at any energy, even mixed together
- It implements direct space-charge and beam-beam effects
- It implements matrix-based symplectic tracking, useful for small storage rings
- It implements special elements, useful for complex simulation scenarios, including muons

In-progress documentation:

- `https://zenodo.org/record/4580369`

Pre-compiled binaries and more up-to-date documentation are available here:

- `https://gitlab.cern.ch/rf-track`

Acknowledgements:

Ccontributions from Avni Aksoy, Javier Olivares Herrador, Mohsen Dayyani Kelisani.