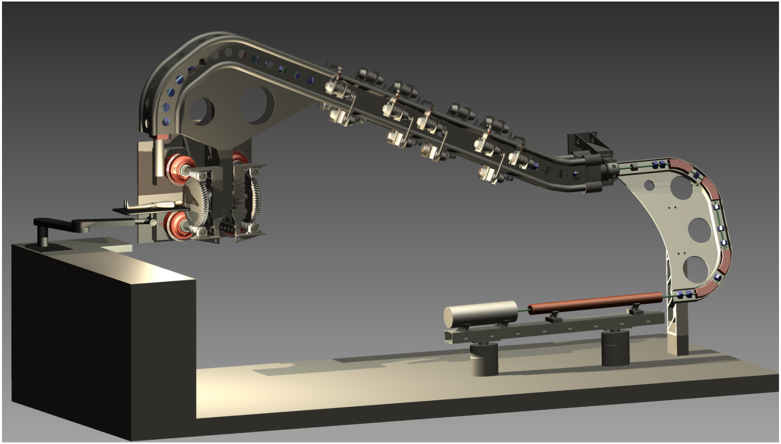
RF-Track: a minimalistic multipurpose tracking code featuring space-charge

Andrea Latina
(BE-ABP-LAT)

ABP Information Meeting - Mar 31, 2016

Motivations for RF-Track: The TULIP Project

Optimisation of a compact high-gradient linac for accelerating protons and carbon ions



$E_{\mathrm{proton}} = 70 - 230$ MeV ; uses 3 GHz backward travelling-wave RF structures

RF-Track Highlights

- Can handle complex 3d field maps of oscillating electromagnetic fields:
  - supports static, as well as fwd / bwd travelling-wave RF fields
- It's fully relativistic
  - no approximations like $\beta_{\text{rel}} \ll 1$ or $\gamma_{\text{rel}} \gg 1$
  - can handle (and has successfully been tested with) : electrons, positrons, protons, antiprotons, ions, at various energies
- Can track mixed-species beams
- Implements high-order integration algorithms
- Implements space-charge
- It is flexible, programmable, and fast

RF-Track Internals

- RF-Track is a C++ library:
  - fast, optimised code
  - modern C++11, natively parallel
  - great care for numerical stability

- Physics-oriented: it's a minimalistic code, relies on two robust and well known open-source libraries for *"all the rest"*
  - GSL, "Gnu Scientific Library", provides a wide range of mathematical routines such as random number generators, ODE integrators, linear algebra packages, . . .
  - FFTW, "Fastest Fourier Transform in the West", the fastest library to compute discrete Fourier transforms freely available

Overview: User interface

RF-Track is a library, loadable from

- Octave, *a high-level language for numerical computations, mostly compatible with Matlab, open-source*
- Python, *general-purpose, high-level programming language, open-source*

Both languages offer powerful toolboxes for numerical experimentation: multidimensional optimisations, fitting routines, data analysis, control tools, ...

Example (user script using octave)

```
% load RF-Track
RF_Track;

% setup simulation
TL = setup_transferline;
B0 = setup_beam;

% track
B1 = TL.track(B0);

% inquire the phase space
M1 = B1.get_phase_space("%x %xp %y %yp");

% plot
plot(M1(:,1), M1(:,2), "*");
xlabel("x [mm]");
ylabel("x' [mrad]");
```

Overview: Input/Output

- I/O mostly through Octave / Python: ASCII files, binary files, HDF5
- RF-Track can also save beam data in DST format (PlotWin)
- Handles automatic compression / decompression of files (useful e.g. w/ field maps)

E.g. one can inquire the phase space with great flexibility:

```
M1 = 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

Tracking: Two beam models

1. Beam moving in space:

   - All particles have the same S position
   - Each particle's phase space is

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

   where t is the proper time of each particle at S
   - Tracking is performed integrating in dS:

   $$S \to S + dS$$

2. Beam moving in time:

   - All particles are taken at same time t
   - Each particle's phase space is

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

   - Notice: can simulate particles with $P_z < 0$ or $P_z = 0$ (!)
   - Tracking is performed integrating in dt:

   $$t \to t + dt$$

- Additionally each particle stores

m : mass [MeV/c^2], Q : charge [e^+], N : nb of particles / macroparticle

RF-Track can simulate mixed-species beams

Tracking: Elements, example of RF field maps

Implements a minimal set of elements: Drifts, Quadrupoles, and RF fields

- ..but RF fields and drifts can embed a constant $\vec{B}$ field (e.g. solenoid)

Example: RF-Field maps

- Accepts complex field maps for $\vec{E}$ and $\vec{B}$
  - fwd / bwd traveling + static fields
  - trilinear interpolation
- Accepts half / quarter 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-Numbers are considered as walls
  - allows to precisely track losses in the 3d volume
- Allows to retrieve the fields at any point: e.g.
  `[E,B] = RFQ.get_field(x,y,z,t);`

User input:

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

RFQ = RF_Field( ...
    field.Ex, ... % Efield [V/m]
    field.Ey, ...
    field.Ez, ...
    field.Bx, ... % Bfield [T]
    field.By, ...
    field.Bz, ...
    field.xa(1), ... % x0,y0 [m]
    field.ya(1), ...
    field.hx, ... % mesh size [m]
    field.hy, ...
    field.hz, ...
    field.za(end), ... % length [m]
    field.frequency, ... % [Hz]
    field.direction, ... % +1, -1, 0
    field.P_map, ... % [W]
    field.P_actual);
```

Tracking: Integration algorithms

By default, RF-Track uses:

- "leapfrog" integration algorithm: fast, second-order accurate, symplectic

In some cases, leapfrog might not be accurate enough. RF-Track offers 9 GSL algorithms as an alternative:

- Explicit algorithms:
  - ⋆"rk2" Runge-Kutta (2, 3)
  - ⋆"rk4" 4th order (classical) Runge-Kutta
  - ⋆"rkf45" Runge-Kutta-Fehlberg (4, 5)
  - ⋆"msadams" multistep Adams in Nordsieck form;
    order varies dynamically between 1 and 12
  - ⋆"rkck" Runge-Kutta Cash-Karp (4, 5)
  - ⋆"rk8pd" Runge-Kutta Prince-Dormand (8, 9)
- Implicit algorithms, i.e. symplectic:
  - ⋆"rk1imp", "rk2imp", "rk4imp" implicit Runge-Kutta
  - ⋆"bsimp" Bulirsch-Stoer method of Bader and Deuflhard
  - ⋆"msbdf" multistep backward differentiation formula (BDF) method in Nordsieck form

Example:

```
L = Lattice();
L.append(RFQ);
L.set_odeint_algorithm("msadams");

B1 = L.track(B0, 1.0); % tracks in time, using integration step dt = 1 mm/c
```

Collective effects: Space-charge interaction

RF-Track solves the laws of magneto and electrostatics

- computes the full 3d fields $\vec{E}$ and $\vec{B}$ generated by any particle distribution,
- no approximations such as: small transverse velocities, or $\vec{B} \ll \vec{E}$, or "gaussian bunch", are made.

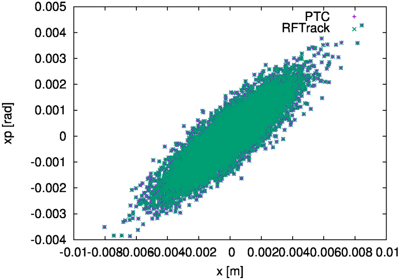
It offers two algorithms:

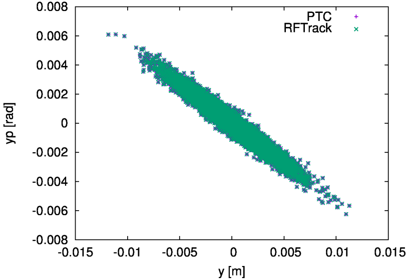
1. particle-2-particle: $O\left(n_{\text{particles}}^2 / n_{\text{cpus}}\right)$ computations
   - computes the electromagnetic interaction between each pair of particles
   - uses a numerically-stable summation of the forces (Kahan summation)
   - it's fully parallel
2. cloud-in-cell: $O\left(n_{\text{grid}} \cdot \log n_{\text{grid}} / n_{\text{cpus}}\right)$ computations $\rightarrow$ much faster
   - uses integrated Green functions over the 3d mesh
   - uses five-point derivatives for computing $\vec{E}$ and $\vec{B}$, error $O\left(h^4\right)$
   - can save E and B field maps on disk, and use them for fast tracking
   - implements continuous beams
   - it's fully parallel

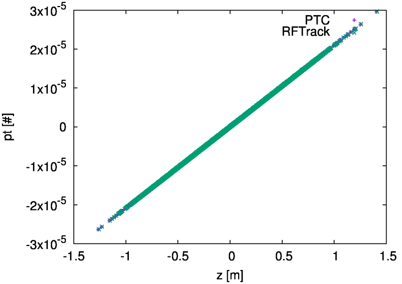
$\Rightarrow$ Can simulate beam-beam forces

Examples: ELENA Transfer Line

(Thanks to Javier R. López, University of Liverpool, not yet published)



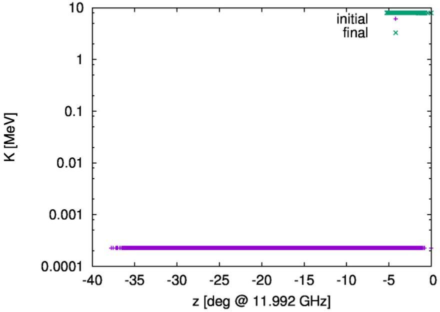




- Antiprotons with kinetic energy $E_{\text{kinetic}} = 100$ keV ($\beta_{\text{rel}} \approx 0.015$)
- Transfer line with 6 FODO cells
- Particle-to-particle comparison with PTC.

Examples: X-band Injector Gun

(Thanks to Avni Aksoy, University of Ankara, not yet published)

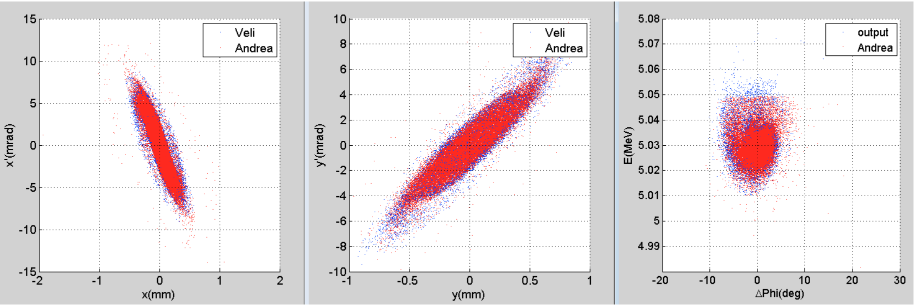


X-band photo injector gun

- Electron photo-injector accelerating e^- from $E_{\text{kinetic}} = 0.05$ eV ($\beta_{\text{rel}}{\approx}0.0004$) to ˜7.5 MeV, using a $5\pi/6$ X-band structure with 200 MV/m gradient
- high-order integration required

Examples: RFQ

(Thanks to Alessandra Lombrardi, Veliko Dimov)

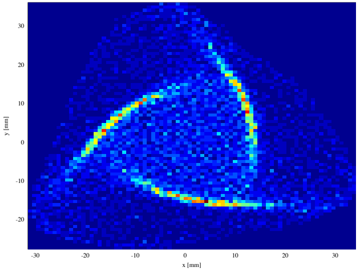


Simulated the tracking in the RFQ. Veliko kindly provided the field maps and the input distribution.

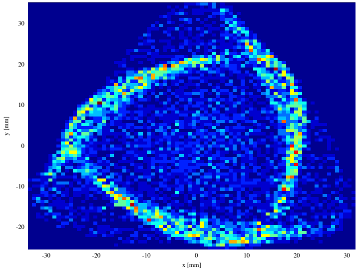
- Accelerates proton from $E_{\text{kinetic}} = 40$ keV to $E_{\text{kinetic}} = 5$ MeV
- Expected transmission confirmed = ~25%
- Field map: $27 \times 27 \times 10'000$
- Tracking 100'000 particles takes less than 1 minute on a modern PC
- 3D tracking of losses

Examples: Lead Ion Source for Linac 3

(Thanks to Alessandra Lombrardi, Marc Maintrot, Ville Toivanen)

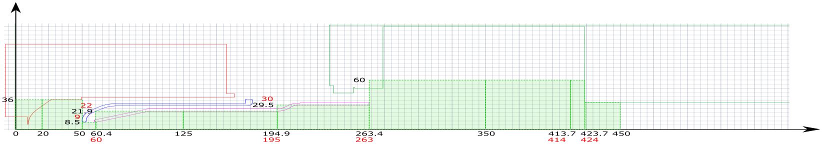


(no space charge)



(with space charge)

- IBSimu-generated input distribution, contains oxygen ions from O^{1+} to O^{8+} , and lead ions from Pb^{21+} to Pb^{36+}
- Lead ions are accelerated from $P = 146$ MeV/c to $P = 450$ MeV/c
- IBSimu-generated field maps (embed the effects of space-charge)



Examples: Beam-beam force (1/2)

(Many thanks to Elias Métral for reviewing these results and for answering my many questions)
As RF-Track solves the full set of Maxwell equations for $\vec{E}$ and $\vec{B}$. With two bunches going in opposite directions, it is possible to simulate beam-beam effects.

Benchmarks against analytical model

- Analytical beam-beam force for a gaussian beam:

$$F_r\left(r\right) = \frac{N q_1 q_2}{2\pi\varepsilon_0 l_b}\left(1+\beta_{\text{rel}}^2\right)\frac{1-\exp\left(-\frac{r^2}{2\sigma^2}\right)}{r}$$

- Simulation setup
  - bunch with 20 million particles
    - gaussian transversely
    - uniform longitudinally
  - mesh 256×256×128

    $\sigma_z \gg \sigma_x, \sigma_y$
  - transverse velocities $= 0$ (to match the analytical assumption)

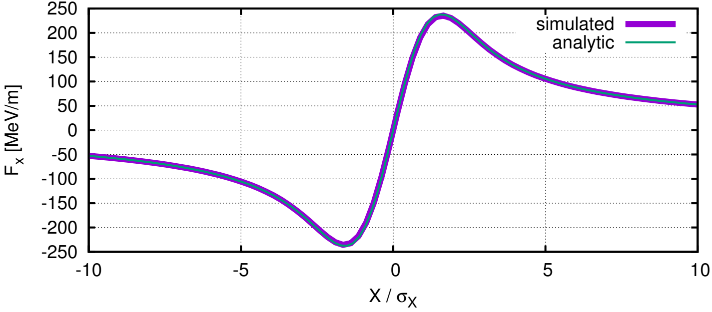
  computational time : $t_{\text{cpu}} \approx 30$ s on my laptop

Examples: Beam-beam force (2/2)

LHC-like parameters:

- head-on collision
- $P_z = 7$ TeV / c;
- $P_x = P_y = 0$
- $\sigma_z = 75.5$ mm
- $\beta^\star = 0.55$ m
- normalised emittance = 3.75 mm·mrad;

Force is calculated in the range $[-10\sigma, +10\sigma]$



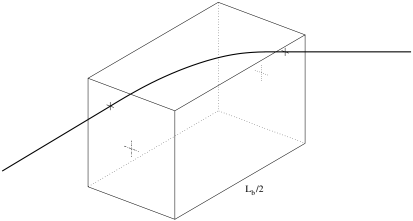
Examples: Beam-beam tune-shift

(Many thanks to Elias Métral for reviewing these results and for answering my many questions)
The tune shift is computed from full 6d tracking

- Same bunch setup as in the previous slide
- Tracks several witness particles through the field
- From the orbit deflection fits the equivalent quadrupole kick, K_x, then computes the phase shift:

$$\Delta\mu_x = -\frac{1}{2}\beta^\star K_x \;\rightarrow\; \Delta Q_x = \frac{\Delta\mu_x}{2\pi}$$

- To speed up the computations, it saves the 3d $\vec{E}$ and $\vec{B}$ fields maps and tracks through them



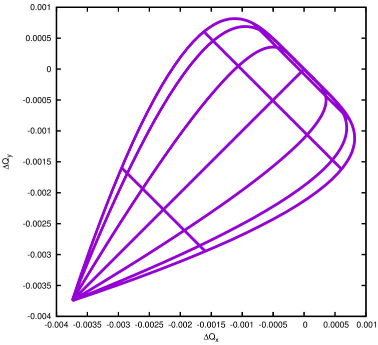
("weak-strong" interaction)

Computed tune shift in $(0,\ 0)$:

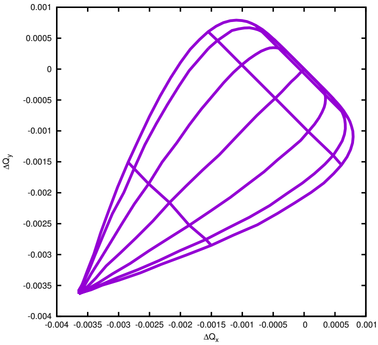
$$\Delta Q_x = -0.003712$$
$$\Delta Q_y = -0.003707$$

Analytical result: $\Delta Q_{x/y} = -\frac{\left(N{=}1.15\cdot 10^{11}\right) r_p}{4\pi\left(\gamma\epsilon_{x/y}{=}3.75\ \mu\text{m}\right)} = -0.0037452$; with $r_p \simeq 1.5347\cdot 10^{-18}$ m, the classical proton radius.

Examples: 1-turn linear beam-beam phase-shift

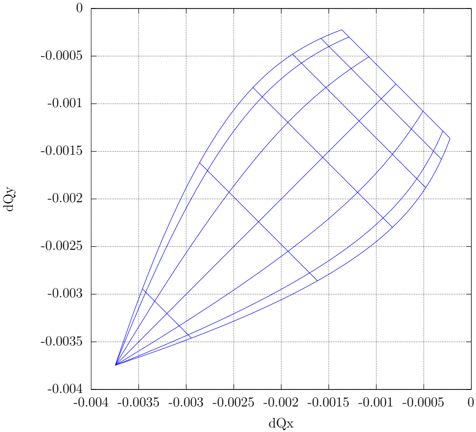


(1 turn, analytic)



(1 turn, simulated)

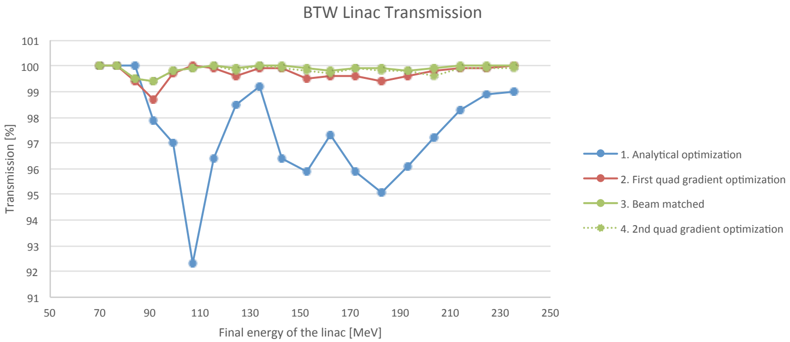
For each point $(x_{n\sigma}, y_{m\sigma})$, the average of the linear tune-shift of all particles with $r = \sqrt{x^2 + y^2} < r_{mn}$ is taken:



(average tune-shift for particles up to 6σ over a disk, simulated)

Last example: TULIP

(Courtesy of Stefano Benedetti, EPFL Doctoral Student in BE-RF-LRF, in progress)



Linac design using permanent magnets

- to accelerate protons from 50 to 240 MeV, using 18 tanks (continuously)
- multidimensional optimisation at all energies
- final energy tuning optimised modulating the input power

Summary and future steps

RF-Track:

- A new code has been developed: minimalistic, parallel, fast
  - lot of experience went into it: C++ programming, numerical algorithms, particle tracking
- Flexible: can track any particles at any energy, all together
- It implements direct space-charge (and beam-beam)
- It is being documented, but it's simple enough to be already usable
  - you are welcome to test it!

Future steps:

- Work on documentation
- Simulate electron cooling (in progress...)

Availability: a pre-compiled version with simple clarifying examples exists on lxplus