Propagation

Maximilian Reininghaus
CORSIKA 8 Workshop
2022-07-13

CORSIKA 8 does...

- particle propagation*
- event generation
- bookkeeping

*propagation: solving equations of motion, considering both deterministic and probabilistic constraints

In the beginning

- life was easy:
  - no magnetic fields
  - no energy loss
  - rectilinear motion, constant momentum
- lifetime & cross-section constant, sampling from exp.
- quadratic eq. to calculate intersections
- difficult aspect: grammage integration in curved atmosphere

energy losses added

- variation of lifetime & cross-section
- initial sample not accurate anymore
- step-length limitation introduced to limit e-loss to < X% per step
  - e-loss process's responsibility, but doesn't know about cross-section/lifetime
  - e-loss calculation happens effectively twice
- e-loss does not know about energy cut, tracks can be too long (→ longitudinal profiles)

magnetic fields added

- lateral/angular displacement depends on step-length
- intersections: step-length depends on displacement
- solution: combine equations, requires solving quartic equations
- simplifications necessary:
  - B evaluated at start of track
  - direction vector not normalized
- energy loss not considered (constant gyroradius)

magnetic fields added

- grammage calculation still with rectilinear path in initial direction
- step-length limitation to ensure deflection < 0.01 rad

New developments: `Step` class

- `particle.getPostion()`, `getMomentum()`, etc. confusing (before/after step?) for developers
- `particle.setXY()` potentially overwritten
- `Step` class keeps pre/post-step information
- adds clarity, but doesn’t really help with the fundamental problems

Sampling problem

- we sample decay time, interaction length
- select minimum: conversion to length using current particle state
  - non-const. velocity & curvature not taken into account

New proposal

- attempt to address all aspects consistently and in a combined way
- still keep flexibility & modularity
- only possible on differential level:
  change of state = sum of individual terms
- solve ODE system numerically with adaptive algorithm
  - should find trade-off between runtime/precision

New proposal

- state $s = (\boldsymbol{x}, \boldsymbol{p})$ (or equiv. representation)
- example equations of motion

$$\frac{\mathrm{d}s}{\mathrm{d}t} = \begin{pmatrix} \boldsymbol{p}/m \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ q\,(\boldsymbol{v} \times \boldsymbol{B}) \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{q}{m}\boldsymbol{E} \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{\boldsymbol{p}}{|\boldsymbol{p}|}\varrho\frac{\mathrm{d}E}{\mathrm{d}X} \end{pmatrix}$$

free propagation mag. deflection el. field ioniz. losses

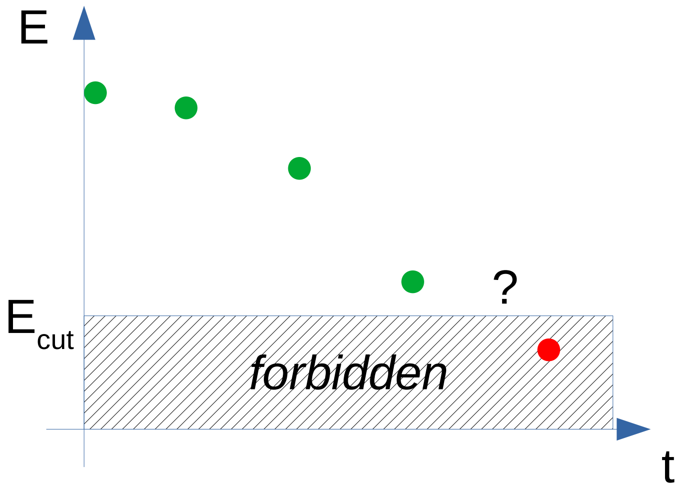
- in general $\frac{\mathrm{d}s}{\mathrm{d}t} = \sum_i \left.\frac{\mathrm{d}s}{\mathrm{d}t}\right|_i (s)$

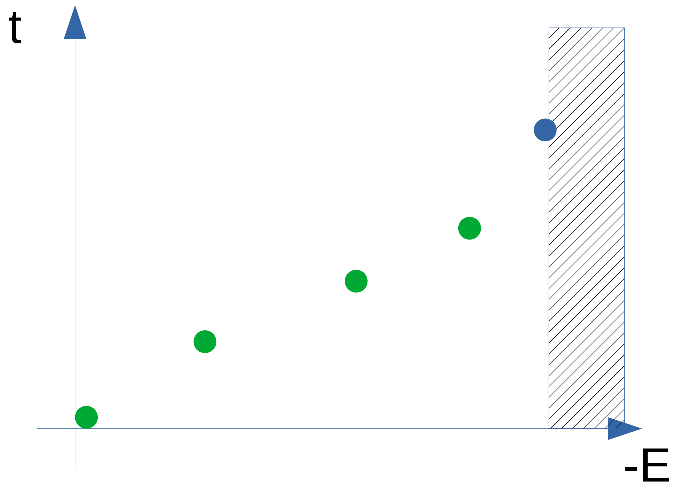
Implementation

- ~~`doContinuous(Step)`~~
- `DiffParticleState process.getDiffState(ParticleState const&)`
  - `ParticleState`: only **local** information (pos., mom., time)
  - `DiffParticleState`: change, $\frac{\mathrm{d}s}{\mathrm{d}t}$
- sum over all contributions
- feed into (adaptive) ODE solver (e.g. some Runge-Kutta integrator)
- while solving, watch out for terminating conditions (cuts, boundaries)
  - inspired by `scipy.integrate.solve_ivp` "events"
- after integration, complete trajectory is available
  - fed into "observing" processes

Independent variable

- What is the best independent variable,
  time, distance / arc length, grammage,… ?
- Ideally the one with cuts, e.g. energy
  - but we have several…





What about sampling?

Survival probability (i.e. not undergoing an interaction/decay from A → B) fulfills:

$$\frac{\mathrm{d}P_\mathrm{s}}{\mathrm{d}t} = -\alpha(s(t))P_\mathrm{s} \qquad \alpha = \frac{\sigma\rho}{\langle m\rangle} + \frac{1}{\beta\gamma c\tau_0}$$

Non-negative **hazard function**

solution: $$P_\mathrm{s}(A,B) = \exp\left(-\int_A^B \alpha(s(t))\,\mathrm{d}t\right)$$

P_s = complementary cumulative distribution function distributed uniformly → inverse sampling

$$P_\mathrm{s}(t) = u \quad \Leftrightarrow \quad t = P_\mathrm{s}^{-1}(u)$$

Sampling: alternatives

treatment like a cut:

1) sample uniform u^*

2) integrate eq. of motion (yielding *s(t)*)

3) stop as soon as

$$\exp\left(-\int_0^T \alpha(s(t))\,\mathrm{d}t\right) = u^*$$

$$\int_0^T \alpha(s(t))\,\mathrm{d}t = -\log(u^*)$$

change of independent variable:

$$\frac{dt}{\mathrm{d}u} = \left(\frac{du}{\mathrm{d}t}\right)^{-1} = -\left(\frac{dP_\mathrm{s}}{\mathrm{d}t}\right)^{-1}$$

$$= \frac{1}{\alpha P_\mathrm{s}} = \frac{1}{\alpha(1-u)}$$

$$\frac{\mathrm{d}s}{\mathrm{d}u} = \frac{\mathrm{d}s}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}u}$$

draw u^* and integrate eq. of motion from $u = 0$ to $u = u^*$

Advantages

- no more inconsistencies
- grammage calculation unnecessary (arbitrary density profiles possible!)
- electric fields straight-forward to add

Issues

- performance impact unknown
- unit system prevents usage of off-the-shelf ODE solver libraries (boost::odeint)
- treatment of multiple scattering consistent with constraints

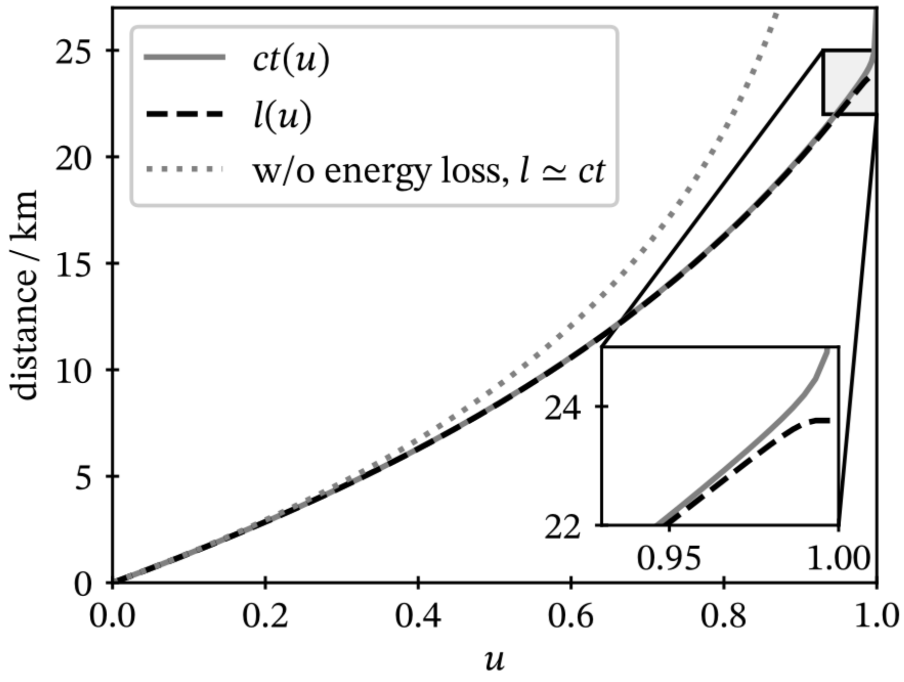
Summary

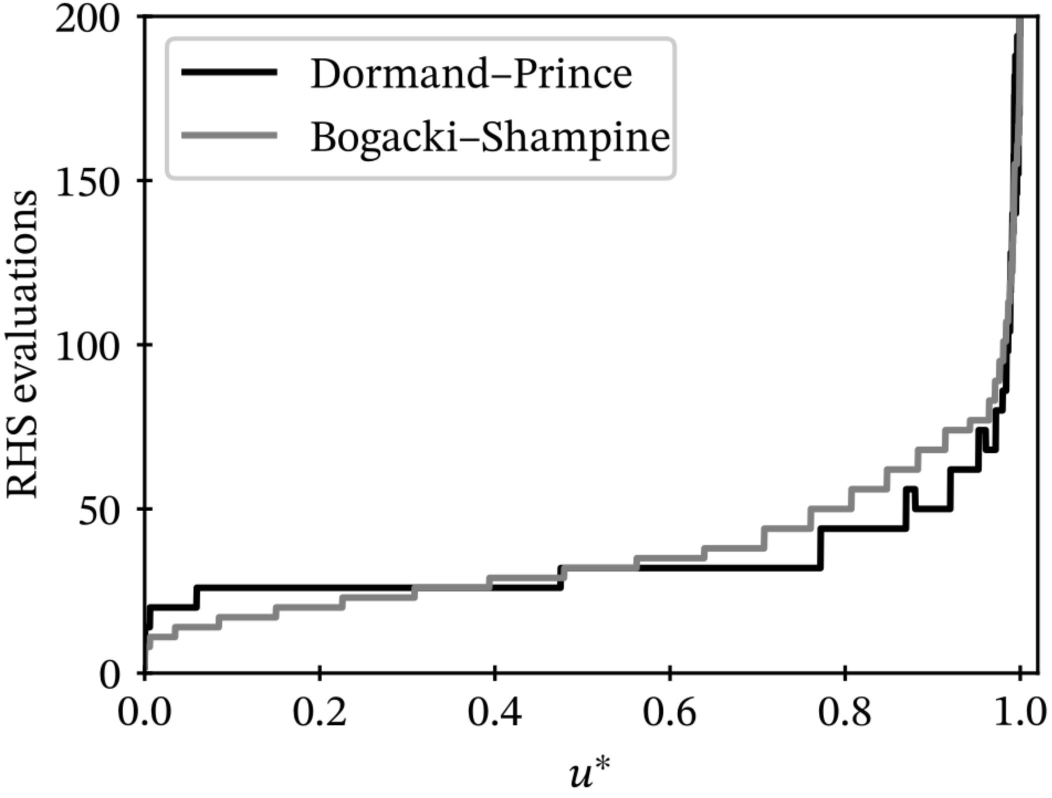
- Propagation as of now is a mess; responsibilities spread out; difficult to enhance
- restructuring necessary on basis of solid formal foundations
- ODE-based solution can solve most issues
- some work already done in MR 322

Supplementary material

Example: MIP muon, 1D

$$\frac{\mathrm{d}ct}{\mathrm{d}u} = \frac{m_\mu}{Ec\tau}\frac{1}{1-u},$$
$$\frac{\mathrm{d}l}{\mathrm{d}u} = \frac{\mathrm{d}l}{\mathrm{d}ct}\,\frac{\mathrm{d}ct}{\mathrm{d}u} = \frac{1}{E}\sqrt{E^2 - m_\mu^2}\,\frac{\mathrm{d}ct}{\mathrm{d}u}\,.$$
$$\frac{\mathrm{d}E}{\mathrm{d}u} = \frac{\mathrm{d}E}{\mathrm{d}X}\,\varrho(h(l))\,\frac{\mathrm{d}l}{\mathrm{d}u}\,.$$





Dormand–Prince
Bogacki–Shampine
RHS evaluations
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
u^*