LhARA linear optics documentation

N. Dover, K.R. Long, J. McGarrigle, M. Maxouti

Contents

1 Introduction **1**

2 Coordinate systems **1**
2.1 Laboratory coordinate system 1
2.2 Reference particle local coordinate system 2
2.3 Transforming to and from reference particle local coordinates to laboratory coordinates 3

3 Phase space, trace space, beam parameters **3**
3.1 Phase space 4
3.2 Trace space 4
3.3 Beam parameters 5

4 Transfer matrices **5**
4.1 Drift 6
4.2 Quadrupole 6
4.3 Solenoid 7
4.4 Non-neutral (electron) plasma (Gabor) lens 8
4.5 Dipole 8

5 Source **9**
5.1 Energy distribution 10
5.2 Angular Distribution 12
5.3 Spatial distribution 12
5.4 Simulated distributions 12

6 Beam-line specification and scripts **15**
6.1 Beam-line specification 15
6.2 Scripts 16

A Module, class and data structures **19**
A.1 `BeamLine` 20
A.2 `Particle` and `ReferenceParticle` 25
A.3 `Beam` and `extrapolateBeam` 33
A.4 `BeamLineElement` 39
A.5 `UserFramework` 60
A.6 `visualise` 60
A.7 `BeamIO` 62
A.8 `Simulation` 64
A.9 `Physical constants` 67
A.10 `Report` 67
A.11 `LaTeX` 69

1 Introduction

The LhARA [1, 2] linear optics package was written to allow rapid calculations to initiate more detailed studies of the LhARA beam lines and for use as a tool to check issues as they arise. The package has been written in Python so that it is accessible and can readily be updated, modified and maintained. At present the code treats proton beams only.

This document presents the approximations and notation used and summarises the module, class and data structures that have been adopted.

2 Coordinate systems

2.1 Laboratory coordinate system

The origin of the LhARA coordinate system, the "laboratory coordinate system" or "laboratory reference frame", is at the position of the laser focus at the laser-target interaction point [3]. The z axis is horizontal and parallel to the nominal capture axis, pointing in the downstream direction. The y axis points vertically upwards and the x axis completes a right-handed orthogonal coordinate system.

Unit vectors along the x, y and z axes are $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ respectively. The position of the reference particle as well as its momentum and energy are described as functions of the distance it has travelled from the origin of coordinates. The distance the reference particle has travelled is s, making the position, $\boldsymbol{r}_0$, momentum, $\boldsymbol{p}_0$, and energy, E_0, of the reference particle position, s:

$$\begin{aligned} \boldsymbol{r}_0 &= \boldsymbol{r}_0(s)\,; \\ \boldsymbol{p}_0 &= \boldsymbol{p}_0(s)\,; \text{and} \\ E_0 &= E_0(s)\,. \end{aligned} \tag{1}$$

The magnitude of the reference particle velocity is v_0 and the relativistic parameters that determine the reference particle energy and momentum are:

$$\begin{aligned} \beta_0 &= \frac{v_0}{c}\,; \text{and} \\ \gamma_0 &= \frac{1}{\sqrt{1-\beta_0^2}}\,; \end{aligned}$$

where c is the speed of light. The time, t, at which the reference particle is at s is also a function of s:

$$t = t(s) = \frac{s}{v_0} = \frac{s}{c}\frac{E_0}{cp_0}\,; \tag{2}$$

where $p_0 = |\boldsymbol{p}_0|$.

2.2 Reference particle local coordinate system

A coordinate system defined relative to the position of the reference particle, the "reference particle local coordinate" (RPLC) system, may be defined using the direction in which the particle is travelling. The position of the particle defines the origin of the RPLC system, see figure 1. The tangent to the reference particle trajectory at s defines the z_r axis with unit vector $\boldsymbol{k}_r$. In the laboratory frame, the presence of local electric or magnetic fields may cause the reference particle's trajectory to change. In the neighbourhood of the particle,

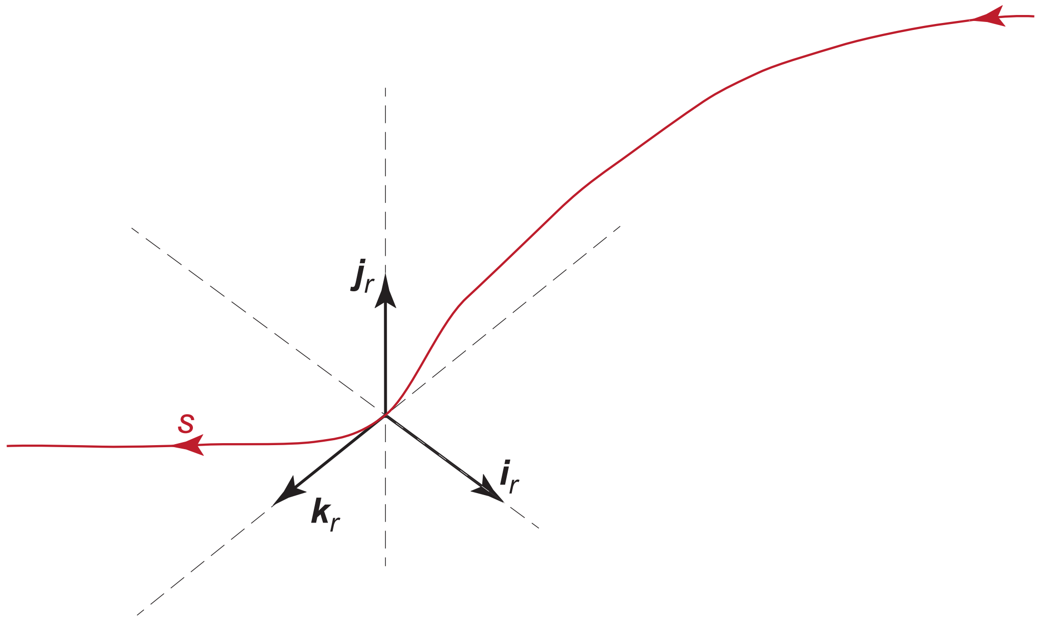

Figure 1: Reference particle local coordinate system. The trajectory of the reference particle is shown as the red line. The distance the reference particle has travelled, measured from the origin of coordinates in the laboratory frame, is labelled s. The origin of the "reference paricle local coordiante (RPLC) system is coincident with the position of the reference particle. The directions of unit vectors along each of three righthanded, orthogonal coordinate axes are shown as black arrows labelled $\boldsymbol{i}_r$, $\boldsymbol{j}_r$, and $\boldsymbol{j}_r$.

the curved trajectory may be described in terms of an arc of a circle. The x_r axis (with unit vector $\boldsymbol{i}_r$) is then taken to be in the direction pointing away from the centre of the circle. The third coordinate axis, y_r, is defined to complete the right-handed orthogonal coordinate system; the unit vector along the y_r axis being given by $\boldsymbol{j}_r = \boldsymbol{k}_r \times \boldsymbol{i}_r$.

The trajectory of the reference particle is a straight line as it traverses a drift space and a variety of beam-line elements. Examples of such beam-line elements include solenoids and quadrupoles. The reference trajectory is also undeviated by passage through an accelerating cavity placed such that the accelerating field is parallel to the reference-particle trajectory.

The RPLC coordinate system at $s = 0$ is taken to coincide with the laboratory coordinate system. Beam-line elements are placed sequentially along the trajectory of the reference particle. If necessary a coordinate transformation is performed to ensure that the RPLC system at the entrance to a particular beam-line element is consistent with the definition given above.

2.3 Transforming to and from reference particle local coordinates to laboratory coordinates

In the RPLC system, the trajectory of the reference particle, $\boldsymbol{R}_0$, is:

$$\boldsymbol{R}_0(s) = \boldsymbol{0}\,. \tag{3}$$

The position of a test particle in the RPLC frame, $\boldsymbol{R}$, is described with reference to the position of the reference particle. In the laboratory frame, the position of the test particle is:

$$\boldsymbol{r}(s) = \boldsymbol{r}_0(s) + \boldsymbol{\delta r}(s)\,; \tag{4}$$

where:

$$\boldsymbol{\delta r}(s) = \underline{\underline{R}}(s)\boldsymbol{R}(s)\,;\text{and} \tag{5}$$

$\underline{\underline{R}}(s)$ is a rotation matrix that takes the RPLCs at s to the laboratory frame coordinates.

In the laboratory frame, the unit vectors $\boldsymbol{i}_r$, $\boldsymbol{j}_r$ and $\boldsymbol{k}_r$ are given by:

$$\begin{aligned}\boldsymbol{i}_r &= \begin{pmatrix} i_{rx} \\ i_{ry} \\ i_{rz} \end{pmatrix}\,; \\ \boldsymbol{j}_r &= \begin{pmatrix} j_{rx} \\ j_{ry} \\ j_{rz} \end{pmatrix}\,;\text{and} \\ \boldsymbol{k}_r &= \begin{pmatrix} k_{rx} \\ k_{ry} \\ k_{rz} \end{pmatrix}\,.\end{aligned} \tag{6}$$

The rotation matrix, $\underline{\underline{R}}$, may now be written:

$$\underline{\underline{R}}(s) = \begin{bmatrix} i_{rx} & j_{rx} & k_{rx} \\ i_{ry} & j_{ry} & k_{ry} \\ i_{rz} & j_{rz} & k_{rz} \end{bmatrix}\,. \tag{7}$$

3 Phase space, trace space, beam parameters

The motion of particles passing through an accelerator is most often described using classical Hamiltonian mechanics; quantum mechanics being required only in particular cases such as the description of spin polarisation in a storage ring. In classical Hamiltonian mechanics the equations of motion are solved to give the evolution of the position, momentum, and energy as functions of a single independent parameter. The independent parameter is often taken to be time.

Relativistic mechanics exploits four-vector position, $\underline{\mathcal{R}} = (\boldsymbol{r}, ct)$, and four-vector momentum, $\underline{\mathcal{P}} = (c\boldsymbol{p}, E)$. In the Hamiltonian description of particle dynamics, these four vectors become functions of the independent variable, i.e. $\underline{\mathcal{R}} = \underline{\mathcal{R}}(t)$ and $\underline{\mathcal{P}} = \underline{\mathcal{P}}(t)$. In the laboratory system, the position of the reference particle along its trajectory is directly related to the time coordinate by $t = c\beta_0 s$. This allows s to be taken as the independent variable and for the motion of particles in the beam to be derived as functions of s.

The 6D phase-space coordinates of a particle as a function of s are given by the position and momentum three vectors. The particle energy may be determined from the invariant mass and the time coordinate from the invariant interval between the origin and the position represented by s.

The "trace-space" coordinates of a particle are defined relative to the reference particle. Usually, a beam is understood to contain particles which follow trajectories that differ rather little from that of the reference particle. Trace space is defined such that the position, "momentum", and "energy" coordinates are small for particles which follow trajectories close to that of the reference particle. The utility of this approach is that trace-space coordinates may be used to perform Taylor expansions of the Hamiltonian which may readily be solved to yield a description of particle transport using functions that are linear in the trace-space coordinates.

The notation used for the 6D phase and trace spaces are defined in this section.

3.1 Phase space

The 6D phase-space vector is defined in terms of the three-vector position and three vector momentum as:

$$\begin{bmatrix} \boldsymbol{r} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \end{bmatrix} \tag{8}$$

The trajectory of the particle may be evaluated as a function of time or s.

3.2 Trace space

Trace space is defined to simplify the calculation of the trajectory of particles through the accelerator lattice and is derived from the phase space expressed in the RPLC frame. Consider a particle with position $\boldsymbol{r}_{\rm RPLC} = (x_{\rm RPLC}, y_{\rm RPLC}, z_{\rm RPLC})$ and momentum $\boldsymbol{p}_{\rm RPLC} = (p_{\rm x\,RPLC}, p_{\rm y\,RPLC}, p_{\rm z\,RPLC})$. Taking the magnitude of the momentum of the reference particle in the laboratory frame to be p_0, the trace-space coordinates are given by:

$$\underline{\phi} = \begin{pmatrix} x_{\rm RPLC} \\ x'_{\rm RPLC} \\ y_{\rm RPLC} \\ y'_{\rm RPLC} \\ z_{\rm RPLC} \\ \delta_{\rm RPLC} \end{pmatrix} ; \tag{9}$$

where:

$$x'_{\rm RPLC} = \frac{\partial x}{\partial s} = \frac{c p_{x\,\rm RPLC}}{c p_0} ; \tag{10}$$

$$y'_{\rm RPLC} = \frac{\partial y}{\partial s} = \frac{c p_{y\,\rm RPLC}}{c p_0} ; \tag{11}$$

$$z_{\rm RPLC} = \frac{s}{\beta_0} - ct = \frac{\Delta s}{\beta_0} ; \text{and} \tag{12}$$

$$\delta_{\rm RPLC} = \frac{E}{c p_0} - \frac{1}{\beta_0} = \frac{\Delta E}{c p_0} . \tag{13}$$

Here $\Delta s = s - s_0$ and $\Delta E = E - E_0$, where s_0 and E_0 are the reference particle position and energy respectively; E and s are the energy and position of a particular particle in the beam.

3.3 Beam parameters

The trace space vector, $\underline{\phi}_i$, of the $i^{\rm th}$ particle at a position along the beam line contains the varaince of the particle coordinates with respect to those of the reference particle. For a sample containing N particles, the covariance matrix, $\underline{\underline{C}}_6$, may therefore be obtained by evaluating:

$$\underline{\underline{C}}_6 = \frac{1}{N} \sum_i^N \underline{\phi}_i \, \underline{\phi}_i^T = \left\langle \underline{\phi} \, \underline{\phi}^T \right\rangle ; \tag{14}$$

where the notation $\langle\rangle$ is used to denote evaluating the expectation value. The RMS emittance of the beam in all 6 trace-space dimensions is then given by:

$$\varepsilon_6 = \sqrt[6]{\left|\underline{\underline{C}}_6\right|} . \tag{15}$$

In an analagous notation, the four-dimensional transverse trace space (x, x', y, y') may be used to define the four-dimensional covariance matrix $\underline{\underline{C}}_4$, which, in turn, can be used to evaluate the four-dimensional transverse emittance:

$$\varepsilon_4 = \sqrt[4]{\left|\underline{\underline{C}}_4\right|} \, . \tag{16}$$

The size of the beam in the two "transverse planes" (x, x') and (y, y') is contained in two, 2×2 submatrices of $\underline{\underline{C}}_6$, $\underline{\underline{C}}_x$ and $\underline{\underline{C}}_y$. If the x and y coordinates are to be taken as uncoupled, then the emittance of the (x, x') and (y, y') trace space my be obtained using:

$$\varepsilon_x = \sqrt{\left|\underline{\underline{C}}_x\right|} \text{ and} \tag{17}$$

$$\varepsilon_y = \sqrt{\left|\underline{\underline{C}}_y\right|} \, . \tag{18}$$

With $u = x$ or y, the area of the trace space ellipse is given by $\pi\varepsilon_u$ and the Twise parameters, α_u, β_u, and γ_u are given by:

$$\sigma_u^2 = \left\langle u^2 \right\rangle = \beta_u, \varepsilon_u \, ; \tag{19}$$

$$\left\langle u'^2 \right\rangle = \gamma_u, \varepsilon_u : \text{and} \tag{20}$$

$$\left\langle uu' \right\rangle = -\alpha_u \varepsilon_u \, ; \tag{21}$$

where:

$$\beta_u \gamma_u - \alpha_u^2 = 1 \, . \tag{22}$$

4 Transfer matrices

A beam line may be described as a series of beam-line elements arranged one after the other. A particle may then be transported through the beam line by transporting it through each element in turn. Taking advantage of the trace-space defined above, the transport of a particle across a particular beam-line element may be performed using a linear transformation:

$$\underline{\phi}_{\text{end}} = \underline{\underline{T}} \, \underline{\phi}_{\text{start}} \, ; \tag{23}$$

where $\underline{\phi}_{\text{start}}$ is the trace-space vector at the start of the beam-line element and $\underline{\phi}_{\text{end}}$ is the transformed trace-space vector at the end. The step across the beam-line element implies an increment, δs, to the s-coordinate given by:

$$s_{\text{end}} = s_{\text{start}} + \delta s \, ; \tag{24}$$

where s_{start} and s_{end} are the coordinate along the reference particle trajectory at the start and end of the beam-line element respectively. Equation 14 implies that the covariance matrix may be transported across a beam-line element using the expression:

$$\underline{\underline{C}}_{6\,\text{end}} = \underline{\underline{T}} \, \underline{\underline{C}}_{6\,\text{start}} \, \underline{\underline{T}}^T \, . \tag{25}$$

There are many excellent descriptions of the derivation of the transfer matrices, $\underline{\underline{T}}$, so only the results are quoted here. The notation used below is developed from that used in [4].

4.1 Drift

A "drift" space refers to a region in which the beam propagates in the absence of any electromagnetic fields. In a drift, particles propagate in straight lines, therefore:

$$\underline{\underline{T}}_{\rm drift} = \begin{pmatrix} 1 & l & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & l & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l}{\beta_0^2\gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} ; \tag{26}$$

where l is the length of the drift. The increment in the reference particle position is:

$$\delta s = l \, . \tag{27}$$

4.2 Quadrupole

The passage of a beam particle through a quadrupole magnet may be described by specifying the field gradient, g, within the magnet and the length, l_q, of the quadrupole measured along its axis. The impact of a quadrupole on the trajectory of a particle in the xy plane is independent of the impact of the magnet on the particle's trajectory in the yz plane. In this sense quadrupole focusing in the xz and yz planes is said to be "uncoupled".

If the field gradient along the x and y axes is identical, then:

$$g_x = \frac{\partial B_{qx}}{\partial x} = g_y = \frac{\partial B_{qy}}{\partial y} = g \, ; \tag{28}$$

where the field in the quadrupole, $\boldsymbol{B_q}$, has components $(B_{qx}, B_{qy}, 0)$.

In the "hard-edge" approximation, where the field falls to zero at the start and end of the quadrupole, the transfer matrix for a quadrupole focusing in the xz plane (a "focusing quadrupole") may be written:

$$\underline{\underline{T}}_{\rm Fquad} = \begin{pmatrix} \cos(\sqrt{k_q} l_q) & \frac{\sin(\sqrt{k_q} l_q)}{\sqrt{k_q}} & 0 & 0 & 0 & 0 \\ -\sqrt{k_q}\sin(\sqrt{k_q} l_q) & \cos(\sqrt{k_q} l_q) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cosh(\sqrt{k_q} l_q) & \frac{\sinh(\sqrt{k_q} l_q)}{\sqrt{k_q}} & 0 & 0 \\ 0 & 0 & \sqrt{k_q}\sinh(\sqrt{k_q} l_q) & \cosh(\sqrt{k_q} l_q) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l_q}{\beta_0^2\gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} ; \tag{29}$$

where:

$$k_q = \frac{gc}{p} \times 10^{-3}\,{\rm m}^{-2} \, , \tag{30}$$

c is the speed of light in metres per second, p is the magnitude of the momentum of the particle in MeV/c, and the field gradient, g, is given in T/m. As before, β_0 is the relativistic velocity of the reference particle and $\gamma_0 = (1-\beta_0^2)^{-\frac{1}{2}}$. The increment in the reference particle position is:

$$\delta s = l_q \, . \tag{31}$$

It is important to include a description of the effect of dispersion on beam transport through the LhARA beam line since the laser-driven proton and ion source provides a broad energy spectrum. Reference [4] describes two methods for the description of dispersion in a linear approximation. The first is to use the reference momentum to calculate the quadrupole focusing strength ($k_{0q} = \frac{gc}{p_0} \times 10^{-3}\,\mathrm{m}^{-2}$) and to include terms in the expressions for x, x', y, and y' dependent on δ. The second is to use equation 30 to calculate the effective quadrupole focusing strength, with k_q evaluated using p. The second approach has been adopted here.

In the same notation, the transfer matrix for a quadrupole focusing in the yz plane (a "defocusing quadrupole") may be written:

$$\underline{\underline{T}}_{\mathrm{Dquad}} = \begin{pmatrix} \cosh(\sqrt{k_q} l_q) & \frac{\sinh(\sqrt{k_q} l_q)}{\sqrt{k_q}} & 0 & 0 & 0 & 0 \\ \sqrt{k_q}\sinh(\sqrt{k_q} l_q) & \cosh(\sqrt{k_q} l_q) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(\sqrt{k_q} l_q) & \frac{\sin(\sqrt{k_q} l_q)}{\sqrt{k_q}} & 0 & 0 \\ 0 & 0 & -\sqrt{k_q}\sin(\sqrt{k_q} l_q) & \cos(\sqrt{k_q} l_q) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l_q}{\beta_0^2\gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} . \tag{32}$$

4.3 Solenoid

The trajectory of a beam particle through a solenoid is determined by the magnetic field strength, $\boldsymbol{B}_s$, within the solenoid and the length of the solenoid, l_s, measured along its axis. As the particle enters the solenoid, the fringe field imparts momentum transverse to the axis of the magnet. This results in the particle executing a helical trajectory, the axis of the helix being parallel to the solenoid axis. The sense of the rotation depends on the particle charge and the polarity of the field. The helical motion means that the evolution of the particle motion in the xz plane is coupled with the evolution of the particle motion in the yz plane.

In the "hard-edge" approximation, the magnetic field inside the magnet is given by $\boldsymbol{B}_s = (0, 0, B_{s0})$, where the solenoid axis lies along the z_{RPLC} axis. The solenoid field-strength parameter is then given by:

$$k_s = \left[\frac{B_{s0} c}{2p} \times 10^{-3}\right]^2 \mathrm{m}^{-2} ; \tag{33}$$

where B_{s0} is measured in T, p in MeV/c and c in m/s.

The transfer matrix for passage of a positive particle through a solenoid with field pointing in the positive z_{RPLC} direction may be written:

$$\underline{\underline{T}}_{\mathrm{Sol}} = \begin{pmatrix} \cos^2(\sqrt{k_s} l_s) & \frac{1}{2\sqrt{k_s}}\sin(\sqrt{k_s} l_s) & \frac{1}{2}\sin(2\sqrt{k_s} l_s) & \frac{1}{\sqrt{k_s}}\sin^2(\sqrt{k_s} l_s) & 0 & 0 \\ -\frac{\sqrt{k_s}}{2}\sin(2\sqrt{k_s} l_s) & \cos^2(\sqrt{k_s} l_s) & -\sqrt{k_s}\sin^2(\sqrt{k_s} l_s) & \frac{1}{2}\sin(2\sqrt{k_s} l_s) & 0 & 0 \\ -\frac{1}{2}\sin(2\sqrt{k_s} l_s) & -\frac{1}{\sqrt{k_s}}\sin^2(\sqrt{k_s} l_s) & \cos^2(\sqrt{k_s} l_s) & \frac{1}{2\sqrt{k_s}}\sin(2\sqrt{k_s} l_s) & 0 & 0 \\ \sqrt{k_s}\sin^2(\sqrt{k_s} l_s) & -\frac{1}{2}\sin(2\sqrt{k_s} l_s) & -\frac{\sqrt{k_s}}{2}\sin(2\sqrt{k_s} l_s) & \cos^2(\sqrt{k_s} l_s) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l}{\beta_0^2\gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} . \tag{34}$$

As in the case of the quadrupoles, dispersion is accounted for by using p to calculate k_s (equation 33). The increment in the reference particle position is:

$$\delta s = l_s . \tag{35}$$

4.4 Non-neutral (electron) plasma (Gabor) lens

A dense gas of electrons confined in a Penning-Malmberg trap provides an electric field that can be used to focus a positive ion beam. The electron gas is confined axially in the lens by an electrostatic potential created using a central anode of length l_G. The gas is confined radially using the uniform field of a solenoid. Assuming a uniform electron density, n_e, the focusing parameter, k_G, may be written:

$$k_G = \frac{e}{2\epsilon_0} \frac{m_p \gamma}{p^2} n_e \quad \mathrm{m}^{-2} \, ; \tag{36}$$

where e is the charge on the electron, ϵ_0 is the permittivity of free space, and m_p is the proton mass. As in the case of the quadrupoles and solenoid, dispersion is accounted for by using p in equation 36. The force on a particle passing through the electron gas is towards the axis of the lens and is proportional to the radial distance of the particle from the axis. Focusing is therefore cylindrically symmetric and does not couple motion in the the xz and yz planes.

In the "hard-edge" approximation, the electric field inside the lens falls to zero at the end of the electron gas and the contribution of the magnetic field used to confine the electron gas in the transverse direction has a negligible effect on particles passing through the lens. The transfer matrix for the passage of a positive particle through the lens may be written:

$$\underline{\underline{T}}_G = \begin{pmatrix} \cos(\sqrt{k_G} l_G) & \frac{\sin(\sqrt{k_G} l_G)}{\sqrt{k_G}} & 0 & 0 & 0 & 0 \\ -\sqrt{k_G}\sin(\sqrt{k_G} l_G) & \cos(\sqrt{k_G} l_G) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(\sqrt{k_G} l_G) & \frac{\sin(\sqrt{k_G} l_G)}{\sqrt{k_G}} & 0 & 0 \\ 0 & 0 & -\sqrt{k_G}\sin(\sqrt{k_G} l_G) & \cos(\sqrt{k_G} l_G) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l}{\beta_0^2 \gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} . \tag{37}$$

The increment in the reference particle position is:

$$\delta s = l_G \, . \tag{38}$$

4.5 Dipole

The reference particle trajectory in the beam-line elements described above passes along the axis of the element. In contrast, a dipole bends the reference trajectory so that it describes the arc of a circle (see figure 2). The code provides for transport through a "sector dipole" in the hard-edge approximation. In this case, the field within the magnet is taken to be constant and parallel to $\boldsymbol{j}_{\mathrm{RPLC}}$, i.e. $\boldsymbol{B}_D = (0, B_{D0}, 0)$. No edge focusing is considered.

The passage of particles through a dipole may be described by defining the parameter, k_D:

$$k_D = \left[\frac{B_{D0} c}{p} \times 10^{-3} \right]^2 \mathrm{m}^{-2} \, . \tag{39}$$

The momentum of the reference particle is related to the curvature. ρ, by:

$$p_0 = B_{D0} \rho \, ; \tag{40}$$

so:

$$k_D = \frac{1}{\rho} \, ; \tag{41}$$

and the angle ϕ is given by:

$$\phi = \frac{l_D}{\rho} \, . \tag{42}$$

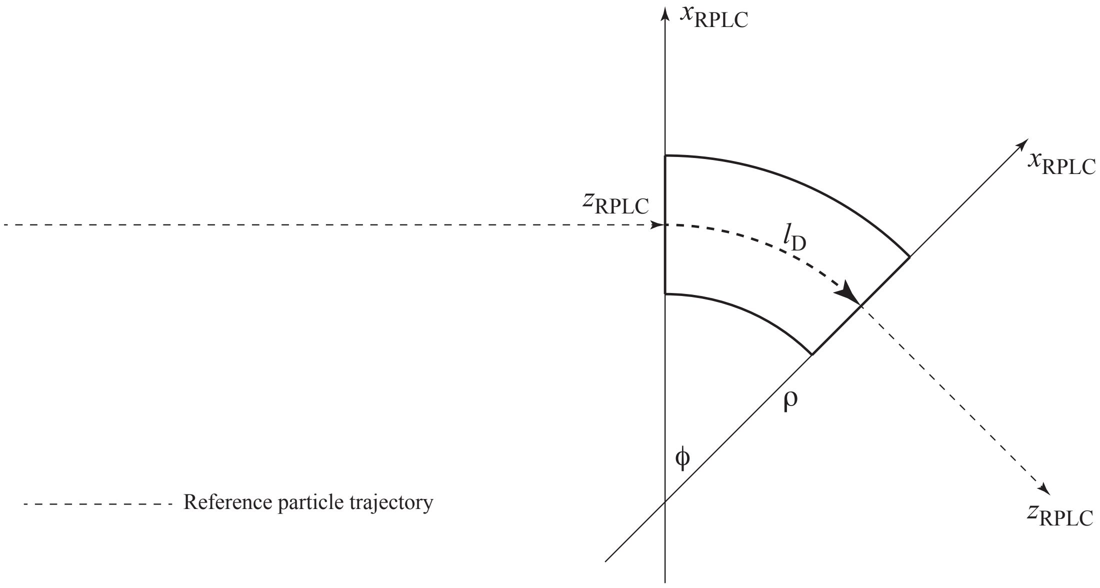

Figure 2: Schematic representation of the passage of the reference particle through a sector dipole. The outline of the sector dipole is shown by the solid black lines. The trajectory of the reference particle is shown as the dashed line. The length of the reference-particle trajectory inside the field of the sector dipole is l_D. The $x_{\rm RPLC}$ and $z_{\rm RPLC}$ coordinate axes at the entry and exit of the sector dipole are shown. The radius of curvature of the reference particle trajectory inside the magnet is ρ and the angle through which the $x_{\rm RPLC}$ is rotated is ϕ.

With these definitions the transfer matrix for passage through a dipole may be written:

$$\underline{\underline{T}}_D = \begin{pmatrix} \cos(\phi) & \rho\sin(\phi) & 0 & 0 & 0 & \frac{\rho}{\beta_0}\left(1-\cos(\phi)\right) \\ -\frac{\sin(\phi)}{\rho} & \cos(\phi) & 0 & 0 & 0 & \frac{\sin(\phi)}{\beta_0} \\ 0 & 0 & 1 & l & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -\frac{\sin(\phi)}{\beta_0} & -\frac{\rho}{\beta_0}\left(1-\cos(\phi)\right) & 0 & 0 & 1 & \frac{l}{\beta^2\gamma^2} - \frac{l-\rho\sin(\phi)}{\beta_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} . \tag{43}$$

The increment in the reference particle position is:

$$\delta s = l_D \, . \tag{44}$$

5 Source

A variety of options for the generation of the particle distribution at source are included in the package (see section **??**). The principal, and the default, option is the target-normal sheath acceleration (TNSA) model presented in [5] . The implementation of this model is summarised below. The laboratory and RPLC reference frames coincide at the target, therefore trace- and phase-coordinates are not distinguished in the presentatiaon of the particle-production model.

5.1 Energy distribution

The typical kinetic energy spectrum produced in target-normal sheath acceleration falls rapidly with kinetic energy before dropping rapidly to zero above a maximum "cut off" energy $\varepsilon_{\rm max}$. The kinetic-energy spectrum of the TNSA model presented in [5] is given by:

$$\frac{dN}{d\varepsilon} = \frac{n_{e0} c_s t_{laser} S_{sheath}}{\sqrt{2\varepsilon T_e}} \exp\left(-\sqrt{\frac{2\varepsilon}{T_e}}\right) \, ; \tag{45}$$

where N is the number of protons or ions produced per unit solid angle, ε is the ion kinetic energy, n_{e0} and T_e are the hot electron density and temperature respectively, c_s is the ion acoustic velocity, t_{laser} is the duration of the laser pulse, and S_{sheath} is the effective area over which the TNSA mechanism takes place. The variables and the units in which they are expressed are presented in table 1.

Equation 45 is based on time-limited fluid-dynamical models which are unable to predict the cut-off kinetic energy accurately. The cut-off energy is taken to be that given by the model described in [6] in which the time over which the laser pulse creates the conditions necessary for acceleration is derived. The kinetic energy cut-off is given by:

$$\varepsilon_{max} = X^2 \varepsilon_{i,\infty} \, ; \tag{46}$$

where X is obtained by solving:

$$\frac{t_{laser}}{t_0} = X\left(1 + \frac{1}{2}\frac{1}{1 - X^2}\right) + \frac{1}{4}\ln\left(\frac{1+X}{1-X}\right) \, . \tag{47}$$

Here t_0 is the time over which the ion acceleration may be treated as ballistic and $\varepsilon_{i,\infty}$ is given in table 1.

To generate the kinetic energy spectrum, the probability density function, $g(\varepsilon)$, is defined such that the probability, $\delta\mathcal{P}$, of a particle being generated in the interval $\varepsilon \to \varepsilon + \delta\varepsilon$ is given by:

$$\delta\mathcal{P} = g\left(\varepsilon\right) \delta\varepsilon \, . \tag{48}$$

$g(\varepsilon)$ can be written in terms of the differential spectrum given in equation 45 through the introduction of a normalisation constant $\mathcal{N}$:

$$g(\varepsilon) = \frac{1}{\mathcal{N}} \frac{dN}{d\varepsilon} \, . \tag{49}$$

The cumulative distribution funtion, $G(\varepsilon)$, is given by:

$$G(\varepsilon) = \int_{\varepsilon_{\text{min}}}^{\varepsilon_{\text{max}}} g(\varepsilon) d\varepsilon \, ; \tag{50}$$

where ε_{min} is the minimum kinetic energy and the normalisation constant, $\mathcal{N}$, is set so that $G(\varepsilon_{\text{max}}) = 1$. Carrying out the integration yields:

$$G(\varepsilon) = \frac{2}{\mathcal{N}} \frac{n_{e0} c_s t_{laser} S_{sheath}}{\sqrt{2T_e}} \sqrt{\frac{T_e}{2}} \left[\exp\left(-\sqrt{\frac{2\varepsilon_{\text{min}}}{T_e}}\right) - \exp\left(-\sqrt{\frac{2\varepsilon}{T_e}}\right)\right] \, ; \tag{51}$$

and the normalisation constant is given by:

$$\mathcal{N} = 2\frac{n_{e0} c_s t_{laser} S_{sheath}}{\sqrt{2T_e}} \sqrt{\frac{T_e}{2}} \left[\exp\left(-\sqrt{\frac{2\varepsilon_{\text{min}}}{T_e}}\right) - \exp\left(-\sqrt{\frac{2\varepsilon}{T_e}}\right)\right] \, . \tag{52}$$

The kinetic energy spectrum may now be obtained by choosing a value for $G(\varepsilon)$ using a probability distribution uniform over the range $0 < G(\varepsilon) < 1$. The generated value of ε is obtained by evaluating:

$$\varepsilon = \left[\sqrt{\varepsilon_{\text{min}}} - \sqrt{\frac{T_e}{2}} \ln\left(1 - \frac{G(\varepsilon)}{G(\varepsilon_{\text{max}})}\right)\right]^2 \, . \tag{53}$$

Table 1: Parameters present in the analytical expression, equation 45, describing target normal sheath acceleration (TNSA).

Parameter	Definition	Value	Unit
N	Ion number	-	-
ε	Ion kinetic energy	-	J
n_{e0}	Hot electron density	$\frac{N_E}{ct_{laser}S_{sheath}}$	pp/m^3
N_e	Accelerated electron number	$\frac{fE_{laser}}{T_e}$	-
E_{laser}	Laser energy	70	J
f	Energy conversion efficiency	$1.2\times10^{-15}I^{0.75}$, max=0.5	-
I	Laser intensity	4×10^{20}	W/cm^2
T_e	Hot electron temperature	$m_ec^2[\sqrt{1+\frac{I\lambda^2}{1.37\times10^{18}}}-1]$	J
m_e	Electron mass	9.11×10^{-31}	Kg
c	Speed of light	3×10^{8}	m/s
λ	Laser wavelength	0.8	μm
t_{laser}	Laser pulse duration	28×10^{-15}	s
B	Radius of electron bunch	$B=r_0+dtan(\theta)$	m
S_{sheath}	Electron acceleration area	πB^2	m^2
r_0	Laser spot radius	$\sqrt{\frac{P_{laser}}{I\pi}}$, I in W/m^2	m
d	Target thickness	$400-600\times10^{-9}$	m
θ	Electron half angle divergence	0.436	rad
P_{laser}	Laser power	2.5×10^{15}, $P_{laser}=\frac{E_{laser}}{t_{laser}}$	W
c_s	Ion-acoustic velocity	$(\frac{Zk_BT_e}{m_i})^{\frac{1}{2}}$	m/s
Z	Ion charge number	1	-
k_B	Boltzmann constant	1.380649×10^{-23}	$m^2kgs^{-2}K^{-1}$
m_i	Proton mass	1.67×10^{-27}	Kg
P_R	Relativistic power unit	$\frac{m_ec^2}{r_e}=8.71\times10^{9}$	W
r_e	Electron radius	2.82×10^{-15}	m
$\varepsilon_{i,\infty}$	Maximum ion kinetic energy	$2Zm_ec^2\sqrt{\frac{fP_{laser}}{P_R}}$	MeV
t_0	Ballistic time	$\frac{B}{v(\infty)}$	s
$v(\infty)$	Ballistic velocity	$\sqrt{\frac{2\varepsilon_{i,\infty}}{m_i}}$	m/s

5.2 Angular Distribution

The angular distribution of the flux of protons and ions produced by the TNSA mechanism may be described as a cone centred on the normal to the foil surface [7]. Radiochromic film has been used to observe the opening angle, 2α, of the cone as a function of energy. The envelope angle, α, defined such that, at a particular energy, all particles are contained within $\pm\alpha(\varepsilon)$ of the z axis. The opening angle is observed to decrease as the ion energy increases.

The distribution of the polar angle, θ_S, at which particles are produced at the laser-driven source is generated by defining r' such that:

$$r' = \frac{\partial r}{\partial s} ; \tag{54}$$

where $r = \sin\theta_S$. x' and y' are sampled independently from the probability density function:

$$g(r') = \frac{3}{4r_m'^2}\left(r_m'^2 - r'^2\right) ; \tag{55}$$

where $r'_m = \sin\alpha$. At low kinetic energy ($\varepsilon \sim \varepsilon_{\rm min}$), $\alpha(\varepsilon)$ is taken to be $\sim 20^\circ$. $\alpha(\varepsilon)$ is assumed to decrease linearly with energy such that:

$$\alpha(\varepsilon) = 20^\circ - 15^\circ\frac{\varepsilon}{\varepsilon_{max}} ; \tag{56}$$

i.e. $\alpha(\varepsilon)$ decreases from 20° at $\varepsilon = 0$ to 5° at ε_{max}. Finally, the azimuthal angle, ϕ_S, is chose from a distribution uniform over the range $0 < \phi_S < 2\pi$.

5.3 Spatial distribution

The x and y distributions at production are assumed to be independent and Gaussianly distributed with a standard deviation given by the radius of the laser spot focused on the target.

5.4 Simulated distributions

Distributions 10^6 protons produced by the TNSA mechanism using the algorithm described above are shown in figure 4. The parameters used in the algorithm are presented in table 2. The generated distribution of kinetic energy is in good agreement with the distribution implied by equation 45. The width of the generated polar-angle distribution is observed to fall with kinetic energy and the kinetic-energy dependence of the RMS calculated from the generated particles is in good agreement with that expected from equation 56. As a result, the distribution of θ_S is approximately Gaussian with a width dominated by the contribution of protons with kinetic energy close to $\varepsilon_{\rm min}$. The generated ϕ_S distribution is flat in the range $0^\circ < \phi_S < 360^\circ$ and the (x, y) distribution is Gaussian in both the x and y projections.

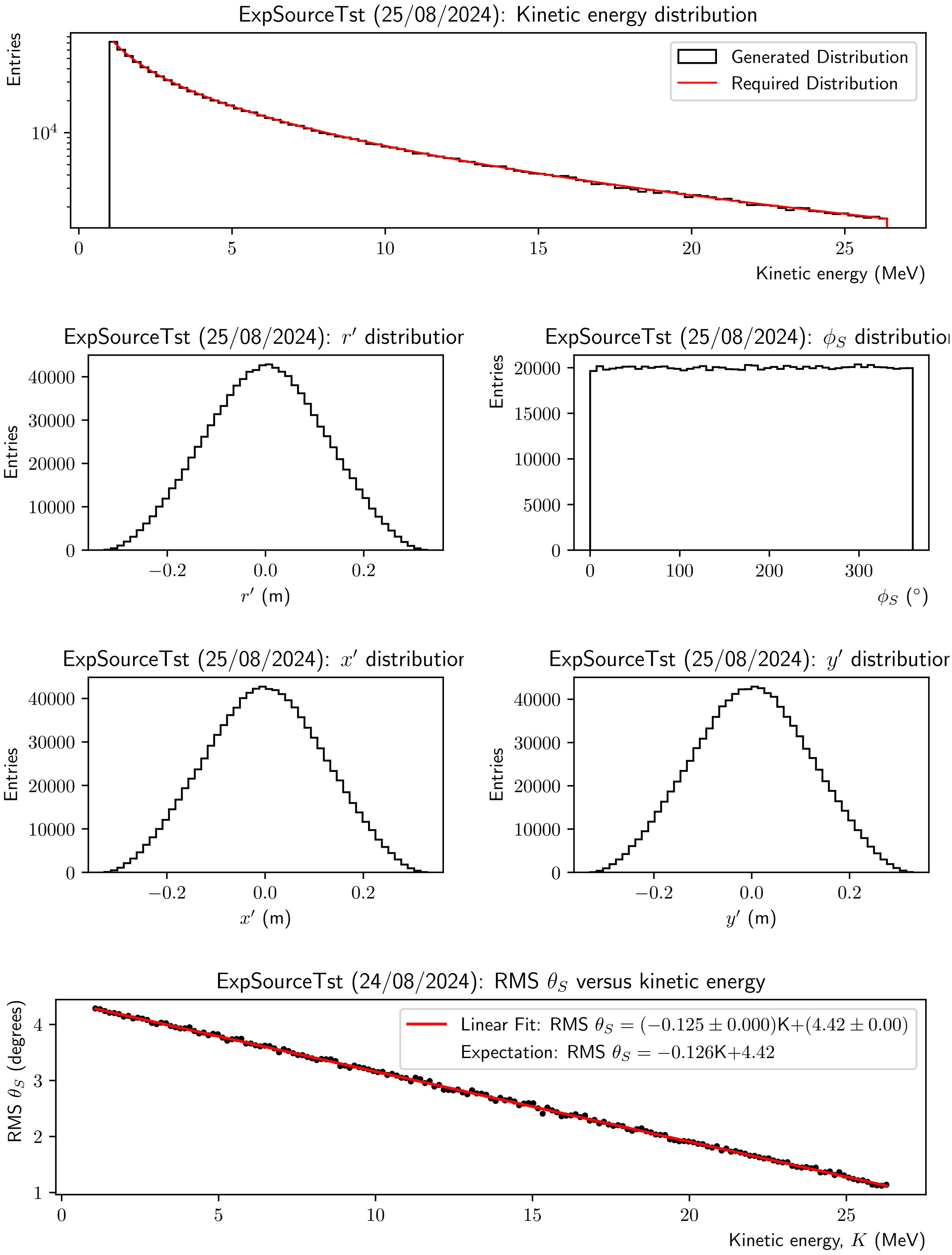

Figure 3: Kinematic distributions of particles at the point of production. Top: The generated kinetic energy disribution is shown as the solid histogram. The required distribution (equation 45), normalised to the lowest kinetic-energy bin, is shown as the solid red line. Second row: Generated r' and ϕ_S distributions. Third row: Generataed x' and y' distributions. Bottom: RMS of the θ_S distribution versus kinetic energy. The solid circles are calculated using slices of width 0.15 MeV. The red line shows the result of a straight line fit to the data. The expected dependence from integration of equation 56 is presented in the legend.

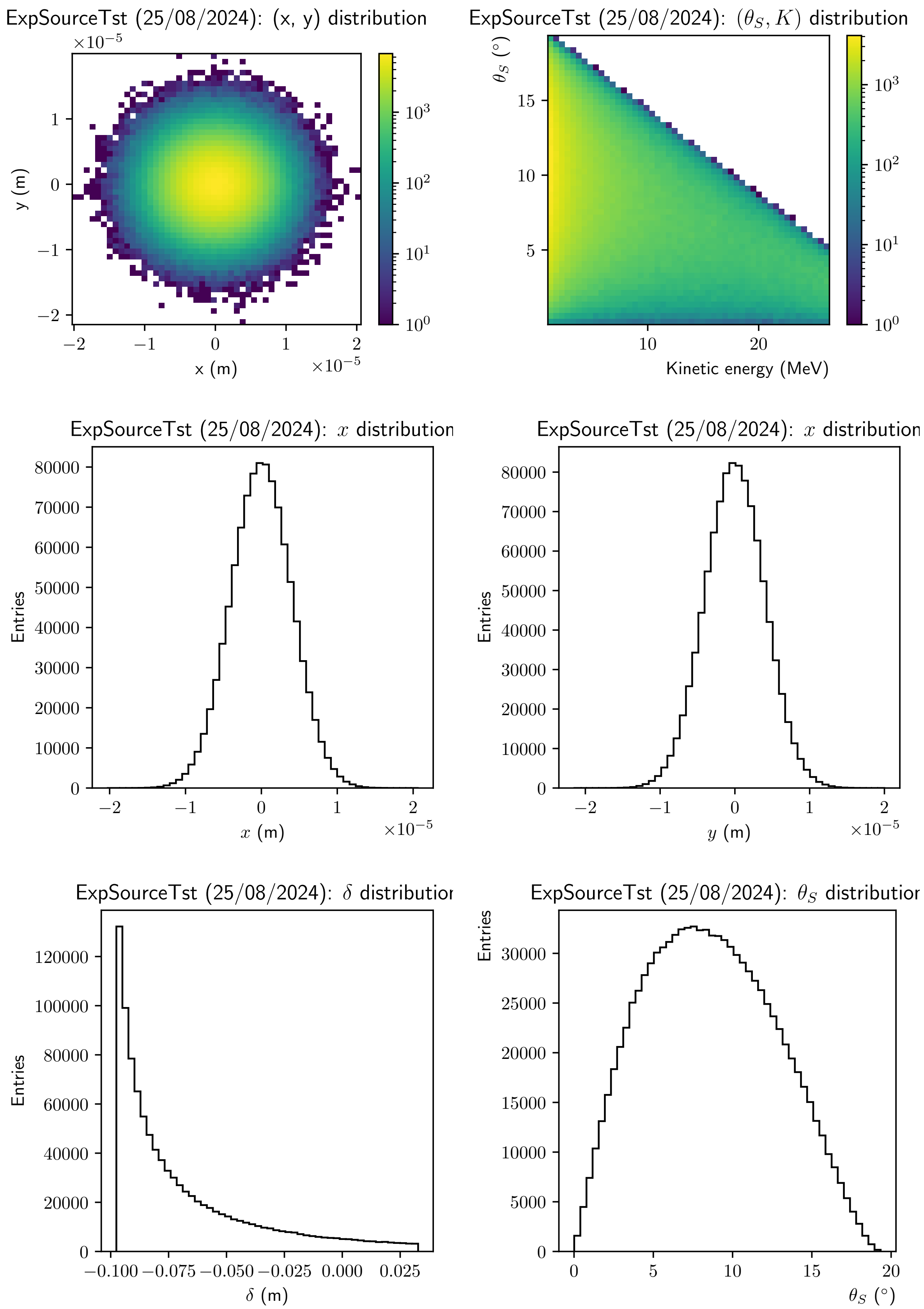

Figure 4: Top left: Generated (x, y) distribution of the particle-production point. Top right: The distribution of θ_S versus kinetic energy. Centre left, right: Gemerated disributions of x' and y'. Bottom left: Generated distribution of δ. Bottom right: Generated distribution of θ_S.

Table 2: Parameterised laser driven

Parameter	Value	Unit
σ_x	4e-06	μm
σ_y	4e-06	μm
$\cos\theta_S\|_{\rm min}$	0.998	
$\varepsilon_{\rm min}$	1.0	MeV
$\varepsilon_{\rm max}$	25.0	MeV
nPnts	1000	
Laser power	2500000000000000.0	W
Laser energy	70.0	J
Laser wavelength	0.8	μm
Laser pulse duration	2.8e-14	s
Laser spot size	4e-07	μm
Laser intensity	4e+20	J$/m^2$
Electron divergence angle	25.0	degrees
RMS θ_S at $K = 0\,$MeV	20	degrees
Scaled slope of RMS θ_S versus K	15	degrees

6 Beam-line specification and scripts

6.1 Beam-line specification

The specification is defined in the form of a parameter table. The table is loaded using a plain text file in CSV format. An example, specifying the PoPLaR beam line is shown in table 3. The simplest way to generate the CSV file is to use Excel to manipulate one of the examples provided in the `11-Parameters` directory of the `LinearOptics` package. The parameter table is organised in 8 columns:

`Stage`: (integer) allows stages in the beam line to be defined. The number is arbitrary and is used only in the generation of a unique name for each element. `Stage` can be defined for clarity in labelling and presentation.

`Section`: (string) allows sections in the beam line to be distinguished. As in the case of `Stage`, `Section` may be defined for clarity and convenience. `Section` is used in the generation of the unique name of a particular element of the beam line.

`Element`: (string) indicates that the line will define one parameter for the particular beam-line element "`Element'`". Each beam-line element defined in appendix **??** is specified by a series of consecutive lines, each line defining one of the parameters that specifies the element. The lines required to specify each element are given in appendix **??**.

`Type`: (string) specifies the type of `Element` that is to be used. For example, an `Aperture` along the beam line may be circular, elliptical, or rectangular. The `Type` keyword allows the required "type" of a particular beam-line element to be specified.

`Parameter`: (string) defines the parameter the value of which is specified by the line. For example, the length of an element is specified by setting the `Parameter` field to "`Length`".

`Value`: (float, integer) value of parameter.

`Unit`: (string) unit for parameter. Presently the code does not require `Unit` to be specified, it is included to record the unit of the value.

`Comment`: (string) free format comment.

Table 3: Example beam-line specification file in Excel (xlsx) format.

Stage	Section	Element	Type	Parameter	Value	Unit	Comment
0	Facility	Global	Name	Name	PoPLaR		
0	Facility	Global	Reference particle	Kinetic energy	10	MeV	
0	Facility	Global	Vacuum chamber	Mother volume radius	0.5	m	
1	Source	Source	Parameterised TNSA	SourceMode	0		Mode
1	Source	Source	Parameterised TNSA	SigmaX	0.000004	m	Gaussian width, x
1	Source	Source	Parameterised TNSA	SigmaY	0.000004	m	Gaussian width, y
1	Source	Source	Parameterised TNSA	Emin	1	MeV	Minimum of energy distribution
1	Source	Source	Parameterised TNSA	Emax	25	MeV	Maximum of energy distribution
1	Source	Source	Parameterised TNSA	nPnts	1000		Number of points to sample for integration of PDF
1	Source	Source	Parameterised TNSA	MinCTheta	0.999691155		Maximum theta for flat cos theta
1	Source	Source	Parameterised TNSA	Power	2.5E+15	W	Laser power
1	Source	Source	Parameterised TNSA	Energy	70	J	Laser energy
1	Source	Source	Parameterised TNSA	Wavelength	0.8	um	Laser wavelength
1	Source	Source	Parameterised TNSA	Duration	2.80E-14	s	Laser pulse duration
1	Source	Source	Parameterised TNSA	Thickness	0.0000004	m	Target thickness
1	Source	Source	Parameterised TNSA	Intensity	4.00E+20	W/cm2	Laser intensity
1	Source	Source	Parameterised TNSA	DivAngle	25	degrees	Electron divergence angle
1	Nozzle	Drift		Length	0.03	m	
1	Nozzle	Aperture	Circular	Radius	0.002	m	
1	Nozzle	Drift		Length	0.003	m	
1	Nozzle	Aperture	Circular	Radius	0.002	m	
1	Capture	Drift		Length	0.005	m	
1	Capture	Aperture	Circular	Radius	0.01	m	
1	Capture	Fquad		Length	0.04	m	
1	Capture	Fquad		Strength	150	m	
1	Capture	Aperture	Circular	Radius	0.01	m	
1	Capture	Drift		Length	0.005	m	
1	Capture	Aperture	Circular	Radius	0.01	m	
1	Capture	Dquad		Length	0.04	m	
1	Capture	Dquad		Strength	150	m	
1	Capture	Aperture	Circular	Radius	0.01	m	
1	Capture	Drift		Length	1.7	m	

The beam line is built by reading the file sequentially from the top. After the header fields, the first lines specify the Facility, giving its Name, the kinetic energy of the reference particle and radius of the "mother volume". The mother volume is a cylinder around the beam line, a particle hitting the boundary of the cylindrical volume is not propagated further along the beam line. The lines which follow specify each element of the beam line in turn, grouping them by Stage and Section as requured.

6.2 Scripts

Scripts to perform common tasks are provided in the 03-Scripts directory. The scripts and the tasks they perform are;

runBeamSim.py -b <beamlinefile> -i <inputfile> -o <outputfile> -n <nEvts>
: uses the beam line specified in <beamlinefile> to generate <nEvts> and propagate them down the beam line. The output is written to <outputfile> in the format defined in the BeamIO class. If specified, <inputfile> is a BeamIO data file. In this case, beam transport starts from the most downstream element of the beam line defined in <inputfile>. Only one of <beamlinefile> and <inputfile> may be specified.

readBeamSim.py -i <inputfile> -o <outputfile> -n <nEvts>
: reads <nEvts> events from the BeamIO-format data file <inputfile>. Two PDF files are written to 99-Scratch. The first, ParticleProgressionPlot.pdf, contains plots of the transverse trace space coordinates of the particles at the exit of each beamm line element. The second,

ParticleLongiProgressionPlot.pdf, contains plots of the longitudina trace space coordinates of the particles at the exit of each beamm line element. The option -o <outputfile> has not yet been implemented.

plotBeam.py -i <inputfile> -n <nEvts> -o <outputfile> -l <startlocation> [-b <beamlinefile>]

calculates the covariance matrix of the trace-space coordinates at the exit from each vbean-line element and uses these covariance matrices to calculate the RMS beam size and Twiss parameters. A PDF file, BeamProgress.pdf, showing the evolution of these quantities along the is written to 99-Scratch. The calculations are performed on <nEvts> read from <inputfile>. If specified, a CSV file containing a summary of the beam evolution is written to <outputfile>. Optionally, if -l is specified, the beam line evolution will be plotted from from location <startlocation>. The option -b <beamlinefile> is retained in case early versions of BeamIO-format data files are to be read.

plotextrapolatedBeam.py -i <inputfile> -n <nEvts> -o <outputfile> -l <startlocatio [-b <beamlinefile>]

calculates the covariance matrix of the beam at location <startlocation> and then calculates the evolution of the covariance matrix and Twiss parameters using the beam-line element transfer matrices. The PDF file, extrapolatedBeamProgress.pdf, is created to contain plots of the beam evolution. If <startlocation> is not specified, the covariance matrix is calculated at the first element of the beam line and the propogation of the beam envelope starts at the beginning of the beam line. The meaning of the other switches are as for plotBeam.py.

2BDSIM.py -i <inputfile> -o <outputfile> -l <start location> -n <nEvts>

reads <nEvts> events from the BeamIO-format file <inputfile> and writes the ascii file <outputfile> in the format required by BDSIM. If specified, the BDSIM-format file is generated at location <start location>. If <start location> is not specified, the BDSIM-format is created with particles at the end of the beam line.

References

[1] G. Aymar, T. Becker, *et al.*, "LhARA: The Laser-hybrid Accelerator for Radiobiological Applications," *Frontiers in Physics* **8** (2020) . `https://www.frontiersin.org/articles/10.3389/fphy.2020.567738`.

[2] The LhARA consortium, "The Laser-hybrid Accelerator for Radiobiological Applications," Tech. Rep. CCAP-TN-01, The Centre for the Clinical Application of Particles, Imperial College London, 2020. `https://ccap.hep.ph.ic.ac.uk/trac/raw-attachment/wiki/Communication/Notes/CCAP-TN-01.pdf`.

[3] The LhARA collaboration, "Baseline for the LhARA design update," Tech. Rep. CCAP-TN-11, The Centre for the Clinical Application of Particles, Imperial College London, 2022. `https://ccap.hep.ph.ic.ac.uk/trac/raw-attachment/wiki/Communication/Notes/CCAP-TN-11-LhARA-Design-Baseline.pdf`.

[4] A. Wolski, *Beam dynamics in high energy particle accelerators*. Imperial College Press, 57 Shelton Street, Covent Garden, London WC2H 9HE, 2014.

[5] J. Fuchs, P. Antici, *et al.*, "Laser-driven proton scaling laws and new paths towards energy increase," *Nature Physics* **2** (01, 2006) .

[6] J. Schreiber, F. Bell, *et al.*, "Analytical Model for Ion Acceleration by High-Intensity Laser Pulses," *Physical review letters* **97** (08, 2006) 045005.

[7] F. Nürnberg, M. Schollmeier, *et al.*, "Radiochromic film imaging spectroscopy of laser-accelerated proton beams," *Review of Scientific Instruments* **80** no. 3, (03, 2009) 033301, `https://pubs.aip.org/aip/rsi/article-pdf/doi/10.1063/1.3086424/13601448/033301_1` `https://doi.org/10.1063/1.3086424`.

A Module, class and data structures

The linear optics package has been written in object-oriented Python and is broken down in four principal modules:

- `BeamLineElement`: provides the various beam-line elements required to build a description of the beam line. Each individual element, such as a drift, quadrupole, etc., is described in a class derived from the `BeamLineElement` parent class.

- `BeamLine`: provides code to assemble the elements into a coherent beam line. `BeamLine` is a singleton class to ensure that two beam lines can not be simulated in a single run of the package. The `extrapolateBeam` class is derived from the `Beam` class to handle the propagation of beam envelopes without the need to track individual particles.

- `Beam`: provides code to calculate ensemble properties of the beam such as emittance. The ensemble properties are stored as instance attributes of the `Beam` class.

- `Particle`: provides code to record beam particles at positions along the beam line. The module provides the singleton `ReferenceParticle` class derived from the `Particle` class.

Other modules: `BeamIO`, `LaTeX`, `PhysicalConstants`, `Report`, `Simulation`, `UserFramework`, and `Visualise` support the principal modules or provide services. The data structure is implemented as attributes of the instances of the various classes. This section describes the implementation of the various modules, the classes of which they are composed, and how access to the data is provided.

Each class has methods by which to access a list of the class instances and a Boolean flag by which to generate debug print out (see table 4).

Table 4: Methods by which to set and access class attributes.

Method	**Argument**	**Return**	**Comment**
`getinstances()`		List of instances of class	For singleton classes such as `BeamLine`, `getinstances()` returns a single instance rather than a list.
`setDebug(Debug)`	Boolean		Sets flag to generate debug print-out
`getDebug()`		Boolean debug flag	If True, generate debug print-out
`setAll2None()`			Set all instance attributes to `None` at start of instantiation.
`SummaryStr()`		String	Text sring to record parameters in debug print out.

A.1 BeamLine

BeamLine is a singleton class that sets up the beam line geometry and provides methods to track particles through the beam line using the transfer matarices defined in section A.4. The beam-line geometry is provided in the form of a "csv" file read using pandas. The format of the "csv" file is defined in section ??. Alternatively, if a data file written using the BeamIO package is being read, the beam-line geometry is read from the top of the data file. The instance of the BeamLine class is created when the first record of the data file is read.

A.1.1 Instantiation

The call to instantiate the BeamLineElenent class is:

```
BeamLine(BeamLineSpecificationCSVfile, readDataFile)
```

BeamLineSpecificationCSVfile is a the full path of the CSV file containing the beam-line specificiation. readDataFile is a boolean flag. If readDataFile is set to True, then the BeamLine instance will be created and the beam-line geometry will be read from the header of the BeamIO data file. If readDataFile is not set or is set to False, the beam-line geometry will be read from BeamLineSpecificationCSVfile.

A.1.2 Instance attributes and access methods

The instance attributes are presented in table 5 and the access methods are summarised in table 6.

Table 5: Definition of attributes of instances of the BeamLine class.

Attribute	Type	Comment
BeamLineSpecificationCSVfile*	path	Full path to beam-line specification csv file.
BeamLineParamPandasInstance	dataframe	Pandas data frame containing beam-line specification.
Element	list	List of instances of BeamLineElement class containing pointers to the elements that make up the beam line.

A.1.3 Processing methods

Table 7 presents the processing methods provided in the BeamLine class.

A.1.4 I/o methods

Table 8 presents the i/o methods provided in the BeamLine class.

A.1.5 Utilities

Table 9 presents the utilities provided in the `BeamLine` class.

Table 6: Definition of access methods for the `BeamLine` class.

Set method	**Get method**	**Comment**
`setSrcTrcSpc(SrcTrcSpc)`	`getSrcTrcSpc()`	Set trace space at source; SrcTrcSpc presented as `(1,6) np.ndarray`.
	`getinsance()`	Get instance of `BeamLine` class.
	`BeamLineSpecificationCSVfile()`	Get beam line specification csv file.
	`(getBeamLineParamPandas)`	Get pandas dat from containing beam-line specification.
	`getElement()`	get list of `BeamLineElement` instances.

Table 7: Processing methods provided by the `BeamLine` class.

Method	Argument(s)	Return	Comment
`addBeamLine()`		`Success`	Loops through pandas data frame and manages parsing and instanciation of the beam line elements defined in the specification `csv` file. Returns `Success` (bool) which is `True` if the beamline has been set up OK and is `False` otherwise.
`addFacility()`			Manages the extraction of the facility parameters from the pandas data frame and the creation of the single instance of `Facility(BeamLineElement)`.
`addSource()`			Manages the extraction of the source parameters from the pandas data frame and the creation of the single instance of `Source(BeamLineElement)`.
`parseFacility()`		`Name`, `K0`, `VCMVr`	Parses pandas data frane to extract facility parameters. Returns the facility `Name` (str), the kinetic energy of the reference particle, `K0` (float) in MeV, and the vacuum chamber mother volume radius, `VCMVr` (float) in m.
`parseSource()`		`Name`, `Mode`, `Param`	Parses pandas data frane to extract source parameters. Returns `Name` (str), `Mode` (int) `Param` (list) containing the parameters required to instanciate source `Mode`.
`addBeamLineElement(iBLE)`			Adds `BeamLineElement` instance `iBLE` to the list of instances of `BeamLineElement` that make up the beam line.
`checkConsistency()`		`Consistent`	Checks the consistency of the beam line representation in memory with that requested in the specification `csv` file. Returns `Consistent` (bool) which is `True` if the beamline is consistent is `False` otherwise.
`trackBeamn(NEvts, particleFILE)`			Generates `NEvts` (int) particles and tracks them through the beam line.

Table 8: I/o methods provided by the `BeamLine` class.

Method	**Argument(s)**	**Return**	**Comment**
`csv2pandas(csvFILE)`	Path	pandas dataframe	Read CSV file to create pandas data frame. `csvFILE` (path) is the full path to the `csv`
`pandasBeamLine()`		pandas dataframe	Create pandas dataframe from `BeamLine` instance.
`getHeader()`		List	Prepares list of header fields for `pandasBeamLine`.
`readBeamLine(file)`	Path	Boolean	Called from `BeamIO`. Reads `BeamLine` from data file.

Table 9: Utilities provided by the `BeamLine` class.

Method	**Argument(s)**	**Return**	**Comment**
`cleaninstance()`			Remove `BeamLine` instance.
`fixsz()`		List	Loop through `BeamLineElement` instances to set s and z at exit.

A.2 `Particle` and `ReferenceParticle`

The `Particle` class provides methods to transport particles through the beam line. The trace and phase space is recorded at the start and end of each element. The `ReferenceParticle` derived class is a singleton and records the trajectory of the reference particle.

A.2.1 `Particle`

A.2.1.1 Instantiation

The call to instantiate the `Particle` class is:

```
Particle(Species)
```

`Species`, the type of particle to be propagated, is a string containing the particle name. At present valid particle `species` are `proton`, `pion`, and `muon`.

A.2.1.2 Instance attributes and access methods

The instance attributes are presented in table 10 and the access methods are summarised in table 11.

Table 10: Definition of attributes of instances of the `Particle` class.

Attribute	Type	Comment
`Species`	str	Particle species; proton, muon or pion.
`Location`	list	List of strings containing the unique `Name` of the `BeamLineElement` at the particle position is reported.
`s`	list	List of floats recording s coordinate at which particle position is reported.
`TraceSpace`	list	List of `np.ndarray` containing 6D trace space of particle at `s`.
`PhaseSpace`	list	List of `np.ndarray` containing 6D phase space (RPLC) of particle at `s`.
`LabPhaseSpace`	list	List of `np.ndarray` containing 6D phase space (Lab) of particle at `s`.

A.2.1.3 Processing methods

Table 12 presents the processing methods provided in the `Particle` class.

A.2.1.4 I/o methods

The i/o methods provided by the `Particle` class are listed in table 13.

A.2.1.5 Utilities

The utilities provided by the `Particle` class are listed in table 14.

Table 11: Definition of access methods for the `Particle` class.

Set method	**Get method**	**Comment**
`setSpecies`	`getSpecies`	Set/get particle species.
`setLocation`	`getLocation`	Set/get list of locations location.
`sets`	`gets`	Set/get list of s coordinates.
`setTraceSpace`	`getTraceSpace`	Set/get list of trace-space vectors.
`setRPLCPhaseSpace`	`setRPLCPhaseSpace`	Set/get list of phase-space vectirs in RPLC coordinates.
`setLabPhaseSpace`	`getLabPhaseSpace`	Set/get list of phase-space vectirs in Lab coordinates.
`resetParticleInstances`		Resets list of particle instances preserving reference particle as firt instance in the list.
`recordParticle(Loc,z,s,TrcSpc)`		Records particle attributes. Arguments: `Loc`=Location, `z`= z, `s`= s, and `TrcSpc`=trace space.
`setSourceTraceSpace(TrcSpc)`		Sets `TrcSpc`=trace space at source.

Table 12: Processing methods provided by the `Particle` class.

Method	**Argument(s)**	**Return**	**Comment**
`fillPhaseSpaceAll()`		Boolean	Fill phase space for all `Particle` instances. Class Method. Return `True` if successful.
`fillPhaseSpace()`		Boolean	Fill phase space for current `Particle` instance. Return `True` if successful.
`initialiseSums()`			Initialises sums used to calculate covariance matrix.
`incrementSums(iPrtcl)`	`Particle` instance		Increment sums used to calculate covariance matrix.
`calcCovarianceMatrix()`			Calculate covariance matrix.
`evaluateBeam()`			Work through locations and calculate parameters from covariance matrix.
`calcRPLCPhaseSpace(nLoc)`	Int	np.ndarray	Calculate and return phase space in RPLCs at location `nLoc`.
`RPLCTraceSpace2PhaseSpace(TrcSpc)`	np.ndarray	np.ndarray	Transform trace space to phase space in RPLCs.
`RPLCPhaseSpace2TraceSpace(TrcSpc)`	np.ndarray	np.ndarray	Transform phase space to trace space in RPLCs.

Table 13: I/o methods provided by the `Particle` class.

Method	**Argument(s)**	**Return**	**Comment**
`createParticleFile(path, file)`	Path, Str	Path	Class method, kept for backward compatibility.
`flushNcloseParticleFile(file)`	Path		Class method, kept for backward compatibility.
`openParticleFile(path, file)`	Path, Str		Class method, kept for backward compatibility.
`closeParticleFile(path, file)`	Path, Str		Class method, kept for backward compatibility.
`readParticle(file)`	Path	Boolean	Read particle from input stream. Called from `BeamIO`. `file` is full path to file. Return `True` if end of file.
`writeParticle(file, clean)`	Path, boolean		Write particle to output stream. `file` is full path to file. If `clean`, then clean particle instance after write.
`writeParticleBDSIM(file, iLoc, clean)`	Path, integer, boolean		Write particle to BDSIM ascii file. `file` is full path to file. `iLoc` is the location along the beamm line at which to write the particle coordinates. If `clean`, then clean particle instance after write.

Table 14: Utilities provided by the `Particle` class.

Method	Argument(s)	Return	Comment
`cleanAllParticles()`			Delete all `Particle` instances including `ReferenceParticle`.
`cleanParticles()`			Delete all `Particle` instances except `ReferenceParticle`.
`plotTraceSpaceProgression()`			Plot transverse trace space at each location. Class method. Writes file to `99-Scratch/`.
`plotLongitudinalTraceSpaceProgression()`			Plot longitudinal trace space at each location. Class method. Writes file to `99-Scratch/`.
`printProgression()`			Print particle parameters at each location.
`getLines()`			Returns lines to be used to create summary pandas data frame.
`createReport()`			Creates `csv` file containing summary of beam progression.

A.2.2 ReferenceParticle

A.2.2.1 Instantiation

ReferenceParticle is a singleton derived class. The call to instantiate the ReferenceParticle class is:

ReferenceParticle(Species)

Species, the type of particle to be propagated, is a string containing the particle name. At present valid particle species are proton, pion, and muon.

A.2.2.2 Instance attributes and access methods

In addition to the instance attributes inheritted from the parent class, the ReferenceParticle class provides the instance attributes presented in table 15 and the access methods are summarised in table 16.

Table 15: Definition of attributes of instances of the ReferenceParticle class.

Attribute	Type	Comment
_sIn[]	list	List of floats containing the s coordinates at the entrance to the beam line elements along the locus of the reference particle trajectory.
_sOut[]	list	List of floats containing the s coordinates at the exit to the beam line elements along the locus of the reference particle trajectory.
_RrIn[]	list	List of ndarrays containing the four-vector position in laboratory coordinates at the entrance to the beam line elements along the locus of the reference particle trajectory.
_RrOut[]	list	List of ndarrays containing the four-vector position in laboratory coordinates at the exit to the beam line elements along the locus of the reference particle trajectory.
_PrIn[]	list	List of ndarrays containing the four-vector momentum in laboratory coordinates at the entrance to the beam line elements along the locus of the reference particle trajectory.
_PrOut[]	list	List of ndarrays containing the four-vector momentum in laboratory coordinates at the exit to the beam line elements along the locus of the reference particle trajectory.
_Rot2LabIn[]	list	List of ndarrays containing the rotation matrices taking RPLC to laboratory coordinates at the entrance to the beam line elements along the locus of the reference particle trajectory.
_Rot2LabOut[]	list	List of ndarrays containing the rotation matrices taking RPLC to laboratory coordinates at the exit to the beam line elements along the locus of the reference particle trajectory.

Table 16: Definition of access methods for the `ReferenceParticle` class.

Set method	Get method	Comment
`setRPDebug`	`getRPDebug`	Set/get `ReferenceParticle` debug flag.
`setsIn`	`getsIn`	Set/get `_sIn`
`setsOut`	`getsOut`	Set/get `_sOut`
`setRrIn`	`getRrIn`	Set/get `_RrIn`
`setRrOut`	`getRrOut`	Set/get `_RrOut`
`setPrIn`	`getPrIn`	Set/get `_PrIn`
`setPrOut`	`getPrOut`	Set/get `_PrOut`
	`getMomentumIn(iLoc)`	Get magnitude of three-vector momentum at entrance of location `iLoc`
	`getMomentumOut(iLoc)`	Get magnitude of three-vector momentum at exit of location `iLoc`
`setRot2LabIn`	`getRot2LabIn`	Set/get `_Rot2LabIn`
`setRot2LabOut`	`getRot2LabOut`	Set/get `_Rot2LabOut`
	`getb0(iLoc)`	Get β_0.
	`getg0b0(iLoc)`	Get $\gamma_0\beta_0$.
`setAllRP2None`		Set all `ReferenceParticle` attributes to None.

Table 17: Processing methods provided by the `ReferenceParticle` class.

Method	Argument(s)	Return	Comment
`setReferenceParticleAtSource()`		boolean	Set `ReferenceParticle` attributes at source. Returns True if success.
`setReferenceParticleAtDrift(iBLE)`	BLE	boolean	Set `ReferenceParticle` attributes at `BeamLineElement` (BLE) instance iBLE. Returns True if success.

A.2.2.3 Processing methods

Table 17 presents the processing methods provided in the `ReferenceParticle` class.

A.2.2.4 I/o methods

The `ReferenceParticle` class provides no additional i/o methods.

A.2.2.5 Utilities

The `ReferenceParticle` class provides no additional utilities.

A.3 `Beam` and `extrapolateBeam`

The `Beam` class is in some sense a "sister" class to Particle. Whereas an instance of `Particle` records the passage of a particle travelling through the beam line, an instance of `Beam` records the collective properties of the beam such as emittance, as the beam progresses through the beam line. The beam parameters reported in the attributes of an instance of `Beam` are obtained by summing over `nEvtsMax` particles to evaluate the covariance matrices by location.

The `extrapolatedBeam` class is derived from `Beam`. An instance of `extrapolatedBeam` calculates the covariance matrix at a location along the beam line and then uses the transfer matrices to propagate the beam envelope.

A.3.1 `Beam`

A.3.1.1 Instantiation

The call to instantiate the `Beam` class is:

```
Beam(InputDatafile, nEvtMax, outputCSVfile, startlocation,
                    beamlineSpecificationCSVfile))
```

`InputDatafile` is either the full path to a data file in one of the formats specified in `BeamIO` or an instance of the `BeamIO` class that refers to an existing data file that can be read. `nEvtMax` is the maximum number of events to read or process using the `Beam` instance. `outputCSVfile` is the full path to the output file, in CSV format, contining the beam paramters at the locations traversed by the beam. `startlocation` is the location along the beam line at which propagation will start; if `startlocation` is absent or None propagation will start at the source. `beamlineSpecificationCSVfile` is the CSV file specifying the beam line. `beamlineSpecificationCSVfile` is kept for backward compatibilty. If the first record of `InputDatafile` contains the specification of the beam line, `beamlineSpecificationCSVfile` is not required.

A.3.1.2 Instance attributes and access methods

The instance attributes are presented in table 18 and the access methods are summarised in table 19.

A.3.1.3 Processing methods

Table 20 presents the processing methods provided in the `Beam` class.

A.3.1.4 I/o methods

The `Beam` class has no i/o methods.

A.3.1.5 Utilities

The utilities provided by the `Beam` class are listed in table 21.

Table 18: Definition of attributes of instances of the `Beam` class.

Attribute	**Type**	**Comment**
`InputDataFile`	Path/BeamIO	Path to, or `BeamIO` instance of input file.
`nEvtMax`	int	Maximum number of particles to read and process.
`outputCSVfile`	Path	Path to output CSV file to contain beam parmeters by location.
`startlocation`	int	Location at which to start beam propagation.
`BLspecCSVfile`	Path	`beamlineSpecificationCSVfile`, kept for backward compatibility.
`Location[iLoc]`	List	List of locations (int) at which beam parameters are recorded.
`s[iLoc]`	List	List of s coordinates of reference particle at locations at which beam parameters are recorded.
`nParticles[iLoc]`	List	List of number of particles (int) recorded at location `iLoc`.
`CovMtrx[iLoc][6,6]`	List	List of covariance matrices (ndarray) by location.
`sigmaxy[iLoc][2]`	List	List of σ_x = `sigmaxy[iLoc][0]` and σ_y = `sigmaxy[iLoc][1]` (float).
`emittance[iLoc][5]`	List	List of emittance by location: ϵ_x = `emittance[iLoc][0]`, ϵ_y = `emittance[iLoc][1]`, $\epsilon_{\rm L}$ = `emittance[iLoc][2]`, $\epsilon_{\rm 4D}$ = `emittance[iLoc][3]`, $\epsilon_{\rm 6D}$ = `emittance[iLoc][4]`.
`Twiss[iLoc][2][3]`	List	List of Twiss parameters by location: `Twiss[iLoc][0][0:2]` = $[\alpha_x, \beta_x, \gamma_x]$, `Twiss[iLoc][1][0:2]` = $[\alpha_y, \beta_y, \gamma_y]$.

Table 19: Definition of access methods for the `Beam` class.

Set method	Get method	Comment
`setInputDataFile`	`getInputDataFile`	Set path to input file.
`setBeamIOread`	`getBeamIOread`	Set instance of `BeamIO` for input file.
`setnEvtMax`	`getnEvtMax`	Set maximum number of particles to deal with.
`setoutputCSVfile`	`getoutputCSVfile`	Set pathh to output CSV file.
`setstartlocation`	`getstartlocation`	Set start location.
`setbeamlineSpecificationCSVfile`	`getbeamlineSpecificationCSVfile`	Set beam line specification file.
`sets`	`sets`	Set s by location.
`setsigmaxy`	`setsigmaxy`	Set $\sigma_{x,y}$ by location.
`setEmittance`	`setEmittance`	Set emittance list by location.
`setTwiss`	`setTwiss`	Set Twiss paramter list by location.
`resetBeamInstances`		Set list of beam instances to `[]`.
	`getCovSums`	Get list of sums used to calculate covariance matrices.
	`getCovMtrx`	Get list of covariance matrices by location.
	`getnParticles`	Get number of particles entering covariance sums by location.
	`getCovarianceMatrix`	Get list of covariance matrices by location.
	`getCovMtrx`	Get list of covariance matrices by location.

Table 20: Processing methods provided by the `Beam` class.

Method	**Argument(s)**	**Return**	**Comment**
`initialiseSums()`			Initialise sums to be used to calculate covariance matrices by location.
`incfementSums(iPrtcl)`	Particle		Increment sums for covariance matrix calculation for `Particle` instance `iPrtcl`.
`calcCovarianceMatrix()`			Calculate covariance matrices by location.
`evaluateBeam(TrackBeam=False)`	boolean		Loop over `nEvtMax` partices to calculate beam parameters by location. If `TrackBeam` is True, particles are transported through the beam line. If `TrackBeam` is False, covariance matrices stored as `Beam` instance attributes are used.

Table 21: Utilities provided by the `Beam` class.

Method	Argument(s)	Return	Comment
`cleanBeams()`		boolean	Delete all `Beam` instances, returns True if successful.
`printProgression()`			Print beam parameters by location.
`getHeader()`		list	Prepare header for CSV file containing beam parameters by location.
`getLines()`		list	Prepare lines for CSV file containing beam parameters by location.
`createReport()`			Interface to `Report` class to make beam paramter CSV file.
`plotBeamProgression(plotFILE)`		path	Plot beam parameters by location; plotFILE is full path to plot file.

A.3.2 `extrapolateBeam`

A.3.2.1 Instantiation

The call to instantiate the `extrapolateBeam` class is:

```
Beam(InputDatafile, nEvtMax, outputCSVfile, startlocation,
                beamlineSpecificationCSVfile)
```

The arguments are passed directly to a call to instantiate the parent `Beam` class. In the execution of `extrapolateBeam`, the covariance matrices are calculated from `nEvtMax` at `startlocation`. If `startlocation`, the last location provided in the input data file is used as the source.

A.3.2.2 Instance attributes and access methods

Instance attributes are inheritted from `Beam` (table 18). Access methods are also inheritted from `Beam` (table 19). The method `resetextrapolateBeamInstances` is provided to reset only the list of instances of `extrapolateBeam`.

A.3.2.3 Processing methods

Table 22 presents the processing methods provided in the `extrapolateBeam` class.

Table 22: Processing methods provided by the `extrapolateBeam` class.

Method	**Argument(s)**	**Return**	**Comment**
`initialiseSums()`			Initialise sums to be used to calculate covariance matrices at `startlocation`.
`incfementSums(iPrtcl)`	Particle		Increment sums for covariance matrix calculation for `Particle` instance `iPrtcl` at `startlocation`.
`extrapolateCovarianceMatrix()`			Extrapolate covariance matrices along the beam line by location.
`extrapolateBeam()`			Estrapolate beam envelope and beam parameters along the beam line by location.

A.3.2.4 I/o methods

The `extrapolateBeam` class has no i/o methods.

A.3.2.5 Utilities

The utilities provided by the `extrapolateBeam` class are listed in table 23.

Table 23: Utilities provided by the `extrapolateBeam` class.

Method	Argument(s)	Return	Comment
`cleanextrapolateBeams()`		boolean	Delete all `extrapolateBeam` instances, returns True if successful.

A.4 `BeamLineElement`

A.4.1 Parent class

A.4.1.1 Instantiation

The call to instantiate the `BeamLineElenent` class is:

```
BeamLineElement(Name, rStrt, vStrt, drStrt, dvStrt)
```

where:

`Name`: (string) is the unique name of the element;

`rStrt`: (numpy.ndarray(3)) is the three-vector position in laboratory coordinates of the start of the element;

`vStrt`: (numpy.ndarray(1,2)) is the polar, θ, and azimuthal, ϕ, angles that define the y ($i = 0$) and z ($i = 1$) axes of the RPLC coordinate system at the start of the element (`vStrt` $= [[i], [\theta, \phi]]$);

`drStrt`: (numpy.ndarray(3)) error in the three-vector position with respect to the nominal position; and

`dvStrt`: (numpy.ndarray(1,2)) error in the polar and azimuthal angles defining RLPC the y and z axes.

All arguments are required.

A.4.1.2 Instance attributes and access methods

Properties common to all beam-line elements are stored as instance attributes of the parent `BeamLineElement` class. The instance attributes are defined in table 24. The attributes are accessed and set using the methods defined in table 25.

A.4.1.3 Processing methods

Table 26 presents the processing methods provided in the `BeamLineElement` class.

A.4.1.4 I/o methods

Methods to read and write instance attributes to the files defined using the `BeamIO` package (see section **??** are provided. The calls are:

`readElement(dataFILE)` and `writeElement(dataFILE);`

where `dataFILE` is `BeamIO` instance.

A.4.1.5 Utilities

Table 27 presents the utilities provided in the `BeamLineElement` class.

Table 24: Definition of attributes of instances of the `BeamLineElement` class. The attributes marked * above the dividing line are required in the call to instantiate the element. The attributes marked † below the dividing line are calculated.

Attribute	**Type**	**Unit**	**Comment**
`Name`*	String		Name of beam-line element.
`rStrt`*	numpy.ndarray	m	$[x, y, z]$ position of entrance to element in laboratory coordinate system.
`vStrt`*	numpy.ndarray	rad	$[[i], [\theta, \phi]]$ (polar and azimuthal angles) of RPLC y and z axes ($i = 0, 1$ respectively) at start.
`drStrt`*	numpy.ndarray	m	"Error", $[x, y, z]$, displacement of start from nominal position (not yet implemented).
`dvStrt`*	numpy.ndarray	rad	"Error", $[[i], [\theta, \phi]]$, deviation in θ and ϕ from nominal axis (not yet implemented).
`Strt2End`†	numpy.ndarray		1×3 translation from start of element to end; in laboratory coordinates. Set in derived class.
`Rot2LbStrt`†	numpy.ndarray		3×3 rotation matrix that takes RPLC axes to laboratory axes at start.
`Rot2LbEnd`†	numpy.ndarray		3×3 rotation matrix that takes RPLC axes to laboratory axes at end. Set in derived class.
`TnrsMtrx`†	numpy.ndarray		3×3 transfer matrix. Set in derived class.

Table 25: Definition of access methods for the `BeamLineElement` class.

Set method	Get method	Comment
`setName(Name)`	`getName()`	Set/get name of beam-line element.
`setrStrt(rStrt)`	`getrStrt()`	Set/get laboratory $[x, y, z]$ position of entrance.
`setvStrt(vStrt)`	`getvStrt()`	Set/get RPLC $[\theta, \phi]$ of principal axis at start of element.
	`getvEnd()`	Set/get RPLC $[\theta, \phi]$ of principal axis at end of element.
`setdrStrt(drStrt)`	`getdrStrt()`	Set/get "error" displacement.
`setdvStrt(dvStrt)`	`getdvStrt()`	Set/get "error" deviation in $[\theta, \phi]$.
`setLength(length)`	`getLength`	Set/get increment in s across element, (length for elements that do not bend beam).
`setRot2LbStrt()`	`getRot2LbStrt()`	Set/get rotation matrix from RPLC axes to laboratory.
`setRot2LabStrt()`	`getRot2LbStrt()`	Setget rotation matrix from RPLC to laboratory at start.
`setStrt2End(t)`	`getStrt2End()`	Set/get displacement vector start to end in laboratory coordinates. setStrt2End takes 1 argument, `t`, a 1D np.ndarray containing the translation from the start to the end of the element in RPLC.
`setRot2LbEnd(R)`	`getRot2LbEnd()`	Set/get rotation matrix from RPLC to laboratory at end. setRot2LbEnd takes 1 argument, `R`, a 2D np.array containing the rotation matrix to be set.
	`getTransferMatrix()`	Get transfer matrix set in derived class.
	`getLines()`	Get lines to write LaTeX specification of element.

Table 26: Processing methods provided by the `BeamLineElement` class.

Method	**Argument(s)**	**Return**	**Comment**
`OutsideBeamPipe(R)`	Float	Boolean	Returns False if particle is inside beam pipe. If `R`, radial distance from z axis in RPLC, falls outside beam pipe, returns True.
`ExpansionParameterFail(R)`	Float	Boolean	Calculates an approximate expansion parameter and returns False if the parameter is large (> 1). Not yet used in `Transport`.
`Transport(V)`	6×1 np.ndarray	6×1 np.ndarray	Transport 6D trace-space vector, `V`, across element. Final trace-space vector returned.
`Shit2Local(V)`	6×1 np.ndarray	6×1 np.ndarray	Transform 6D trace-space vector, `V`, from RPLC to laboratory coordinates. Phase-space vector in laboratory frame returned.
`Shit2Laboratory(U)`	6×1 np.ndarray	6×1 np.ndarray	Transform 6D phase-space vector, `U`, from laboratory coordinates to trace-space coordinates in the RPLC frame. Trace-space vector in RLPC frame returned.

Table 27: Utilities provided by the BeamLineElement class.

Method	Argument(s)	Return	Comment
cleaninstances()			Delete (using "del") all instances of the BeamLineElement class. Reset instances list.
removeInstance(inst)	Instance of BLE		Remove instance inst and remove from list of instances of BeamLineElement.
visualise(axs, CoordSys,Proj)	axs – MatPlotLib "axes" instance		Manages plotting (visualisation) of element.
	CoordSys – string		"Lab" or "RPLC", coordinate system in which to visualise element.
	Proj – string		"xz" or "yz" projection to visualise.

A.4.2 Derived class: `Facility(BeamLineElement)`

A.4.2.1 Instantiation

The call to instantiate the `Facility` derived class is:

```
FacilityName, rStrt, vStrt, drStrt, dvStrt, p0, VCMV)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`. The `Facility` arguments are translated into instance attributes as described in section A.4.2.2 and defined in table 28.

A.4.2.2 Instance attributes and access methods

The instance attributes are defined in table 28. The attributes are accessed and set using the methods defined in table 29.

Table 28: Definition of attributes of instances of the `Facility(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`p0`	float	MeV	Kinetic energy of reference particle.
`VCMV`	float	m	Radius of vacuum-chamber mother volume. The radius defines edge of the volume at which a particle trajectory is terminated. It may be necessary to introduce a beam pipe later.

Table 29: Definition of access methods for the `Facility` derived class.

Set method	Get method	Comment
`setp0(Name)`	`getp0()`	Set/get momentum of reference particle (in MeV).
`setVCMV(VCMV)`	`getrVCMV()`	Set/get radius of vacuum chamber mother volume.

A.4.2.3 Processing methods

The `Facility` derived class has no processing methods other than those inheritted from the parent class.

A.4.2.4 I/o methods

The `Facility` derived class has no i/o methods other than those inheritted from the parent class.

A.4.2.5 Utilities

The `Facility` derived class has no utilities other than those inheritted from the parent class.

A.4.3 Derived class: `Drift(BeamLineElement)`

A.4.3.1 Instantiation

The call to instantiate the `Drift` derived class is:

```
Drift(Name, rStrt, vStrt, drStrt, dvStrt, Length)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`.

A.4.3.2 Instance attributes and access methods

The instance attributes are defined in table 30. The attributes are accessed and set using the methods defined in table 31.

Table 30: Definition of attributes of instances of the `Drift(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Length`	float	m	Length of drift.

Table 31: Definition of access methods for the `Facility` derived class.

Set method	Get method	Comment
`setLength(Length)`	`getLength()`	Set/get length of drift (in m).
`setTransferMatrix()`		Set transfer matrix.

A.4.3.3 Processing methods

The `Drift` derived class has no processing methods other than those inheritted from the parent class.

A.4.3.4 I/o methods

The `Drift` derived class has no i/o methods other than those inheritted from the parent class.

A.4.3.5 Utilities

The `Drift` derived class has no utilities other than those inheritted from the parent class.

A.4.4 Derived class: `Aperture(BeamLineElement)`

A.4.4.1 Instantiation

The call to instantiate the `Aperture` derived class is:

```
Aperture(Name, rStrt, vStrt, drStrt, dvStrt, ParamList)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`.

A.4.4.2 Instance attributes and access methods

The instance attributes are defined in table 32. The attributes are accessed and set using the methods defined in table 34.

A.4.4.3 Processing methods

The `Aperture` processing method is defined in table 34.

Table 32: Definition of attributes of instances of the `Aperture(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	**Type**	**Unit**	**Comment**
`ParamList`	[]		List containing aperture parameters. The first parameter is an `int` and defines the aperture "`Type`". The remaining elements in the parameter list are `floats` with meanings that depend on `Type`.
`ParamList[0]`	int		`Type`$=0$; circular
`ParamList[1]`	float	m	Radius of circular aperture
`ParamList[0]`	int		`Type`$=1$; Elliptical
`ParamList[1]`	float	m	Radius of elliptical aperture along $x_{\rm RPLC}$ axis
`ParamList[2]`	float	m	Radius of elliptical aperture along $y_{\rm RPLC}$ axis
`ParamList[0]`	int		`Type`$=2$; Rectangular
`ParamList[1]`	float	m	Size of aperture along $x_{\rm RPLC}$ axis
`ParamList[2]`	float	m	Size of aperture along $y_{\rm RPLC}$ axis

Table 33: Definition of access methods for the `Aperture` derived class.

Set method	**Get method**	**Comment**
`setApertureParameters(ParamList)`		Set aperture parameters. Sets `Type` and parameters depending on `Type`.
	`getType()`	Get `Type` of aperture.
	`getParams()`	Get aperture parameters.

Table 34: Utilities provided by the `Aperture` derived class.

Method	**Argument(s)**	**Return**	**Comment**
`Transport(V)`	`np.ndarray`	`np.ndarray` or `None`	Transport trace-space vector `V`. If `V` falls outside of the aperture, return `None`.

A.4.4.4 I/o methods

The `Aperture` derived class has no i/o methods other than those inheritted from the parent class.

A.4.4.5 Utilities

The `Aperture` derived class has no utilities other than those inheritted from the parent class.

A.4.5 Derived class: `FocusQuadrupole(BeamLineElement)`

A.4.5.1 Instantiation

The call to instantiate the `FocusQuadrupole` derived class is:

```
FocusQuadrupole(Name, rStrt, vStrt, drStrt, dvStrt, Length, Strength,
                              kFQ)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`. The quadrupole `Length` is required together with either the field gradient, `Strength` (equation 28), or the quadrupole k parameter, `kFQ` (equation 30).

A.4.5.2 Instance attributes and access methods

The instance attributes are defined in table 35. The attributes are accessed and set using the methods defined in table 37.

Table 35: Definition of attributes of instances of the `FocusQuadrupole(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`FQmode`	int		If 0, use particle momentum in calculation of transfer matrix; if 1, use reference particle momentum.
`Length`	float	m	Effective length of quadrupole.
`Strength`	float	T/m	Magnetic field gradient; required if kFQ is not given.
`kFQ`	float	m^{-2}	Quadrupole k parameter.

Table 36: Definition of access methods for the `FocusQuadrupole` derived class.

Set method	Get method	Comment
`setFQmode(FQmode)`	`getFQmode()`	Set/get FQmode.
`setLength(Length)`	`getLength()`	Set/get length.
`setStrength(Length)`	`getStrength()`	Set/get strength (field gradient).
`setKFQ(Length)`	`getKFQ()`	Set/get kFQ, quadrupole k parameter.

A.4.5.3 Processing methods

The `FocusQuadrupole` processing methods are defined in table 37.

A.4.5.4 I/o methods

The `Focusquadrupole` derived class has no i/o methods other than those inheritted from the parent class.

Table 37: Utilities provided by the `FocusQuadrupole` derived class.

Method	Argument(s)	Return	Comment
`calckFQ()`		float	Calculates `kFQ` if strength is given in instance attributes.
`calcStrength()`		float	Calculates `Strength` if `kFQ` is is given in instance attributes.

A.4.5.5 Utilities

The `Focusquadrupole` derived class has no utilities other than those inheritted from the parent class.

A.4.6 Derived class: `DefocusQuadrupole(BeamLineElement)`

A.4.6.1 Instantiation

The call to instantiate the `DefocusQuadrupole` derived class is:

```
DefocusQuadrupole(Name, rStrt, vStrt, drStrt, dvStrt, Length, Strength,
                                    kDQ)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`. The quadrupole `Length` is required together with either the field gradient, `Strength` (equation 28), or the quadrupole k parameter, `kDQ` (equation 30).

A.4.6.2 Instance attributes and access methods

The instance attributes are defined in table 38. The attributes are accessed and set using the methods defined in table 40.

Table 38: Definition of attributes of instances of the `DefocusQuadrupole(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`DQmode`	int		If 0, use particle momentum in calculation of transfer matrix; if 1, use reference particle momentum.
`Length`	float	m	Effective length of quadrupole.
`Strength`	float	T/m	Magnetic field gradient; required if kDQ is not given.
`kDQ`	float	m^{-2}	Quadrupole k parameter.

A.4.6.3 Processing methods

The `DefocusQuadrupole` processing methods are defined in table 40.

A.4.6.4 I/o methods

The `Defocusquadrupole` derived class has no i/o methods other than those inheritted from the parent class.

A.4.6.5 Utilities

The `Defocusquadrupole` derived class has no utilities other than those inheritted from the parent class.

Table 39: Definition of access methods for the `DefocusQuadrupole` derived class.

Set method	Get method	Comment
`setDQmode(DQmode)`	`getDQmode()`	Set/get DQmode.
`setLength(Length)`	`getLength()`	Set/get length.
`setStrength(Length)`	`getStrength()`	Set/get strength (field gradient).
`setKDQ(Length)`	`getKDQ()`	Set/get kDQ, quadrupole k parameter.

Table 40: Utilities provided by the `DefocusQuadrupole` derived class.

Method	Argument(s)	Return	Comment
`calckDQ()`		float	Calculates `kDQ` if strength is specified.
`calcStrength()`		float	Calculates `Strength` if `kDQ` is specified.

A.4.7 Derived class: `SectorDipole(BeamLineElement)`

A.4.7.1 Instantiation

The call to instantiate the `SectorDipole` derived class is:

`SectorDipole(Name, rStrt, vStrt, drStrt, dvStrt, Angle, B)`

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`.

The orientation of the RLPC coordinate axes with respect to those of the laboratory frame changes from the start of sector dipole to its end. Referring to figure 2, the vector, $\boldsymbol{v}_{\rm ES}$, that translates the origin of the RLPC coordinate system at the start of the sector dipole to the origin of the RLPC coordinate system at its end is given by:

$$\boldsymbol{v}_{\rm ES} = 2\rho_0 \sin\left(\frac{\phi}{2}\right) \begin{pmatrix} \sin\left(\frac{\phi}{2}\right) \\ 0 \\ \cos\left(\frac{\phi}{2}\right) \end{pmatrix} ; \tag{57}$$

where ρ_0 is the radius of the circular locus of the trajectory of the reference particle. If the rotation matrix taking the RPLC axes at the start of the sector dipole to the laboratory coordinate axes is $\underline{\underline{R}}_{\rm S}$, then the vector, $\boldsymbol{v}_{\rm ES}^{\rm lab}$, that translates from the start of the sector dipole to its end in laboratory coordinates is given by:

$$\boldsymbol{v}_{\rm ES}^{\rm lab} = \underline{\underline{R}}_{\rm S} \boldsymbol{v}_{\rm ES} \, . \tag{58}$$

The rotation matrix that transforms from the RPLC system at the end of the sector dipole to the laboratory coordinate system, $\underline{\underline{R}}_{\rm E}$ is given by:

$$\underline{\underline{R}}_{\rm E} = \underline{\underline{R}}_{\rm S} \underline{\underline{R}} ; \tag{59}$$

where:

$$\underline{\underline{R}} = \begin{pmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{pmatrix} . \tag{60}$$

A.4.7.2 Instance attributes and access methods

The instance attributes are defined in table 41. The attributes are accessed and set using the methods defined in table 42.

Table 41: Definition of attributes of instances of the `SectorDipole(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Angle`	float	rad	Angle through which sector dipole bends positive reference particle.
`B`	float	T	Magnetic field.

Table 42: Definition of access methods for the `SectorDipole` derived class.

Set method	Get method	Comment
`setAngle(Angle)`	`getAngle()`	Set/get bending angle.
`setB(B)`	`getB()`	Set/get dipole magnetic field.
`setLength()`	`getLength()`	Set/get length of reference particle trajectory through sector dipole (arc length).

A.4.7.3 Processing methods

The `SectorDipole` derived class has no processing methods other than those inheritted from the parent class.

A.4.7.4 I/o methods

The `SectorDipole` derived class has no i/o methods other than those inheritted from the parent class.

A.4.7.5 Utilities

The `SectorDipole` derived class has no utilities other than those inheritted from the parent class.

A.4.8 Derived class: `Solenoid(BeamLineElement)`

A.4.8.1 Instantiation

The call to instantiate the `Solenoid` derived class is:

```
Solenoid(Name, rStrt, vStrt, drStrt, dvStrt, Length, Strength, kSol)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`. The solenoid `Length` is required together with either the magnetic field strength, `Strength` or the solenoid k parameter, `kSol` (equation 33).

A.4.8.2 Instance attributes and access methods

The instance attributes are defined in table 43. The attributes are accessed and set using the methods defined in table 45.

A.4.8.3 Processing methods

The `Solenoid` processing method is defined in table 45.

A.4.8.4 I/o methods

The `Solenoid` derived class has no i/o methods other than those inheritted from the parent class.

Table 43: Definition of attributes of instances of the `Solenoid(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	**Type**	**Unit**	**Comment**
`Length`	float	m	Effective length of solenoid.
`Strength`	float	T/m	Magnetic field gradient; required if kSol is not given.
`kSol`	float	m^{-2}	GaborLens k parameter required if `Strength` not given.

Table 44: Definition of access methods for the `Solenoid` derived class.

Set method	**Get method**	**Comment**
`setLength(Length)`	`getLength()`	Set/get length.
`setStrength(B)`	`getStrength()`	Set/get strength (solenoid magnetic field).
`setKSol(Length)`	`getKFQ()`	Set/get kSol, solenoid k parameter.

A.4.8.5 Utilities

The `Solenoid` derived class has no utilities other than those inheritted from the parent class.

Table 45: Utilities provided by the `Solenoid` derived class.

Method	**Argument(s)**	**Return**	**Comment**
`calckSol()`		float	Calculates `kSol` if strength is specified.
`calcStrength()`		float	Calculates `Strength` if `kSol` is specified.

A.4.9 Derived class: `GaborLens(BeamLineElement)`

A.4.9.1 Instantiation

The call to instantiate the `GaborLens` derived class is:

```
GaborLens(Name, rStrt, vStrt, drStrt, dvStrt, Bz, VA, RA, Rp, Length,
                              kSol)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`. The Gabor lens `Length` is required together with either the parameters `Bz`, `VA`, `RA`, and `Rp` corresponding, respectively, to the parameters B_z, V_A, V_A and R_p defined in section 4.4, or $kSol$, the solenoid strength parameter of the equivalent solenoid (see section 4.4). The effective electon number density inside the trap is calculated using either `Bz`, `VA`, `RA` and `Rp` or `kSol`.

A.4.9.2 Instance attributes and access methods

The instance attributes are defined in table 46. The attributes are accessed and set using the methods defined in table 47.

A.4.9.3 Processing methods

The `GaborLens` processing method is defined in table 45.

Table 46: Definition of attributes of instances of the `GaborLens(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Bz`	float	T	Effective length of Gabor lens.
`VA`	float	V	Effective length of Gabor lens.
`RA`	float	m	Effective length of Gabor lens.
`RP`	float	m	Effective length of Gabor lens.
`Length`	float	m	Effective length of Gabor lens.
`Strength`	float	T/m	Magnetic field gradient; required if kSol is not given.
`kSol`	float	m^{-2}	k parameter of the solenoid with the equivalent focusing strength.

Table 47: Definition of access methods for the `GaborLens` derived class.

Set method	Get method	Comment
`setBz(Bz)`	`getBz()`	Set/get magnetic field of the Penning-Malmberg trap.
`setVA(VA)`	`getVA()`	Set/get anode voltage of the Penning-Malmberg trap.
`setRA(RA)`	`getRA()`	Set/get radius of the anode of the Penning-Malmberg trap.
`setRP(RP)`	`getRP()`	Set/get magnetic effective radiius of the plasma confined within the Penning-Malmberg trap.
`setLength(Length)`	`getLength()`	Set/get effective length of the lens.
`setStrength(Strength)`	`getStrength()`	Set/get k-parameter of the solenoid with the equivalent focal length.
`setElectronDenisty()`	`getElectronDenisty()`	Set/get electron density.

A.4.9.4 I/o methods

The `GaborLens` derived class has no i/o methods other than those inheritted from the parent class.

A.4.9.5 Utilities

The `GaborLens` derived class has no utilities other than those inheritted from the parent class.

A.4.10 Derived class: `CylindricalRFCavity(BeamLineElement)`

A.4.10.1 Instantiation

The call to instantiate the `CylindricalRFCavity` derived class is:

```
CylindricalRFCavity(Name, rStrt, vStrt, drStrt, dvStrt, Gradient,
                    Frequency, Phase)
```

Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.4.1.2. These arguments are passed directly to `BeamLineElement`.

A.4.10.2 Instance attributes and access methods

The instance attributes are defined in table 48. The attributes are accessed and set using the methods defined in table 49.

Table 48: Definition of attributes of instances of the `CylindricalRFCavity(BeamLineElement)` derived class. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Gradient`	float	MV/m	Peak electric field gradient on axis.
`Frequency`	float	MHz	Resonant frequency.
`Phase`	float	rad	Phase cavity at time reference particle crosses centre of cavity, “linac convention”.
`TransitTimeFactor`	float		Transit time factor (equation **??**).
`V0`	float	MV	Peak voltage.
`alpha`	float		α parameter defined in equation **??**.
`wperp`	float		$\omega_\perp$ parameter defined in equation **??**.
`cperp`	float		$c_\perp$ parameter defined in equation **??**.
`sperp`	float		$s_\perp$ parameter defined in equation **??**.
`wprll`	float		$\omega_{\|\|}$ parameter defined in equation **??**.
`cprll`	float		$c_{\|\|}$ parameter defined in equation **??**.
`sprll`	float		$s_{\|\|}$ parameter defined in equation **??**.

A.4.10.3 Processing methods

The `CylindricalRFCavity` derived class has no processing methods other than those inheritted from the parent class.

A.4.10.4 I/o methods

The `CylindricalRFCavity` derived class has no i/o methods other than those inheritted from the parent class.

A.4.10.5 Utilities

The `CylindricalRFCavity` derived class has no utilities other than those inheritted from the parent class.

A.4.10.6 Processing methods

The `CylindricalRFCavity` dericed class has no processing methods.

Table 49: Definition of access methods for the CylindricalRFCavity derived class.

Set method	Get method	Comment
setGradient(Gradient)	getGradient()	Set/get peak electric field gradient.
setFrequency(Frequency)	getFrequency()	Set/get frequency.
setAngularFrequency(AngFreq)	getAngularFrequency()	Set/get angular frequency.
setPhase(Phase)	getPhase()	Set/get phase.
setWaveNumber(WaveNumber)	getWaveNumber()	Set/get wavenumber.
setLength(Length)	getLength()	Set/get Length.
setRadius(Radius)	getRadius()	Set/get Radius.
setTransitTimeFactor (TransitTimeFactor)	getTransitTimeFactor()	Set/get TransitTimeFactor.
setV0(V0)	getV0()	Set/get peak voltage.
setalpha(alpha)	getalpha()	Set/get alpha.
setwperp(wperp)	getwperp()	Set/get wperp.
setcperp(cperp)	getcperp()	Set/get cperp.
setsperp(sperp)	getsperp()	Set/get sperp.
setwprll(wprll)	getwprll()	Set/get wprll.
setcprll(cprll)	getcprll()	Set/get cprll.
setsprll(sprll)	getsprll()	Set/get sprll.
setmrf(mrf)	getmrf()	Set/get mrf.

A.4.11 Derived class: Source(BeamLineElement)

A.4.11.1 Instantiation

The call to instantiate the Source derived class is:

```
Source(Name, rStrt, vStrt, drStrt, dvStrt, Mode, Parameters)
```

Parent class arguments Name, rStrt, vStrt, drStrt, and dvStrt are described in section A.4.1.2. These arguments are passed directly to BeamLineElement. Mode is an integer that transmits the type of source to be generated. The specification of the Parameters list depends on the value of Mode. The content of the Parameters list is transferred directly to the Param instance attribute.

A.4.11.2 Instance attributes and access methods

The instance attributes are defined in tables 50, 51, and 52, each table refers to a particular source Mode. The attributes are accessed and set using the methods defined in table 54.

A.4.11.3 Processing methods

The Source derived class has no processing methods other than those inheritted from the parent class.

A.4.11.4 I/o methods

The Source derived class has no i/o methods other than those inheritted from the parent class.

Table 50: Definition of attributes of instances of the `Source(BeamLineElement)` derived class for source `Mode`$=0$, the parameterised TNSA model. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Mode`	integer		`Mode`$=0$; parameterised laser-driven source.
`Param[0]`	float	m	Standard deviation of normal distribution from which x coordinate is sampled.
`Param[1]`	float	m	Standard deviation of normal distribution from which y coordinate is sampled.
`Param[2]`	float		Minimum $\cos\theta_S$. The specification of a minimum for $\cos\theta_S$ improves the efficiency of generation as it may be used to restrice generation to the set of partices that will enter the downstream acceptane.
`Param[3]`	float	MeV	Minimum kinetic energy ($K_{\rm min}$.
`Param[4]`	float	MeV	Maximum kinetic energy ($K_{\rm max}$. Value entered here is overwritten when calculated in `getLaserDrivenParticleEnergy` during initialisation.
`Param[5]`	integer		`nPnts`: Number of points to sample for integration of PDF (kept for backward compatibility).
`Param[6]`	float	W	P_L: Laser power.
`Param[7]`	float	J	E_L: Laser energy.
`Param[8]`	float	μm	λ: Laser wavelength.
`Param[9]`	float	s	$t_{\rm laser}$: Laser pulse duration.
`Param[10]`	float	m	d: Diameter of laser spot at focus.
`Param[11]`	float	W/cm^2	I: Laser intensity
`Param[12]`	float	$^\circ$	$\theta_{\rm degrees}$: Electron divergence angle.
`Param[13]`	float	$^\circ$	Intercept of α, maximum half opening angle at $K=0$.
`Param[14]`	float	$^\circ$	Scaled slope of $\alpha(K)$.

A.4.11.5 Processing methods

595 The processing methods provided by the `Source` derived class are listed in table 55.

A.4.11.6 Utilities

The utilities provided by the `Source` derived class are listed in table 56.

Table 51: Definition of attributes of instances of the `Source(BeamLineElement)` derived class for source `Mode`$=1$ in which energy is sampled from a normal distribution. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Mode`	integer		`Mode`$=1$; parameterised laser-driven source.
`Param[0]`	float	m	Standard deviation of normal distribution from which x coordinate is sampled.
`Param[1]`	float	m	Standard deviation of normal distribution from which y coordinate is sampled.
`Param[2]`	float		Minimum $\cos\theta_S$. The specification of a minimum for $\cos\theta_S$ improves the efficiency of generation as it may be used to restrice generation to the set of partices that will enter the downstream acceptane.
`Param[3]`	float	MeV	Mean kinetic energy.
`Param[4]`	float	MeV	Standard deviation of kinetic energy distribution.

Table 52: Definition of attributes of instances of the `Source(BeamLineElement)` derived class for source `Mode`$=2$ in which energy is sampled from a uniform distribution. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Mode`	integer		`Mode`$=2$; parameterised laser-driven source.
`Param[0]`	float	m	Standard deviation of normal distribution from which x coordinate is sampled.
`Param[1]`	float	m	Standard deviation of normal distribution from which y coordinate is sampled.
`Param[2]`	float		Minimum $\cos\theta_S$. The specification of a minimum for $\cos\theta_S$ improves the efficiency of generation as it may be used to restrice generation to the set of partices that will enter the downstream acceptane.
`Param[3]`	float	MeV	Mean kinetic energy.
`Param[4]`	float	MeV	Maximum kinetic energy.

Table 53: Definition of attributes of instances of the `Source(BeamLineElement)` derived class for source `Mode`$=3$ in which parameters of particle at source are read from an input file. All attributes are required in the call to instantiate the element.

Attribute	Type	Unit	Comment
`Mode`	integer		`Mode`$=3$; partice trace space read from file. In this case no additional paramters are required.

Table 54: Definition of access methods for the `Source` derived class.

Set method	Get method	Comment
`setMode`	`getMode()`	Set/get Mode.
`setModeText`	`getModeText()`	Set string with readable name for mode.
`setModeParamterText`	`getParameterText()`	Set/get list of strings with readable name for paramter.
`setParameters`	`getParameters()`	Set/get parameters; list of paramters as defined above.
`setParameterUnit`	`getParameterUnit()`	Set/get list of strings containing parameter units.

Table 55: Definition of processing methods provided by the `Source` derived class.

Method	Arguments	Return	Comment
`getParticleFromSource()`		ndarray	Get trace space for particle at source, returns np.array.
`getParticle()`		floats	Called from `getParticleFromSource()`, returns parameters used to create trace space of particle at source.
`getFlatThetaPhi()`		floats	Called from `getParticle()` if flat $\cos\theta_S$, flat ϕ_S distribution is requested.
`getgofrp(rpmax, xp, yp)`	floats	float	Returns $g(r')$ given input $r'_{\max}$, x', and y'.
`g_theta(Energy)`	float		Returns θ_S generated using a guassian distribution. Depricated.
`angle_gemerator(Energy)`	float		Returns θ_S, ϕ_S using `g_theta` and uniform distribution for ϕ_S. Depricated.
`getGaussianThetaPhi(Energy)`	float		Returns $\cos\theta_S$ and ϕ_S using `angle_generator`.
`paramters(P, E, l, tl, d, I, t)`	floats		Returns derived parameters used to generate TNSA proton kinetic energy spectrum.
`equation(x, tl, t0)`	floats		Returns result of evaluation equation **??**.
`getLaserDrivenProtonEnergy()`			Generates proton kinetic energy at source for paramterised TNSA model.
`getLaserCumProbParam()`			
`getLaserCumProb()`			
`getLaserDrivenProtonEnergyProbDensity()`			
`getTraceSpace()`			

Table 56: Definition of utilities provided by the `Source` derived class.

Method	Arguments	Return	Comment
`CheckSourceParam(Mode, Param)`	Integer, list	boolean	Class method. Check that mode and parameters are valid. Calls `CheckMode` and `CheckParam`
`CheckMode(Mode)`	integer	boolean	Class method. Check is valid.
`CheckParam`	list	boolean	Class method. Check paramter list is valid.
`tabulateParameters()`			

A.5 UserFramework

UserFramework is a module that provides a set of methods used in the code provided to allow users easy access to the code and data. The following methods are provided:

startAnalysis(argv): processes input flags passed in call to run script.
  Argument:
   argv: list of arguments passed to main by call to run script.
  Returns:
   Success: (boolean) True if processing successful.
   Debug: (boolean) Set to True if flag -d is set to True.
   inputfile: (path) full path to input file; set using flag -y.
   outputfile: (path) full path to output file; set using flag -o.
   nEvts: (integer) number of events to process; set using flag -n.
   bdsimfile: (path) full path to bdsim format file; set using flag -z.
   beamspecfile: (path) full path to beam specification CSV file; set using flag -b.

handleFILES(beamspecfile, inputfile, outputfile, bdsimFILE=False): File handling method to check files exist and create relevant BeamIO instances.
  Arguments:
   beamspecfile: (path) full path to beam specification CSV file;
   inputfile: (path) full path to input file;
   outputfile: (path) full path to output file;
   bdsimfile: (path) full path to bdsim format file.
  Returns:
   Success: (boolean) True if file handling successful.
   ibmIOr: (BeamIO) instance of BeamIO class for file to be read.
   ibmIOw: (BeamIO) instance of BeamIO class for file to be written.

EventLoop(iUsrAnl, ibmIOr, ibmIOw, nEvtsIn): executes loop over nEvtsIn events. Reads event record or generates event if ibmIOr=None, and handles end-of-file. Passes control to UserAnal.EventLoo
  Arguments:
   iUsrAnl: (UserAnal) instance.
   ibmIOr: (BeamIO) instance of BeamIO class for file to be read.
   ibmIOw: (BeamIO) instance of BeamIO class for file to be written.
   nEvtsIn: (integer) number of events to process.
  Returns:
   Success: (boolean) True if successful.

A.6 visualise

The visualise class manages the visualisation of the beam line and particles traversing it.

A.6.1 Instantiation

The call to instantiate the `visualise` class is:

`Beam(CoordSys, Projection)`

`CoordSys` (string) is either “RPLC” to visualise the beam line in the reference particle local coordinate system or “Lab”. `Projection` (string) is either “xs” (RPLC) or “xz” (lab) to visualise the xs or xz projecton and eithed “ys” (RPLC) or “yz” (lab) to visualise the xs or xz projecton.

A.6.2 Instance attributes and access methods

The instance attributes are presented in table 57 and the access methods are summarised in table 58.

Table 57: Definition of attributes of instances of the `visualise` class.

Attribute	Type	Comment
`CoordSys`	String	Either “RPLC” to visualise the in the reference particle local coordinate system or “Lab”.
`Projection`	String	Either “xs” (RPLC) or “xz” (lab) to visualise the xs or xz projecton and eithed “ys” (RPLC) or “yz” (lab) to visualise the xs or xz projecton.

Table 58: Definition of access methods for the `visualise` class.

Set method	Get method	Comment
`setCoordSys`	`getCoordSys`	Set coordinate system for visualisation.
`setProjection`	`setProjection`	Set projection for visualisation.

A.6.3 Processing methods

Table 59 presents the processing methods provided in the `visualise` class.

Table 59: Processing methods provided by the `visualise` class.

Method	Argument(s)	Return	Comment
`Particles(axs, nPrtcl)`	plot, integer		Manage ploting of `nPrtcl` particles on `matplotlib` `axes` instance.
`BeamLine(axs)`	plot		Manage ploting of beam line on `matplotlib` `axes` instance.

A.6.4 I/o methods

The `visualise` class has no i/o methods.

A.6.5 Utilities

The `visualise` class has no i/o methods.

A.7 `BeamIO`

The `BeamIO` class provides interfaces to the reading and writing of beam specification and beam data files.

A.7.1 Instantiation

The call to instantiate the `BeamIO` class is:

```
BeamIO(datafilePATH, datafileNAME, create, BDSIMfile)
```

`datafilePATH`: (string) Path to directory in which input file is to be found, or, in which output file is to be created. `datafilePATH` can be set to None if the full path is specified in `datafileNAME`.

`datafileNAME`: (string) File name (string) which, when appendded to `datafilePATH` gives the full path to the input or output data file, or. full path to the input or output file.

`create`: (boolean) if true indicates that the file must be created. If a file exists at the location specified by `datafilePATH` and `datafileNAME` it will be overwritten.

`BDSIMfile`: (boolean) If True then the file is to be read or written in `BDSIMfile` format.

A.7.2 Instance attributes and access methods

The `BeamIO` instance attributes are presented in table 60 and the access methods are summarised in table 61.

Table 60: Definition of attributes of instances of the `BeamIO` class.

Attribute	Type	Comment
`dataFILE`	Path	Full path to data file.
`Read1stRecord`	Boolean	True if first record has been read from file.
`dataFILEversion`	Integer	For `BeamIO` files version number identifying file format.
`create`	Boolean	True if data file is to be created.
`BDSIMfile`	Boolean	True id reading or writing a BDSim file.

A.7.3 Processing methods

The `BeamIO` class has no processing methods.

A.7.4 I/o methods

Table 62 presents the i/o methods provided by the `BeamIO` class.

A.7.5 Utilities

Table 63 presents the utilities provided by the `BeamIO` class.

Table 61: Definition of access methods for the `BeamIO` class.

Set method	Get method	Comment
`setpathFILE`	`getpathFILE`	Set/get path to directory containing data file.
`setdataFILE`	`getdataFILE`	Set/get full path to data file.
`setReadFirstRecord`	`getReadFirstRecord`	Set/get flag indicating whether the first record has been read.
`setcreate`	`getcreate`	Set/get flag indicating whether data file is to be created.
`setdataFILEversion`	`getdataFILEversion`	Set/get `BeamIO` version number.
`setBDSIMfile`	`getBDSIMfile`	Set/get flag indicating whether the data file is in BDSIM format.

Table 62: I/o methods provided by the `BeamIO` class.

Method	Argument(s)	Return	Comment
`readBeamDataRecord()`		Boolean	Manages reading/writing of a record from/to the data file. Returns a boolean, `EoF`, set to True of end of file has been detected.
`readVersion()`		Integer	Reads data-file format version number from file. Returns integer version number.
`flushNclosedataFile(dataFILE)`	Path		Flush and close data `dataFILE` at end of processing.

Table 63: Utilities provided by the `BeamIO` class.

Method	Argument(s)	Return	Comment
`resetinstances()`			Class method. Sets list of instances to `[]`.
`cleanBeamIOfiles()`			Class method. Delete `BeamIO` instances and reset list of instances.

A.8 Simulation

The singleton Simulation class provides a framework and utilities for the simulation of the passage of particles through the beam lines defined through the classes described in this document. By default the seed for the random number generator is set using the system time.

A.8.1 Instantiation

The call to instantiate the Simulation class is:

```
Simulation(NEvts, BeamLineSpecificationCVSfile, dataFileDir,
                          dataFileName)
```

NEvts: (integer) Number of events to generate.

BeamLineSpecificationCVSfile: (string) String containing path to the beam-line specification CVS file.

dataFileDir: (string) String containing path to directory in which data file is to be written.

dataFileName: (string) Name of file to be written.

Simulation has two methods defined outside the class:

getRandom(): returns number between 0. and 1. drawn from a uniform distributin; and

getParabolic(umax): returns number between −umax and umax drawn from a parabolic distribution with a maximum at 0.

A.8.2 Instance attributes and access methods

The Simulation instance attributes are presented in table 64 and the access methods are summarised in table 65.

Table 64: Definition of attributes of instances of the Simulation class.

Attribute	Type	Comment
NEvt	integer	Number of events to generate.
ParamFileName	string	Path to beam-line parameter CSV file.
dataFileDir	string	Path to the directory in which data file is to be written.
dataFileName	string	Name of file to be created.
Facility	Facility	Instance of the derived Facility class derived from BeamLineElement.
iBmIOw	BeamIO	Instance of the derived BeamIO class for the file to be written.
ProgressPrint	boolean	If True the progress towards the NEvt requested events is printed.

A.8.3 Processing methods

The Simulation class provides one processing method:

```
RunSim()
```

which manages the generation of NEvts events. The specification of the beam line and the individual events are written to the output data file.

A.8.4 I/o methods

The `Simulation` class provides no i/o methods.

A.8.5 Utilities

The `Simulation` provides no utilities.

Table 65: Definition of access methods for the `Simulation` class.

Set method	Get method	Comment
`setNEvt`	`getNEvt`	Set/get number of events to generate.
`setBeamLineSpecificationFile`	`getBeamLineSpecificationFile`	Set/get full path to beam line specification file.
`setdataFileDir`	`getdataFileDir`	Set/get path to directory in which data file is to be written.
`setdataFileName`	`getdataFileName`	Set/get name of data file to be created.
`setFacility`	`getFacility`	Set/get `Facility` instance.
`setiBmIOw`	`getiBmIOw`	Set/get `BeamIO` instance specifying the file to be written.
`setProgressPrint`	`getProgressPrint`	Set/get flag that controls the progress printout.

A.9 Physical constants

The PhysicalConstants class provides the physical constants that are required to carry out the linear optics calculations. The values are taken from, for example, the Particle Data Group book. The constants packages is implemented as a singleton class so that there is no ambiguity about which values are in use.

A.9.1 Instantiation

The call to instantiate the PhysicalConstants class is:

```
PhysicalConstants()
```

A.9.2 Instance attributes and access methods

The PhysicalConstants has no instance attributes. The access methods are summarised in table 66.

Table 66: Definition of access methods for the PhysicalConstants class.

Set method	Get method	Comment
	getPDGref	Returns PDG reference used.
	getSoL	Returns speed of light in m/s.
	getSpecies	Returns list of particle species for which constants are stored.
	getParticleMASS(Species)	Returns particle mass for Species. Species is a list of strings: ["proton", "pion", "muon", "neutrino"].
	mp	Returns proton mass in MeV.
	getmPion	Returns pion mass in MeV.
	getmMuon	Returns muon mass in MeV.
	getmNeutrino	Returns 0, neutrino mass (for nuSIM).
	getm0	Returns permittivity of free space.

A.9.3 Processing methods

The PhysicalConstants class provides no processing methods.

A.9.4 I/o methods

The PhysicalConstants class provides no i/o methods.

A.9.5 Utilities

The PhysicalConstants provides no utilities.

A.10 Report

The Report parent class supports a collection of derived classes to generate reports, usually in the form of a CSV file, based on the data stored in the attributes of the Beam, BeamLine, BeamLineElement, Particle, and other classes.

A.10.1 Instantiation

The call to instantiate the `Report` class is:

```
Report(Name, ReportPath, FileName, Header, Lines)
```

`Name`: (string) Name of report;

`ReportPath`: (string) String containing path to the directory where report will be written;

`FileName`: (string) String containing name of the file in which report will be written.

`Header`: (list of strings) List containing strings that will form the header record of the report; and

`Lines`: (list of lists) The lines that will make up the lines of the report. List[i][j] provides the value of the item to be recorded in column `j` of row `i` of the report.

A.10.2 Instance attributes and access methods

The `Report` parent class supports a collection of derived classes to generate reports, usually in the form of a CSV file, based on the data stored in the attributes of the `Beam`, `BeamLine`, `BeamLineElement`, `Particle`, and other classes. instance attributes are presented in table 67 and the access methods are summarised in table 68.

Table 67: Definition of attributes of instances of the `Report` class.

Attribute	Type	Comment
`NEvt`	integer	Number of events to generate.
`ParamFileName`	string	Path to beam-line parameter CSV file.
`dataFileDir`	string	Path to the directory in which data file is to be written.
`dataFileName`	string	Name of file to be created.
`Facility`	Facility	Instance of the derived `Facility` class derived from `BeamLineElement`.
`iBmIOw`	BeamIO	Instance of the derived `BeamIO` class for the file to be written.
`ProgressPrint`	boolean	If True the progress towards the `NEvt` requested events is printed.

Table 68: Definition of access methods for the `Report` class.

Set method	Get method	Comment
`setName`	`getName`	Set/get name of report.
`setReportPath`	`getReportPath`	Set/get path to director into which report file is to be written.
`setFileName`	`getFileName`	Set/get name of file to be written.
`setHeader`	`getHeader`	Set/get list of strings forming header fields.
`setLines`	`getLines`	Set/get list of lists containing report entries line by line.

A.10.3 Processing methods

Table 70 presents the processing methods provided in the `Report` class.

Table 69: Processing methods provided by the `Report` class.

Method	Argument(s)	Return	Comment
`createPandasDataFrame()`		Data frame instance	Create pandas data frame using the data contained in `Header` and `Lines`. The instance is returned.

A.10.4 I/o methods

Table **??** presents the i/o methods provided in the `Report` class.

Table 70: I/o methods provided by the `Report` class.

Method	Argument(s)	Return	Comment
`createCSV(DataFrame)`	Data frame instance		Create CSV file from pandas data `DatsFrame` frame using the data contained in `Header` and `Lines`.
`asCSV()`			Write pandas dataframe and write it to CSV file.

A.11 `LaTeX`

The `LaTeX` module provides 2 methods to support the generation of LaTeX tables from data stored in the class and instance attributes. The first method, `TableHeader`, creates the header of LaTeX table. The method is accessed via the call:

```
TableHeader(FilePath, TabString, Caption)
```

where:

`FilePath`: (path) is the full path to the file to contain the table;

`TabString`: (string) is the string that defines the columns, e.g., `'|c|c|'`, would result in a two-column table in which the contents of each column is centred; and

`Caption`: (string) Is the string to be used as the table caption; and

Each line in the table are entered with the call:

```
TableLine(FilePath, Line)
```

where:

`FilePath`: (path) is the full path to the file to contain the table;

`Line`: (string) is a list of strings that contain the contents of the line. A typical use case would be to create the table header, enter a line containing the header fields and then procede to add each line in the table in turn; and LaTeX commants are allowed, e.g. `Line = \hline` will produce a horizontal line.

The final lines of LaTeX code is entered with the call:

```
TableTrailer(FilePath)
```

where:

`FilePath`: (path) is the full path to the file to contain the table.

B Set-up and run

Introduction

This section summarises the steps needed to set-up and run the linear optics simulation of the LhARA beam line. A summary of the tasks that the software suite performs will be documented in due course. The code has been developed in python; python 3 is assumed.

Getting the code

The linear optics package is maintained using the GitHub version-control system. The latest release can be downloaded from:

```
\centerline{
  \href{https://github.com/ImperialCollegeLondon/LhARAlinearOptics.git}{https://c
}
```

Dependencies and required packages

The linear optics code requires the following packages:

- Python modules: `scipy` and `matplotlib`.

It may be convenient to run the package in a "virtual environment". To set this up, after updating your python installation to python 3.9, execute the following commands:

1. `python3 -m venv --system-site-packages venv`
   - This creates the director `venv` that contains files related to the virtual environment.
2. `source venv/bin/activate`
3. `python -m pip install pandas scipy matplotlib`

To exit from the virtual environment, execute the command `deactivate`. The command `source venv/bin/activat` places you back into the virtual environment.

The Imperial HEP linux cluster provides python 3.9.18 by default.

Unpacking the code, directories, and running the tests

After downloading the package from GitHub, or cloning the repository, you will find a "`README.md`" file which provides some orientation and instructions to run the code. In particular, a `bash` script "`startup.bash`" is provided which:

- Sets the "`LhARAOpticsPATH`" environment variable so that the files that hold constants etc. required by the code can be located; and
- Adds "`01-Code`" (see below) to the PYTHONPATH. The scripts in "`02-Tests`" (see below) may then be run with the command "`python 02-Tests/<filename>.py`".

Below the top directory, the directory structure in which the code is presented is:

`01-Code`: contains the python implementation as a series of modules. Each module contains a single class or a related set of methods.

`02-Tests`: contains self-contained test scripts that run the various methods and simulation packages defined in the code directory.

`11-Parameters`: contains the parameter set used to specifiy the various beam lines presently implemented.

The instructions in the `README.md` file should be followed to set up and run the code.

Running the code

Execute ”`startup.bash`” from the top directory (i.e. run the bash command ”`source startup.bash`”). This will:

- Set up ”`LhARAOpticsPATH`”; and
- Add ”`01-Code`” to the `PYTHONPATH`. The scripts in ”`02-Tests`” may then be run with the command ”`python 02-Tests/<filename>.py`”;
- Example scripts are provided in ”`03-Scripts`”, these can be used first to “`Run`” the simulation and then to “`Read`” the data file produced. Example scripts are provided for the DRACO, LION, and LhARA Stage 1 beam lines.