

# LhARA linear optics documentation

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

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Coordinate systems</b>	<b>1</b>
	2.1 Laboratory coordinate system . . . . .	1
5	2.2 Reference particle local coordinate system . . . . .	2
	2.3 Transforming to and from reference particle local coordinates to laboratory coordinates . . . .	3
<b>3</b>	<b>Phase space and trace space</b>	<b>3</b>
	3.1 Phase space . . . . .	4
	3.2 Trace space . . . . .	4
10	<b>4 Transfer matrices</b>	<b>5</b>
	4.1 Drift . . . . .	5
	4.2 Quadrupole . . . . .	5
	4.3 Solenoid . . . . .	6
	4.4 Non-neutral (electron) plasma (Gabor) lens . . . . .	7
15	4.5 Dipole . . . . .	8
<b>5</b>	<b>Source</b>	<b>9</b>
	5.1 Energy distribution . . . . .	9
	5.2 Angular Distribution . . . . .	11
	5.3 Spatial distribution . . . . .	11
20	5.4 Simulated distributions . . . . .	11
<b>A</b>	<b>Module, class and data structures</b>	<b>14</b>
	A.1 BeamLineElement . . . . .	14
	A.2 BeamLine . . . . .	31
	A.3 Particle and reference particle . . . . .	34
25	<b>B Set-up and run</b>	<b>36</b>

## 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  
 30 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

35 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  $\mathbf{i}$ ,  $\mathbf{j}$  and  $\mathbf{k}$  respectively. The position of the reference particle as  
40 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,  $\mathbf{r}_0$ , momentum,  $\mathbf{p}_0$ , and energy,  $E_0$ , of the reference particle position,  $s$ :

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

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

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

45 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  $p_0 = |\mathbf{p}_0|$ .

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

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  
60 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.



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 particle local coordinate (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  $\mathbf{i}_r$ ,  $\mathbf{j}_r$ , and  $\mathbf{k}_r$ .

## 2.3 Transforming to and from reference particle local coordinates to laboratory co-ordinates

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

$$\mathbf{R}_0(s) = \mathbf{0}. \quad (3)$$

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

$$\mathbf{r}(s) = \mathbf{r}_0(s) + \delta\mathbf{r}(s); \quad (4)$$

where:

$$\delta\mathbf{r}(s) = \underline{\underline{R}}(s)\mathbf{R}(s); \text{ and} \quad (5)$$

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

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

$$\begin{aligned} \mathbf{i}_r &= \begin{pmatrix} i_{rx} \\ i_{ry} \\ i_{rz} \end{pmatrix}; \\ \mathbf{j}_r &= \begin{pmatrix} j_{rx} \\ j_{ry} \\ j_{rz} \end{pmatrix}; \text{ and} \\ \mathbf{k}_r &= \begin{pmatrix} k_{rx} \\ k_{ry} \\ k_{rz} \end{pmatrix}. \end{aligned} \quad (6)$$

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

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

### 3 Phase space and trace space

75 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.

80 Relativistic mechanics exploits four-vector position,  $\underline{\mathcal{R}} = (\mathbf{r}, ct)$ , and four-vector momentum,  $\underline{\mathcal{P}} = (c\mathbf{p}, E)$ . 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  $s$ .

85 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.

#### 95 3.1 Phase space

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

$$\begin{bmatrix} \mathbf{r} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \end{bmatrix} \quad (8)$$

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  $\mathbf{r}_{\text{RPLC}} = (x_{\text{RPLC}}, y_{\text{RPLC}}, z_{\text{RPLC}})$  and momentum  $\mathbf{p}_{\text{RPLC}} = (p_{x\text{RPLC}}, p_{y\text{RPLC}}, p_{z\text{RPLC}})$ . 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:

$$\underline{\phi} = \begin{pmatrix} x_{\text{RPLC}} \\ x'_{\text{RPLC}} \\ y_{\text{RPLC}} \\ y'_{\text{RPLC}} \\ z_{\text{RPLC}} \\ \delta_{\text{RPLC}} \end{pmatrix}; \quad (9)$$

where:

$$x'_{\text{RPLC}} = \frac{\partial x}{\partial s} = \frac{cp_{x \text{ RPLC}}}{cp_0}; \quad (10)$$

$$y'_{\text{RPLC}} = \frac{\partial y}{\partial s} = \frac{cp_{y \text{ RPLC}}}{cp_0}; \quad (11)$$

$$z_{\text{RPLC}} = \frac{s}{\beta_0} - ct = \frac{\Delta s}{\beta_0}; \text{ and} \quad (12)$$

$$\delta_{\text{RPLC}} = \frac{E}{cp_0} - \frac{1}{\beta_0} = \frac{\Delta E}{cp_0}. \quad (13)$$

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.

## 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:

$$\underline{\phi}_{\text{end}} = \underline{T} \underline{\phi}_{\text{start}}; \quad (14)$$

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,  $\delta s$ , to the  $s$ -coordinate given by:

$$s_{\text{end}} = s_{\text{start}} + \delta s; \quad (15)$$

where  $s_{\text{start}}$  and  $s_{\text{end}}$  are the coordinate along the reference particle trajectory at the start and end of the beam-line element respectively. There are many excellent descriptions of the derivation of the transfer matrices,  $\underline{T}$ , 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:

$$\underline{T}_{\text{drift}} = \begin{pmatrix} 1 & l & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & l & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l}{\beta_0^2 \gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}; \quad (16)$$

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

$$\delta s = l. \quad (17)$$

## 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:

$$g_x = \frac{\partial B_{qx}}{\partial x} = g_y = \frac{\partial B_{qy}}{\partial y} = g; \quad (18)$$

where the field in the quadrupole,  $\mathbf{B}_q$ , 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:

$$\underline{T}_{\text{Fquad}} = \begin{pmatrix} \cos(\sqrt{k_q}l_q) & \frac{\sin(\sqrt{k_q}l_q)}{\sqrt{k_q}} & 0 & 0 & 0 & 0 \\ -\sqrt{k_q}\sin(\sqrt{k_q}l_q) & \cos(\sqrt{k_q}l_q) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cosh(\sqrt{k_q}l_q) & \frac{\sinh(\sqrt{k_q}l_q)}{\sqrt{k_q}} & 0 & 0 \\ 0 & 0 & \sqrt{k_q}\sinh(\sqrt{k_q}l_q) & \cosh(\sqrt{k_q}l_q) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l_q}{\beta_0^2\gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}; \quad (19)$$

130 where:

$$k_q = \frac{gc}{p} \times 10^{-3} \text{ m}^{-2}, \quad (20)$$

$c$  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,  $\beta_0$  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:

$$\delta s = l_q. \quad (21)$$

135 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 ( $k_{0q} = \frac{gc}{p_0} \times 10^{-3} \text{ m}^{-2}$ ) and to include terms in the expressions for  $x$ ,  $x'$ ,  $y$ , and  $y'$  dependent on  $\delta$ . The second is to use equation 20 to calculate the effective quadrupole focusing strength, with  $k_q$  evaluated using  $p$ . The second approach has been adopted here.

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

$$\underline{T}_{\text{Dquad}} = \begin{pmatrix} \cosh(\sqrt{k_q}l_q) & \frac{\sinh(\sqrt{k_q}l_q)}{\sqrt{k_q}} & 0 & 0 & 0 & 0 \\ \sqrt{k_q}\sinh(\sqrt{k_q}l_q) & \cosh(\sqrt{k_q}l_q) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(\sqrt{k_q}l_q) & \frac{\sin(\sqrt{k_q}l_q)}{\sqrt{k_q}} & 0 & 0 \\ 0 & 0 & -\sqrt{k_q}\sin(\sqrt{k_q}l_q) & \cos(\sqrt{k_q}l_q) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l_q}{\beta_0^2\gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (22)$$

### 4.3 Solenoid

The trajectory of a beam particle through a solenoid is determined by the magnetic field strength,  $B_s$ , 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  $\mathbf{B}_s = (0, 0, B_{s0})$ , where the solenoid axis lies along the  $z_{\text{RPLC}}$  axis. The solenoid field-strength parameter is then given by:

$$k_s = \left[ \frac{B_{s0}c}{2p} \times 10^{-3} \right]^2 \text{ m}^{-2}; \quad (23)$$

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  $z_{\text{RPLC}}$  direction may be written:

$$\underline{T}_{\text{Sol}} = \begin{pmatrix} \cos^2(\sqrt{k_s}l_s) & \frac{1}{2\sqrt{k_s}} \sin(\sqrt{k_s}l_s) & \frac{1}{2} \sin(2\sqrt{k_s}l_s) & \frac{1}{\sqrt{k_s}} \sin^2(\sqrt{k_s}l_s) & 0 & 0 \\ -\frac{\sqrt{k_s}}{2} \sin(2\sqrt{k_s}l_s) & \cos^2(\sqrt{k_s}l_s) & -\sqrt{k_s} \sin^2(\sqrt{k_s}l_s) & \frac{1}{2} \sin(2\sqrt{k_s}l_s) & 0 & 0 \\ -\frac{1}{2} \sin(2\sqrt{k_s}l_s) & -\frac{1}{\sqrt{k_s}} \sin^2(\sqrt{k_s}l_s) & \cos^2(\sqrt{k_s}l_s) & \frac{1}{2\sqrt{k_s}} \sin(2\sqrt{k_s}l_s) & 0 & 0 \\ \sqrt{k_s} \sin^2(\sqrt{k_s}l_s) & -\frac{1}{2} \sin(2\sqrt{k_s}l_s) & -\frac{\sqrt{k_s}}{2} \sin(2\sqrt{k_s}l_s) & \cos^2(\sqrt{k_s}l_s) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l}{\beta_0^2 \gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (24)$$

As in the case of the quadrupoles, dispersion is accounted for by using  $p$  to calculate  $k_s$  (equation 23). The increment in the reference particle position is:

$$\delta s = l_s. \quad (25)$$

### 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:

$$k_G = \frac{e}{2\epsilon_0} \frac{m_p \gamma}{p^2} n_e \text{ m}^{-2}; \quad (26)$$

where  $e$  is the charge on the electron,  $\epsilon_0$  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 26. 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



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  $x_{\text{RPLC}}$  and  $z_{\text{RPLC}}$  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  $\rho$  and the angle through which the  $x_{\text{RPLC}}$  is rotated is  $\phi$ .

through the lens may be written:

$$\underline{T}_G = \begin{pmatrix} \cos(\sqrt{k_G} l_G) & \frac{\sin(\sqrt{k_G} l_G)}{\sqrt{k_G}} & 0 & 0 & 0 & 0 \\ -\sqrt{k_G} \sin(\sqrt{k_G} l_G) & \cos(\sqrt{k_G} l_G) & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(\sqrt{k_G} l_G) & \frac{\sin(\sqrt{k_G} l_G)}{\sqrt{k_G}} & 0 & 0 \\ 0 & 0 & -\sqrt{k_G} \sin(\sqrt{k_G} l_G) & \cos(\sqrt{k_G} l_G) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \frac{l}{\beta_0^2 \gamma_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (27)$$

170 The increment in the reference particle position is:

$$\delta s = l_G. \quad (28)$$

## 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  $\mathbf{j}_{\text{RPLC}}$ , i.e.  $\mathbf{B}_D = (0, B_{D0}, 0)$ . No edge focusing is considered.

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

$$k_D = \left[ \frac{B_{D0} c}{p} \times 10^{-3} \right]^2 \text{ m}^{-2}. \quad (29)$$



The momentum of the reference particle is related to the curvature.  $\rho$ , by:

$$p_0 = B_{D0}\rho; \quad (30)$$

so:

$$k_D = \frac{1}{\rho}; \quad (31)$$

180 and the angle  $\phi$  is given by:

$$\phi = \frac{l_D}{\rho}. \quad (32)$$

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

$$\underline{T}_D = \begin{pmatrix} \cos(\phi) & \rho \sin(\phi) & 0 & 0 & 0 & \frac{\rho}{\beta_0} (1 - \cos(\phi)) \\ -\frac{\sin(\phi)}{\rho} & \cos(\phi) & 0 & 0 & 0 & \frac{\sin(\phi)}{\beta_0} \\ 0 & 0 & 1 & l & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -\frac{\sin(\phi)}{\beta_0} & -\frac{\rho}{\beta_0} (1 - \cos(\phi)) & 0 & 0 & 1 & \frac{l}{\beta^2 \gamma^2} - \frac{l - \rho \sin(\phi)}{\beta_0^2} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (33)$$

The increment in the reference particle position is:

$$\delta s = l_D. \quad (34)$$

## 5 Source

185 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.

### 5.1 Energy distribution

190 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  $\varepsilon_{\max}$ . The kinetic-energy spectrum of the TNSA model presented in [5] is given by:

$$\frac{dN}{d\varepsilon} = \frac{n_{e0} c_s t_{laser} S_{sheath}}{\sqrt{2\varepsilon T_e}} \exp\left(-\sqrt{\frac{2\varepsilon}{T_e}}\right); \quad (35)$$

where  $N$  is the number of protons or ions produced per unit solid angle,  $\varepsilon$  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.

195 Equation 35 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:

$$\varepsilon_{max} = X^2 \varepsilon_{i,\infty}; \quad (36)$$

where  $X$  is obtained by solving:

$$\frac{t_{laser}}{t_0} = X \left(1 + \frac{1}{2} \frac{1}{1 - X^2}\right) + \frac{1}{4} \ln \left(\frac{1 + X}{1 - X}\right). \quad (37)$$

Table 1: Parameters present in the analytical expression, equation 35, describing target normal sheath acceleration (TNSA).

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

200 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  $\varepsilon \rightarrow \varepsilon + \delta\varepsilon$  is given by:

$$\delta\mathcal{P} = g(\varepsilon) \delta\varepsilon. \quad (38)$$

$g(\varepsilon)$  can be written in terms of the differential spectrum given in equation 35 through the introduction of a normalisation constant  $\mathcal{N}$ :

$$g(\varepsilon) = \frac{1}{\mathcal{N}} \frac{dN}{d\varepsilon}. \quad (39)$$

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

$$G(\varepsilon) = \int_{\varepsilon_{\min}}^{\varepsilon_{\max}} g(\varepsilon) d\varepsilon; \quad (40)$$

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

$$G(\varepsilon) = \frac{2}{\mathcal{N}} \frac{n_{e0} c_s t_{laser} S_{sheath}}{\sqrt{2T_e}} \sqrt{\frac{T_e}{2}} \left[ \exp\left(-\sqrt{\frac{2\varepsilon_{\min}}{T_e}}\right) - \exp\left(-\sqrt{\frac{2\varepsilon}{T_e}}\right) \right]; \quad (41)$$

and the normalisation constant is given by:

$$\mathcal{N} = 2 \frac{n_{e0} c_s t_{laser} S_{sheath}}{\sqrt{2T_e}} \sqrt{\frac{T_e}{2}} \left[ \exp\left(-\sqrt{\frac{2\varepsilon_{\min}}{T_e}}\right) - \exp\left(-\sqrt{\frac{2\varepsilon}{T_e}}\right) \right]. \quad (42)$$

210 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  $\varepsilon$  is obtained by evaluating:

$$\varepsilon = \left[ \sqrt{\varepsilon_{\min}} - \sqrt{\frac{T_e}{2}} \ln\left(1 - \frac{G(\varepsilon)}{G(\varepsilon_{\max})}\right) \right]^2. \quad (43)$$

## 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. The opening angle of the cone decreases as the ion energy considered increases.

215 The distribution of the polar angle,  $\theta_S$ , at which particles are produced at the laser-driven source has been approximated using the positive half of a Gaussian distribution [7]. At low kinetic energy ( $\varepsilon \sim \varepsilon_{\min}$ ), the standard deviation,  $\sigma_{\theta_S}(\varepsilon)$  of the Gaussian distribution is taken to be  $\sim 20^\circ$ .  $\sigma_{\theta_S}(\varepsilon)$  is assumed to decrease linearly with energy such that

$$\sigma_{\theta_S}(\varepsilon) = 20^\circ - 15^\circ \frac{\varepsilon}{\varepsilon_{max}}; \quad (44)$$

220 i.e.  $\sigma_{\theta_S}(\varepsilon)$  decreases from  $20^\circ$  at  $\varepsilon = 0$  to  $5^\circ$  at  $\varepsilon_{max}$ . The polar angle,  $\theta_S$ , is then chosen from the Gaussian distribution with sigma given by equation 44.

Finally, the azimuthal angle,  $\phi_S$ , is chosen 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.

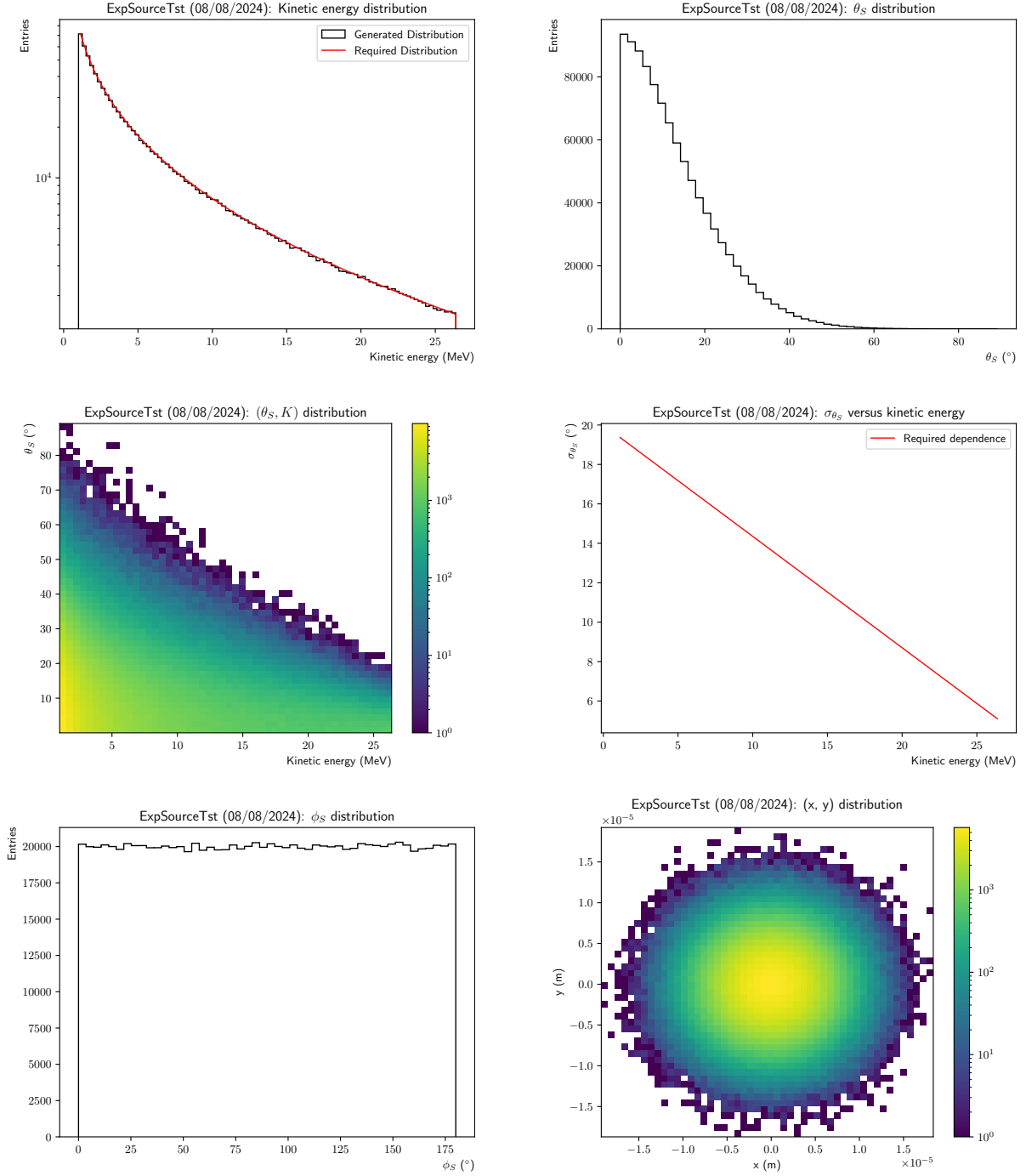


Figure 3: Kinematic distributions of particles at the point of production. Top left: the kinetic energy. The distribution generated using the algorithm described in the text is shown as the solid histogram. The required distribution defined by equation 35 is shown as the solid red line. The required distribution is normalised to the lowest kinetic-energy bin. Top right: the distribution of the polar angle with respect to the normal to the target-foil surface ( $\theta_S$ ). Centre left: the distribution of  $\theta_S$  versus kinetic energy. Centre right: the dependence of the RMS of the  $\theta_S$  distribution plotted as a function of kinetic energy. The solid circles are calculated using slices of width 1 MeV from the distribution in the centre-left panel. The expected dependence based on equation 44 is shown as the solid red line. Bottom left: distribution of the azimuthal angle,  $\phi_S$ . Bottom right:  $(x, y)$  distribution of the particle-production point.

Table 2: Parameterised laser driven

Parameter	Value	Unit
$\sigma_x$	4e-06	$\mu\text{m}$
$\sigma_y$	4e-06	$\mu\text{m}$
$\cos \theta_S _{\min}$	0.998	
$\varepsilon_{\min}$	1.0	MeV
$\varepsilon_{\max}$	25.0	MeV
nPnts	1000	
Laser power	2500000000000000.0	W
Laser energy	70.0	J
Laser wavelength	0.8	$\mu\text{m}$
Laser pulse duration	2.8e-14	s
Laser spot size	4e-07	$\mu\text{m}$
Laser intensity	4e+20	$\text{J}/\text{m}^2$
Electron divergence angle	25.0	degrees

## 5.4 Simulated distributions

Distributions  $10^6$  protons produced by the TNSA mechanism using the algorithm described above are shown in figure 3. 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 35. 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 44. As a result, the distribution of  $\theta_S$  is approximately Gaussian with a width dominated by the contribution of protons with kinetic energy close to  $\varepsilon_{\min}$ . The generated  $\phi_S$  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.

## 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) .
- 250 [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] P. Casolaro, L. Campajola, G. Breglio, S. Buontempo, M. Consales, A. Cusano, A. Cutolo, F. Di Capua, F. Fienga, and P. Vaiano, “Real-time dosimetry with radiochromic films,” *Scientific Reports* **9** (03, 2019) .

## 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`, `Utilities`, 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 3).

Table 3: Methods by which to set and access class attributes.

Method	Argument	Return	Comment
<code>getinstances()</code> <code>setDebug(Debug)</code> <code>getDebug()</code> <code>setAll2None()</code>  <code>SummaryStr()</code>	Boolean	List of instances of class  Boolean debug flag  String	Sets flag to generate debug print-out If True, generate debug print-out Set all instance attributes to <code>None</code> at start of instantiation. Text string to record parameters in debug print out.

### A.1 `BeamLineElement`

#### A.1.1 Parent class

##### A.1.1.1 Instantiation

The call to instantiate the `BeamLineElement` 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,  $\theta$ , and azimuthal,  $\phi$ , 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.1.1.2 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 4. The attributes are accessed and set using the methods defined in table 5.

Table 4: 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  $\dagger$  below the dividing line are calculated.

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

#### A.1.1.3 Processing methods

Table 6 presents the processing methods provided in the `BeamLineElement` class.

#### A.1.1.4 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.1.1.5 Utilities

Table 7 presents the utilities provided in the `BeamLineElement` class.



Table 5: Definition of access methods for the BeamLineElement class.

Set method	Get method	Comment
setName (Name)	getName ()	Set/get name of beam-line element.
setStrt (rStrt)	getStrt ()	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 ()	getRot2LabStrt ()	Set/get 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 6: 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.
ExpansionParamterFail (R)	Float	Boolean	Calculates an approximate expansion parameter and returns False if the parameter is large ( $> 1$ ). Not yet used in Transport.
Transport (V)	$6 \times 1$ np.ndarray	$6 \times 1$ np.ndarray	Transport 6D trace-space vector, V, across element. Final trace-space vector returned.
Shit2Local (V)	$6 \times 1$ np.ndarray	$6 \times 1$ np.ndarray	Transform 6D trace-space vector, V, from RPLC to laboratory coordinates. Phase-space vector in laboratory frame returned.
Shit2Laboratory (U)	$6 \times 1$ np.ndarray	$6 \times 1$ np.ndarray	Transform 6D phase-space vector, U, from laboratory coordinates to trace-space coordinates in the RPLC frame. Trace-space vector in RPLC frame returned.

Table 7: 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 CoordSys – string Proj – string		Manages plotting (visualisation) of element. “Lab” or “RPLC”, coordinate system in which to visualise element. “xz” or “yz” projection to visualise.

## A.1.2 Derived class: Facility (BeamLineElement)

### A.1.2.1 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.2.2. These arguments are passed directly to BeamLineElement.

### A.1.2.2 Instance attributes and access methods

The instance attributes are defined in table 8. The attributes are accessed and set using the methods defined in table 9.

Table 8: 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 9: 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.1.2.3 Processing methods

The Facility derived class has no processing methods.

### A.1.2.4 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 the a file instance managed by BeamIO.

### A.1.2.5 Utilities

The Facility derived class has no utilities.

### A.1.3 Derived class: `Drift (BeamLineElement)`

#### A.1.3.1 Instantiation

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

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

320 Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.2.2. These arguments are passed directly to `BeamLineElement`. The `Drift` arguments are translated into instance attributes as described in section A.1.3.2 and defined in table 10.

#### A.1.3.2 Instance attributes and access methods

The instance attributes are defined in table 10. The attributes are accessed and set using the methods defined in table 11.

Table 10: 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
<code>Length</code>	float	m	Length of drift.

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

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

325

#### A.1.3.3 Processing methods

The `Facility` derived class has no processing methods.

#### A.1.3.4 I/o methods

330 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 the a file instance managed by `BeamIO`.

#### A.1.3.5 I/o methods

The `Facility` derived class has no utilities.

### A.1.4 Derived class: `Aperture (BeamLineElement)`

#### 335 A.1.4.1 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.2.2. These arguments are passed directly to `BeamLineElement`. The `Aperture` arguments are translated into instance attributes as described in section A.1.4.2 and defined in table 12.

#### A.1.4.2 Instance attributes and access methods

The instance attributes are defined in table 12. The attributes are accessed and set using the methods defined in table 14.

Table 12: 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
<code>ParamList</code>	<code>[]</code>		List containing aperture parameters. The first parameter is an <code>int</code> and defines the aperture “Type”. The remaining elements in the parameter list are <code>floats</code> with meanings that depend on Type.
<code>ParamList[0]</code> <code>ParamList[1]</code>	<code>int</code> <code>float</code>	<code>m</code>	Type= 0; circular Radius of circular aperture
<code>ParamList[0]</code> <code>ParamList[1]</code> <code>ParamList[2]</code>	<code>int</code> <code>float</code> <code>float</code>	<code>m</code> <code>m</code> <code>m</code>	Type= 1; Elliptical Radius of elliptical aperture along $x_{RPLC}$ axis Radius of elliptical aperture along $y_{RPLC}$ axis

Table 13: Definition of access methods for the `Facility` derived class.

Set method	Get method	Comment
<code>setApertureParameters(ParamList)</code>		Set aperture paramters. Sets Type and parameters depending on Type.
	<code>getType()</code>	Get Type of aperture.
	<code>getParams()</code>	Get aperture parameters.
<code>setTransferMatrix()</code>		Set transfer matrix.

#### A.1.4.3 Processing methods

The `Aperture` processing method is defined in table 14.

#### A.1.4.4 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 the a file instance managed by `BeamIO`.

Table 14: Utilities provided by the `Aperture` derived class.

Method	Argument(s)	Return	Comment
<code>Transport (V)</code>	<code>np.ndarray</code>	<code>np.ndarray</code> or <code>None</code>	Transport trace-space vector $V$ . If $V$ falls outside of the aperture, return <code>None</code> .

#### A.1.4.5 Utilities

The `Aperture` derived class has no utilities.

### A.1.5 Derived class: `FocusQuadrupole (BeamLineElement)`

#### A.1.5.1 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.2.2. These arguments are passed directly to `BeamLineElement`. The `FocusQuadrupole` arguments are translated into instance attributes as described in section A.1.5.2 and defined in table 15. The quadrupole `Length` is required together with either the field gradient, `Strength` (equation 18), or the quadrupole  $k$  parameter, `kFQ` (equation 20).

#### A.1.5.2 Instance attributes and access methods

The instance attributes are defined in table 15. The attributes are accessed and set using the methods defined in table 17.

Table 15: 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
<code>FQmode</code>	int		If 0, use particle momentum in calculation of transfer matrix; if 1, use reference particle momentum.
<code>Length</code>	float	m	Effective length of quadrupole.
<code>Strength</code>	float	T/m	Magnetic field gradient; required if <code>kFQ</code> is not given.
<code>kFQ</code>	float	$\text{m}^{-2}$	Quadrupole $k$ paramter.

#### A.1.5.3 Processing methods

The `FocusQuadrupole` processing method is defined in table 17.

#### A.1.5.4 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) ;
```

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

Set method	Get method	Comment
<code>setFQmode(FQmode)</code>	<code>getFQmode()</code>	Set/get FQmode.
<code>setLength(Length)</code>	<code>getLength()</code>	Set/get length.
<code>setStrength(Length)</code>	<code>getStrength()</code>	Set/get strength (field gradient).
<code>setKFQ(Length)</code>	<code>getKFQ()</code>	Set/get kFQ, quadrupole $k$ parameter.
<code>setTransferMatrix()</code>		Set transfer matrix.

Table 17: Utilities provided by the `FocusQuadrupole` derived class.

Method	Argument(s)	Return	Comment
<code>calcKFQ()</code>		float	Calculates kFQ if strength is specified.
<code>calcStrength()</code>		float	Calculates Strength if kFQ is specified.

370 where `dataFILE` is the a file instance managed by `BeamIO`.

#### A.1.5.5 Utilities

The `FocusQuadrupole` derived class has no utilities.

### A.1.6 Derived class: `DefocusQuadrupole` (`BeamLineElement`)

#### A.1.6.1 Instantiation

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

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

380 Parent class arguments `Name`, `rStrt`, `vStrt`, `drStrt`, and `dvStrt` are described in section A.2.2. These arguments are passed directly to `BeamLineElement`. The `DefocusQuadrupole` arguments are translated into instance attributes as described in section A.1.6.2 and defined in table 18. The quadrupole `Length` is required together with either the field gradient, `Strength` (equation 18), or the quadrupole  $k$  parameter, `kDQ` (equation 20).

#### A.1.6.2 Instance attributes and access methods

385 The instance attributes are defined in table 18. The attributes are accessed and set using the methods defined in table 20.

#### A.1.6.3 Processing methods

The `DefocusQuadrupole` processing method is defined in table 20.

#### A.1.6.4 I/o methods

390 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);
```



Table 18: 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 <sup>-2</sup>	Quadrupole $k$ paramter.

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

Set method	Get method	Comment
<code>setDQmode(DQmode)</code>	<code>getDQmode()</code>	Set/get DQmode.
<code>setLength(Length)</code>	<code>getLength()</code>	Set/get length.
<code>setStrength(Length)</code>	<code>getStrength()</code>	Set/get strength (field gradient).
<code>setKDQ(Length)</code>	<code>getKDQ()</code>	Set/get kDQ, quadrupole $k$ parameter.
<code>setTransferMatrix()</code>		Set transfer matrix.

where `dataFILE` is the a file instance managed by `BeamIO`.

#### A.1.6.5 Utilities

The `DefocusQuadrupole` derived class has no utilities.

Table 20: Utilities provided by the `DefocusQuadrupole` derived class.

Method	Argument(s)	Return	Comment
<code>calcKDQ()</code>		float	Calculates <code>kDQ</code> if strength is specified.
<code>calcStrength()</code>		float	Calculates <code>Strength</code> if <code>kDQ</code> is specified.

### A.1.7 Derived class: `SectorDipole` (`BeamLineElement`)

#### A.1.7.1 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.2.2. These arguments are passed directly to `BeamLineElement`. The `SectorDipole` arguments are translated into instance attributes as described in section A.1.7.2 and defined in table 21.

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,  $\mathbf{v}_{ES}$ , 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:

$$\mathbf{v}_{ES} = 2\rho_0 \sin\left(\frac{\phi}{2}\right) \begin{pmatrix} \sin\left(\frac{\phi}{2}\right) \\ 0 \\ \cos\left(\frac{\phi}{2}\right) \end{pmatrix}; \quad (45)$$

where  $\rho_0$  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  $\underline{\underline{R}}_S$ , then the vector,  $\mathbf{v}_{ES}^{\text{lab}}$ , that translates from the start of the sector dipole to its end in laboratory coordinates is given by:

$$\mathbf{v}_{ES}^{\text{lab}} = \underline{\underline{R}}_S \mathbf{v}_{ES}. \quad (46)$$

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

$$\underline{\underline{R}}_E = \underline{\underline{R}}_S \underline{\underline{R}}; \quad (47)$$

where:

$$\underline{\underline{R}} = \begin{pmatrix} \cos \phi & 0 & -\sin \phi \\ 0 & 1 & 0 \\ \sin \phi & 0 & \cos \phi \end{pmatrix}. \quad (48)$$

#### A.1.7.2 Instance attributes and access methods

The instance attributes are defined in table 21. The attributes are accessed and set using the methods defined in table 22.

#### A.1.7.3 Processing methods

The `SectorDipole` derived class has no processing methods.

Table 21: 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 22: Definition of access methods for the `SectorDipole` derived class.

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

#### A.1.7.4 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 the a file instance managed by `BeamIO`.

#### A.1.7.5 Utilities

The `SectorDipole` derived class has no utilities.

### A.1.8 Derived class: `Solenoid (BeamLineElement)`

#### A.1.8.1 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.2.2. These arguments are passed directly to `BeamLineElement`. The `Solenoid` arguments are translated into instance attributes as described in section A.1.8.2 and defined in table 23. The solenoid `Length` is required together with either the magnetic field strength, `Strength` or the solenoid  $k$  parameter, `kSol` (equation 23).

#### A.1.8.2 Instance attributes and access methods

The instance attributes are defined in table 23. The attributes are accessed and set using the methods defined in table 25.

#### A.1.8.3 Processing methods

The `Solenoid` processing method is defined in table 25.

Table 23: 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 <sup>-2</sup>	GaborLens $k$ paramter required if Strength not given.

Table 24: Definition of access methods for the `Solenoid` derived class.

Set method	Get method	Comment
<code>setLength(Length)</code>	<code>getLength()</code>	Set/get length.
<code>setStrength(B)</code>	<code>getStrength()</code>	Set/get strength (solenoid magnetic field).
<code>setKSol(Length)</code>	<code>getKFQ()</code>	Set/get kSol, solenoid $k$ parameter.
<code>setTransferMatrix()</code>		Set transfer matrix.

#### 435 A.1.8.4 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 the a file instance managed by `BeamIO`.

#### A.1.8.5 Utilities

440 The `Solenoid` derived class has no utilities.

Table 25: Utilities provided by the `Solenoid` derived class.

Method	Argument(s)	Return	Comment
<code>calcKsol()</code>		float	Calculates <code>kSol</code> if strength is specified.
<code>calcStrength()</code>		float	Calculates <code>Strength</code> if <code>kSol</code> is specified.

## A.1.9 Derived class: `GaborLens` (`BeamLineElement`)

### A.1.9.1 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.2.2. These arguments are passed directly to `BeamLineElement`. The `GaborLens` arguments are translated into instance attributes as described in section A.1.9.2 and defined in table 26. The `Gabor` lens `Length` is required together with either the parameters `Bz`, `VA`, `RA`, `Rp` correspond, 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 electron number density inside the trap is calculated using either `Bz`, `VA`, `RA` and `Rp` or `kSol`.

### A.1.9.2 Instance attributes and access methods

The instance attributes are defined in table 26. The attributes are accessed and set using the methods defined in table 27.

Table 26: 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
<code>Bz</code>	float	T	Effective length of Gabor lens.
<code>VA</code>	float	V	Effective length of Gabor lens.
<code>RA</code>	float	m	Effective length of Gabor lens.
<code>RP</code>	float	m	Effective length of Gabor lens.
<code>Length</code>	float	m	Effective length of Gabor lens.
<code>Strength</code>	float	T/m	Magnetic field gradient; required if <code>kSol</code> is not given.
<code>kSol</code>	float	$\text{m}^{-2}$	$k$ parameter of the solenoid with the equivalent focusing strength.

### A.1.9.3 Processing methods

The `GaborLens` derived class has no processing methods.

### A.1.9.4 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 the a file instance managed by `BeamIO`.

Table 27: Definition of access methods for the `GaborLens` derived class.

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

### A.1.9.5 Utilities

The `GaborLens` derived class has no utilities.

## A.1.10 Derived class: `CylindricalRFCavity(BeamLineElement)`

### A.1.10.1 Instantiation

465 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.2.2. These arguments are passed directly to `BeamLineElement`. The `CylindricalRFCavity` arguments are translated into instance attributes as described in section A.1.10.2 and defined in table 28.

### A.1.10.2 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.

### A.1.10.3 Processing methods

475 The `CylindricalRFCavity` dericed class has no processing methods.

### A.1.10.4 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 the a file instance managed by `BeamIO`.

Table 28: 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		$\alpha$ paramter defined in equation ??.
wperp	float		$\omega_{\perp}$ paramter defined in equation ??.
cperp	float		$c_{\perp}$ paramter defined in equation ??.
sperp	float		$s_{\perp}$ paramter defined in equation ??.
wprll	float		$\omega_{\parallel}$ paramter defined in equation ??.
cprll	float		$c_{\parallel}$ paramter defined in equation ??.
sprll	float		$s_{\parallel}$ paramter defined in equation ??.

#### 480 A.1.10.5 Utilities

The `CylindricalRFCavity` derived class has no utilities.

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

Set method	Get method	Comment
setGradient (Gradient)	getGradient ()	Set/get peak electric field.
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.
setTransferMatrix ()		Set transfer matrix.

## A.2 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 matrices defined in section A.1. 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.

### A.2.1 Instantiation

The call to instantiate the BeamLineElement class is:

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

BeamLineSpecificationCSVfile is the full path of the CSV file containing the beam-line specification. 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.2.2 Instance attributes and access methods

The instance attributes are presented in table 30 and the access methods are summarised in table 31.



Table 30: Definition of attributes of instances of the BeamLine class.

Attribute	Type	Comment
BeamLineSpecificationCSVfile*	path	Full path to beam-line specification file.
BeamLineParamPandasInstance	dataframe	Pandas data frame containing beam line parameters.
Element	list	List of instances of BeamLineElement class containing point cloud data.

Table 31: Definition of access methods for the BeamLine class.

Set method	Get method	Comment
setSrcTrcSpc (SrcTrcSpc)	setSrcTrcSpc ()	Set trace space at source; SrcTrcSpc is represented 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.

### A.2.3 Processing methods

500

Table 32 presents the processing methods provided in the BeamLine class.

### A.2.4 I/o methods

505

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 the a file instance managed by BeamIO. In addition the csv file containing the specification of the beam line is read in using the method:

```
csv2pandas (csvFILE) ;
```

where csvFILE (path) is the full path to the csv file.

### A.2.5 Utilities

510

The BEamLine class has no utilities.

Table 32: Processing methods provided by the BeamLine class.

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

## A.3 Particle and reference particle

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.3.1 Particle

#### A.3.1.1 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.3.1.2 Instance attributes and access methods

The instance attributes are presented in table 33 and the access methods are summarised in table 34.

Table 33: Definition of attributes of instances of the `Particle` class.

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

Table 34: Definition of access methods for the `Particle` class.

Set method	Get method	Comment
<code>setSpecies</code>	<code>getSpecies</code>	Set/get particle species.
<code>setLocation</code>	<code>getLocation</code>	Set/get list of locations location.
<code>sets</code>	<code>gets</code>	Set/get list of <code>s</code> coordinates.
<code>setTraceSpace</code>	<code>getTraceSpace</code>	Set/get list of trace-space vectors.
<code>setRPLCPhaseSpace</code>	<code>setRPLCPhaseSpace</code>	Set/get list of phase-space vectors in RPLC coordinates.
<code>setLabPhaseSpace</code>	<code>getLabPhaseSpace</code>	Set/get list of phase-space vectors in Lab coordinates.

### A.3.1.3 Processing methods

Table 35 presents the processing methods provided in the `Particle` class.

Table 35: Processing methods provided by the `Particle` class.

Method	Argument(s)	Return	Comment
<code>initialiseSums()</code>  <code>incrementSums(iPrtcl)</code>  <code>calcCovarianceMatrix()</code> <code>evaluateBeam()</code>	<code>Particle</code> instance		Initialises sums used to calculate covariance matrix. Increment sums used to calculate covariance matrix. Calculate covariance matrix. Work through locations and calculate paramters from covariance matrix.

### A.3.1.4 I/o methods

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

### A.3.1.5 Utilities

The utilities provided by the `Particle` class are listed in table 36.

Table 36: Utilities provided by the `Particle` class.

Method	Argument(s)	Return	Comment
<code>cleanBeams()</code> <code>printProgression()</code> <code>getHeader()</code>  <code>getLines()</code>  <code>createReport()</code>			Delete (using <code>del</code> ) instances of beam class. Prints the paramters at each location. Returns header for pandas data frame used to store summary. Returns lines to be used to create summary pandas data frame. Creates <code>csv</code> file containing summary of beam progression.

## B Set-up and run

### Introduction

535 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:

540

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

### Dependencies and required packages

545 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`
  2. `source venv/bin/activate`
  3. `python -m pip install pandas scipy matplotlib`
- 550 • This creates the director `venv` that contains files related to the virtual environment.

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

555 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:

- 560 • 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:

565 `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 specify the various beam lines presently implemented.

570 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.

575