# Transverse Linear Beam Dynamics Primer using Python

G. Sterbini, A. Latina, A. Poyet, CERN
V. Ziemann, Uppsala University

September 2, 2022

## 1 Introduction

The purpose of this document is to complement with hands-on numerical exercises the CAS lectures on Transverse Linear Beam Dynamics. The numerical calculations will be illustrated with the help of the Python [1] general-purpose programming language[1]. The reader be asked to "play around" with the software and then use it to solve simple beam optics problems.

This Chapter will introduce the basic concepts and the software framework and will help to maximize the benefit drawn from the tutorial sessions at CAS. We start by introducing some basic theoretical concepts first, but illustrate the ideas with, often trivial, examples in Python as we go along. Exercises are interspersed in the text and readers are encouraged to solve them. We will discuss the solutions at during the hands-on sessions at CAS.

This document focuses on Python, loosely following [4], and its heavily based on the Octave/MATLAB version of this report [5]. Most examples in the first part of this tutorial are addressing the single particle motion.

**Disclaimer**

- These exercises address the absolute newcomer in the field and the level is adapted accordingly. The exercises will help to visualize the concepts and provide an introduction to the basic numerical approach.
- In this hands-on tutorials we will use Python 3.8 and we assume the reader has some basic knowledge of Python.

## 2 Ray tracing

The motion of charged particles with respect to the "center of the beam pipe" in an accelerator conceptually resembles the motion of optical rays with respect to the optical axis. In the latter case, a ray at a given longitudinal position $s$ is characterized by its distance $x$ and its angle $x'$ with respect to the optical axis. In the absence of lenses or prisms, the light ray moves on a straight line. Its distance to the optical axis $\hat{x}$ at a downstream location $\hat{s}$ therefore changes according to $\hat{x} = x + Lx'$, where we assume that $L$ is the distance between $s$, where the ray's coordinates are $(x, x')$, and $\hat{s}$, where the coordinates are $(\hat{x}, \hat{x}')$. Since it moves on a straight line, the angle does not change from $s$ to $\hat{s}$, such that we have $\hat{x}' = x'$. The two equations for $\hat{x}$ and $\hat{x}'$ can be combined in a single equation,

$$\begin{pmatrix} \hat{x} \\ \hat{x}' \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & L \\ 0 & 1 \end{pmatrix}}_{D} \begin{pmatrix} x \\ x' \end{pmatrix} , \tag{1}$$

where the motion from $s$ to $\hat{s}$ is encapsulated in the matrix.

[1] The reader can also use Octave [2] or MATLAB [3] to solve similar problems.

**Exercise 1.** *Show that multiplying two $D$ matrices, one with $L_1$ and the other with $L_2$ in the upper right corner, produces a $D$ matrix with the sum of the distances in the upper right corner.*

When a ray passes a focusing (thin) lens, its angle $x'$ changes proportionally to the transverse distance $x$ of the ray from the center of the lens, which we assume to be aligned with the optical axis. Recall the definition of the *focal length*, which requires a lens to deflect all parallel rays, independently of their transverse position, in such a way, that the rays cross the optical axis at the same point – the focal point. This point is located at a distance $f$, the *focal length*, downstream of the lens. We therefore require that the change of angle in the lens is given by $\hat{x}' = x' - x/f$. Since we assumed the lens to be very thin, the transverse position of the ray does not change and we have $\hat{x} = x$. Here we assumed that the coordinates of the rays, immediately upstream of the lens are $(x, x')$ and immediately downstream they are $(\hat{x}, \hat{x}')$. Note that also these two equations can be combined into the following matrix-valued equation

$$\begin{pmatrix} \hat{x} \\ \hat{x}' \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ -1/f & 1 \end{pmatrix}}_{Q} \begin{pmatrix} x \\ x' \end{pmatrix} , \tag{2}$$

where a $f > 0$ describes a focusing lens and $f < 0$ a defocusing lens.

**Exercise 2.** *How do you describe a ray that is parallel to the optical axis?*

**Exercise 3.** *How do you describe a ray that is on the optical axis?*

**Exercise 4.** *Show that, by multiplying the respective matrices, a parallel ray, which first passes through a lens with focal length $f$ and then moves on a straight line, actually crosses the optical axis at a distance $L = f$ downstream of the lens.* **Hint***: think a little extra about the correct order of the matrices.*

A preliminary analysis of optical systems can easily be described by the matrices, they are called Ray – or ABCD-matrices in the optics literature [6]. But optics is not exactly our topic, and we will therefore turn to charged particles, instead.

As a matter of fact, we can describe the linear motion of charged particles in the presence of magnetic lenses by the same mathematical framework, namely with matrices; in accelerator physics they are usually referred to as *transfer matrices*. We only have to establish a few correspondences beforehand. For charged particles, the optical axis can be visualized as the center of the beam pipe. Moreover, as already mentioned, at a longitudinal position $s$, a particle can also be characterized by its distance $x$ and and angle $x'$ with respect to the center of the beam pipe. Here, we need to point out that for charged particles this description is valid under the approximation of moderately small angles $x'$, which is referred as the *paraxial approximation* and is valid for angles in the range of a few tens of mrad. But this is usually valid in accelerators – 10 mrad correspond to a change of distance of 10 mm per meter!

In the absence of external influences – this can be visualized as a piece of beam pipe without magnets – charged particles also move on a straight line. The matrix that maps the position $x$ and the angle $x'$ from one end of the empty pipe to the other end is the same we used for light rays and is given in Equation 1. This matrix is commonly referred to as the *matrix for a drift space* as the empty piece of beam pipe is referred to as a *drift space*.

The elements corresponding to the optical lenses are magnetic elements with a magnetic field that depends linearly – in linear optics – on the transverse position $x$. Note the resemblance with the light rays: all parallel rays have to go through the focal point downstream and that requires that the deflection angle of each ray has to be proportional to its distance from the optical axis. In the same way charged particles are deflected more, due to the Lorentz force that the moving particles experience, if they are further from the axis of the magnet. Magnets with the required linearly rising magnetic field are *quadrupoles* and, if they are moderately short with respect to its focal length $f$, they can be described by the matrix in Equation 2. Their inverse focal length $1/f$ is directly proportional to the magnetic gradient $\partial B_y/\partial x$, but we leave the details of the conversion to more advanced texts [4, 7, 8, 9].

We note, however, that the magnetic fields in the quadrupoles have to fulfill Maxwell's equations. This forces the transverse components $B_x$ and $B_y$ of the magnetic field to obey the curl equation: $\partial B_y/\partial x - \partial B_x/\partial y = 0$. A linearly increasing field component in one direction causes the other component to decrease. Quadrupoles therefore focus particles in one transverse plane, but defocus the same particles in the other plane. Since we will only consider one plane in this first part of the tutorial, we will not encounter this "feature" directly. We will, however, build beamlines consisting of both focusing and defocusing quadrupoles. The

motivation to use both types of quadrupoles is to be able to describe the other plane of the accelerator as well.

There are many more magnets that are used to guide the beam – big dipoles and small steering magnets – and other magnets to correct various effects, but we will, in the early sections of this tutorial, confine ourselves to drift spaces and quadrupoles and will build larger beam optical systems with these elements alone. We will implement most matrix manipulations in software in order to avoid excessive matrix calculations by hand.

# 3 A simple code to transport particles

In the following we are going to introduce a simple code for transporting particles in a beamline using Python. The code is deliberately kept simple in order to make its structure transparent, sometimes at the expense of its overall performance.

## 3.1 The element: a matrix representation

Finally our task is to follow a particle through a sequence of drift spaces and quadrupoles (elements), described by the matrices from Equation 1 and 2.

To do so, we first encode the matrices in a convenient way by using functions, e.g.,

```
import numpy as np
def D(L):
    '''Returns the matrix of a L-long drift'''
    return np.array([[1, L],[0, 1]])
```

The commands above define a function `D(·)` that receives the length `L` as input and returns the corresponding drift matrix from Equation 1. Likewise, we define the function

```
def Q(f):
    '''Returns the matrix of a quadrupole with focal length f'''
    return np.array([[1, 0],[-1/f, 1]])
```

which returns the matrix from Equation 2.

## 3.2 Combining several elements: the beamline

By combining drifts an quadrupoles we can define a beamline. We can think a beamline as a new element and associate to it an "equivalent" matrix obtained by the multiplication of the matrices of its elements.

**Nota Bene**

In Python 3, the operator for matrix multiplication is the '@' operator.

For example, let us assume to have a sequence of a drift of 2.5 m followed by a quadrupole with a focal length of 3 m and a final drift of drift of 1.2 m. The equivalent matrix, $M$, of the combination of these elements is:

```
# Multiplication of matrices
M = D(1.2) @ Q(3) @ D(2.5)
```

**Nota Bene**

Please note the ordering of the matrices. Since we are considering to right-multiply our matrices with the particle vector $(x,x')$, the elements matrices should be ordered from the right to the left.

A natural way to define the beamline is to consider it as a list of elements.

```
mybeamline = [D(2.5), Q(3), D(1.2)]
```

This has the advantage to use the '+' and '*' operation to combine beamlines as in the following example

```
# two simple beamlines
mybeamline_1 = [D(2.5), Q(3), D(1.2)]
mybeamline_2 = [D(2), Q(-1)]
# building a third beamline using the list algebra of python
mybeamline_3 = (mybeamline_1*2+mybeamline_2)*3
```

Then we can compute the equivalent matrix of the beamline.

```
1 def getEquivalentElement(beamline):
2     '''Returns the equivalent single element of a beamline.'''
3     # np.eye(2) gives the 2x2 identity matrix
4     equivalentElement = np.eye(2)
5     # NB: we reverse the order of the beamline ([ -1:: -1])
6     for elements in beamline[-1::-1]:
7         equivalentElement = equivalentElement @ elements
8     return equivalentElement
```

In line 6 of the previous code block the [-1::-1] expression is a trick to reverse the order of the element in the beamline (the last element will become the first one considered in the loop).

With the equivalent matrix we can therefore compute, given the initial position (x, x') of the particle at the entrance of the beamline, its position at the end.

```
1 # initial position of the particle
2 x_start = [[0.2],[.3]]
3 # final position of the particle
4 x_end = getEquivalentElement(mybeamline_3) @ x_start
```

The above approach has at least one limit: we cannot resolve the position of the particle "inside" the beamline but only at its end.

To overcome this limitation, we can build a tracking engine that allows us to track the particle for all the elements in a beamline. For instance we can define

```
1 def transportParticles(x0, beamline):
2     '''Returns the list of position in the beamline
3     starting from x0.'''
4     x = [x0]
5     for elements in beamline:
6         x.append(elements @ x[-1])
7     return np.array(x).transpose()[0]
```

and then

```
1 transportParticles(x_start, mybeamline_3)
```

Still there is a problem: we are not able to associate to each element a corresponding $s$-position. We will overcome this limitation in the next Section.

## 3.3 Tracking a particle along a beamline

As presented, having an "equivalent" matrix allows to build complex elements starting from simpler ones. The drawback of this approach is that we lose the information of the particle position $s$ along the line.

In order to recover this flexibility we can define the elements as a dictionary having both the information of the matrix and the one relative to the elements length. For example

```
1  def D(L):
2      '''Returns the list of a L-long drift'''
3      # NB: we return a list with a dict
4      # the dict contains the matric (the transformation)
5      # and the element length
6      return [{'matrix':np.array([[1, L],[0, 1]]), 'length':L}]
7
8  def Q(f):
9      '''Returns the list of a quadrupole with focal length f'''
10     # NB: we return a list with a dict
11     # the dict contains the matric (the transformation)
12     # and the element length
13     return [{'matrix':np.array([[1, 0],[-1/f, 1]]), 'length':0}]
```

We can therefore redefine the `getEquivalentElement` and `transportParticles` methods accordingly.

```
1 def getEquivalentElement(beamline):
2     '''Returns the equivalent single element of a beamline'''
3     # we start from an identity matrix (np.eye)
4     # with the same dimension of the matrix of the
5     # first element of the beamline
6     equivalentElement = np.eye(beamline[0]['matrix'].shape[0])
7     length = 0
```

```
    # NB: we reverse the order of the beamline ([-1::-1])
    for elements in beamline[-1::-1]:
        # we multiply the matrices
        equivalentElement = equivalentElement @ elements['matrix']
        # and we add the lengths
        length = length + elements['length']
    # we return the dict with the "usual" keys (matrix and length)
    # as for the definition of the D and Q functions
    return {'matrix':equivalentElement, 'length':length}

def transportParticles(x0,beamline):
    '''Track the particle(s) x0 along the beamline'''
    coords = [x0]
    s = [0]
    for elements in beamline:
        coords.append(elements['matrix'] @ coords[-1])
        s.append(s[-1] + elements['length'])
    coords = np.array(coords).transpose()
    return {'x':  coords[:,0,:],
            'px': coords[:,1,:],
            's':  np.array(s),
            'coords': coords,}
```

and then

```
# NB: since each element is a list, we can concatenate
# with the `+` and `*` operators.
mybeamline_1 = D(2.5) + Q(3) + D(1.2)
mybeamline_2 = D(2) + Q(-1)
# building a third beamline using the list algebra of python
mybeamline_3 = (mybeamline_1*2+mybeamline_2)*3
```

For convenience, we explicitly output the x and px parameters in the function `transportParticles`. In summary, using the simple approach shown above, we can

- describe an element as a dictionary giving the information about linear matrix and element length,
- combine several elements in a list (beamline) and use a convenient algebra ('+' and '*') to concatenate and replicate different beamlines,
- reduce a beamline in a an equivalent element,
- track a particle along a beamline.

As example, we map the initial coordinates `x_start`, specified as a column vector, along the beamline and plot the transverse position x as a function of the longitudinal position $s$.

```
import matplotlib.pyplot as plt
f=1
L=1/10

# NB: combination of list to prepare a generic beamline
beamline= 5*D(L)+Q(-f)+10*D(L)+Q(.84*f)+5*D(L)
x_start = [[0.002],[0]]
# We encourage to try other initial condition, e.g.
# x_start = [[0],[0]]

# NB: we concatenate 40 beamline
output=transportParticles(x_start,beamline*40)
# plotting the results

plt.plot(output['s'],output['x'][0]*1000,'o-b',lw=3)
plt.grid(True)
plt.xlabel('s [m]')
plt.ylabel('x [mm]')
```

The result of the above snippet of code is shown in Figures 1-2.

We will see in the next exercises how to use those simple functions to put our hands-on the beam dynamics. A limited part of the exercises comes with some hints to suggest the direction of the solutions. The solutions are available in a separate document [11].

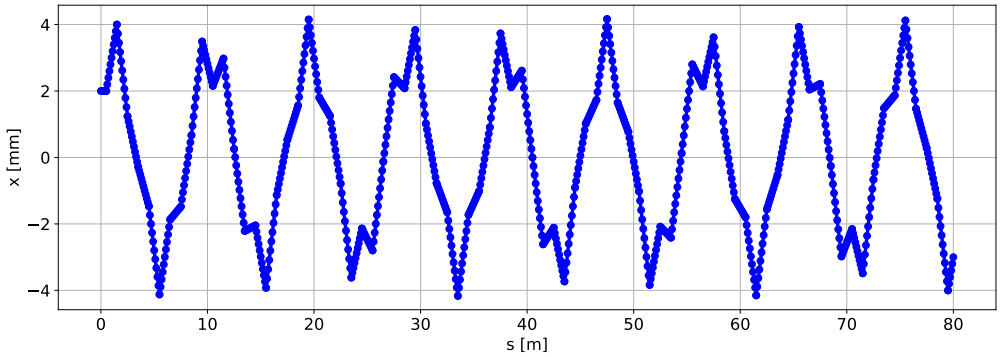


Figure 1: The x-position of our particle along the beamline.

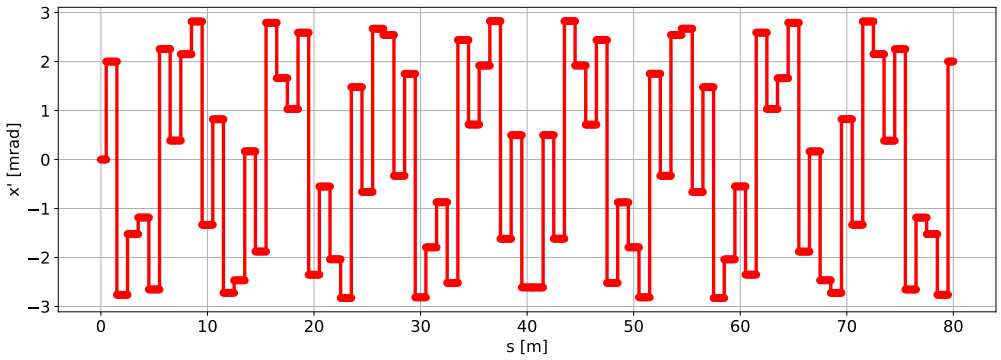


Figure 2: The x'-position of our particle along the beamline.

**Exercise 5.** *Set $F = 3$ and numerically verify what you found in Exercise 4, namely that parallel rays cross the axis after a distance $L = F$.*

**Hint***: use, first, the `getEquivalentElement` and, then, the `transportParticles` functions, as defined in Section 3.3, and show that the conclusion is the same.*

# 4 Studying simple beamlines

There are notable beamlines frequently considered in synchrotrons, such as the FODO beamline (also known as FODO lattice):

```
#the FODO beamline
f = 2.5
beamline = 5*D(0.1) + Q(f) + 10*D(0.1) + Q(-f) + 5*D(0.1)
```

The first line defines the focal length `f` to be 2.5 m and then defines a list, that follows the conventions defined above. The beamline is composed summing 5 lists. The first list of the beamline describes 5 segments of a drift space, each being 0.1 m long. The second list describes a thin quadrupole. The third list describes a drift space, consisting of ten segments with length 0.1 m each. The fourth list describes a second quadrupole, this time with a negative focal length. The last list is equal to the first one. We call the list that describes our beamline `FODO`, because it consists of an alternating sequence of focusing (F) and defocusing (D) quadrupoles, separated by not-focusing (O) drift spaces. The arcs of the LHC collider or the SPS synchrotron are based on them. The reason to alternate the quadrupole polarity is to focus in both transverse planes equally, despite each quadrupole affects the two planes in the opposite manner.

**Exercise 6.** *Recall that the imaging equation for a lens is $1/b + 1/g = 1/f$, which corresponds to a system of one focusing lens with focal length $f$, sandwiched between drift spaces with length $b$ and $g$, respectively. Write a beamline description that corresponds to this system. We will later return to it and analyze it.*

**Exercise 7.** *Prepare initial coordinates that describe a particle that is on the optical axis, but has an initial angle $x'$ and plot the position $x$ along the beamline.*

**Exercise 8.** *Plot the angle $x'$ along the beamline.*

The code only shows the trajectory in a single FODO cell. If we want to display the trajectory, for example, through five consecutive cells, we only have to replace the definition of the `beamline` early in the script by

```
beamline = 5*beamline
```

**Exercise 9.** *Plot both the position $x$ and the angle $x'$ through five cells.*

**Exercise 10.** *Plot the position $x$ through 100 cells, play with different values of the focal length `F` and explore whether you can make the oscillations grow.*

**Exercise 11.** *Use the beamline for the imaging system you prepared in Exercise 6 and launch a particle with $x_0 = 0$ and an angle of $x'_0 = 1$ mrad at one end. Verify that this particle crosses the center of the beam pipe at the exit of the beamline, provided that $b, g$, and $f$ satisfy the imaging equation that is shown in Exercise 6.*

Up to now we transported single particles through a beamline. But a beam consists of more than a few particles and in the following section we will show how to describe such an ensembles of particles.

# 5 Many particles, the beam

Each of the many particles in a beam has its individual coordinates $(x, x')$. Under most circumstances are the positions $x$ and angles $x'$ distributed according to a normal (also referred as Gaussian) distribution. We create the coordinates of $N$ particles as a $2 \times N$ array `beam` that we fill with normally distributed random numbers with the help of the function `randn()`.

The approach is very similar to the one used for the single particle but in this case the beam will be represented by a $2\times$ `N_particles` matrix.

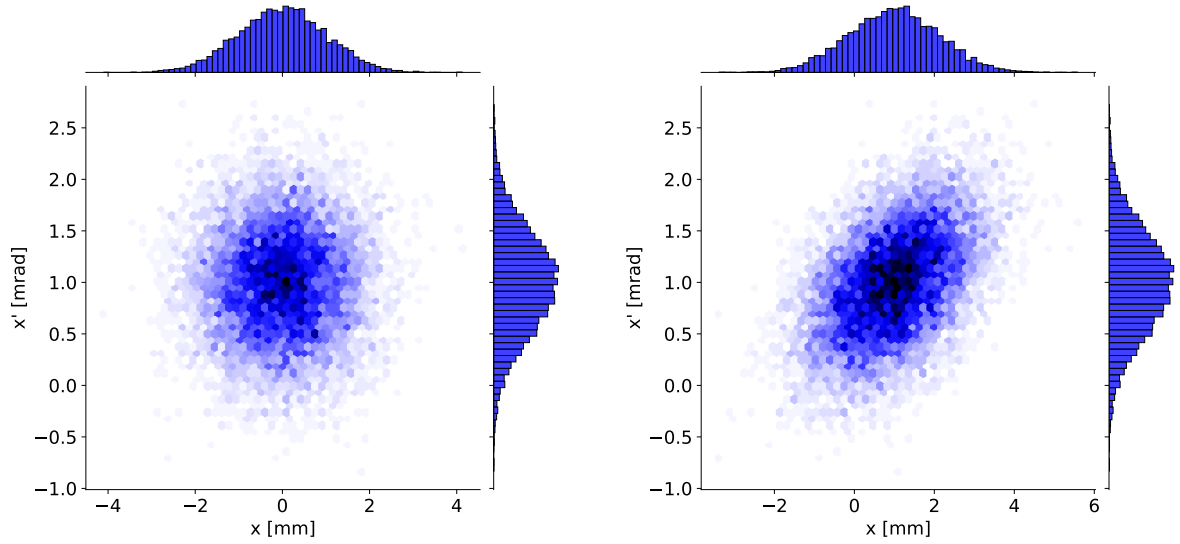


Figure 3: Particles at start (left) and after 1 m drift space (right).

```
N_particles = 1000
beam = np.random.randn(2, N_particles)
x0 = 0
xp0 = 1
sigx = 1
sigxp = 0.5
beam[0,:] = sigx*beam[0,:] + x0
beam[1,:] = sigxp*beam[1,:] + xp0
```

The random numbers returned by `np.random.randn` have a mean of zero and unit rms, such that we scale them by multiplying the first coordinate by the rms beam size `sigx` and the second coordinate by the rms angular divergence `sigxp`. We can also introduce the initial offsets `x0` and `xp0`, which are then the respective means of the distribution. We can now verify that the distribution has the desired properties by calculating the beam parameters via

```
print(f'Beam mean x [mm]: {np.mean(beam[0,:])}')
print(f"Beam mean  x' [mrad]: {np.mean(beam[1,:])}")
print(f'Beam rms size[mm]: {np.std(beam[0,:])}')
print(f"Beam rms divergence [mrad]: {np.std(beam[1,:])}")
```

where `beam[0,:]` and `beam[1,:]` refer to the first and second row of the matrix `beam`, which contains the positions and angles of all particles. The built-in functions `np.mean()` and `np.std()` return the average and rms values of the distribution, respectively.

**Exercise 12.** *Define an ensemble of 1000 particles with an arbitrary first order (*`x0`, `xp0`*) and second order momenta(* `sigx` *and* `sigxp`*). Verify the angular divergence of the beam is the one set.*

We can visualize the particle distribution by plotting one and two-dimensional histograms and contour plots. We can display the distribution using the `seaborn` plotting package, see left-side plot of Figure 3.

```
import seaborn as sns
g = sns.jointplot(x=beam[0,:], y=beam[1,:], kind="hex", color="b")
g.set_axis_labels("x [mm]", "x' [mrad]");
```

We then use a transfer matrix `R` to propagate the `beam` to a downstream location, here through a drift space with a length of 1 m, by multiplying the `beam` with `R` , as shown in the following example.

```
beamAfterDrift = D(1)[0]['matrix'] @ np.array(beam)
```

Calling again the `seaborn` plotting function one gets the right-hand side in Figure 3. We find that the distribution has slightly changed shape: the ellipse has rotated. Moreover, the mean of the horizontal distribution has changed to 1 mm. This is a consequence of the initial angle $x_0' = 1\,\text{mrad}$, which caused the beam to

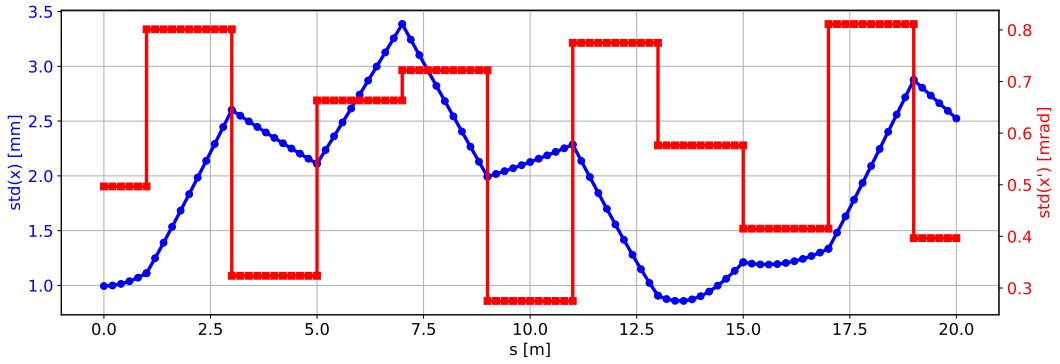


Figure 4: The beam size along the beamline made of five FODO cells.

move towards the positive x-axis by 1 mm over the 1 m long drift space. The horizontal width has increased to 1.1 mm.

Note that we specified the transverse dimensions and angle in mm and mrad instead of m and rad. This is permissible, because transfer matrices describe a `linear` transformation, and multiplying the vector to the right by a factor, here $10^3$, causes the output on the left of the matrix multiplication to be scaled by the same number.

**Exercise 13.** *Transport the beam distribution of Exercise 12 in a drift of length 1 m. Compare the initial and final distribution.*

*Test of linearity. Scale the input vector by 17 times the month of your birthday (85 if you are born in May) and verify that the output vector from the matrix multiplication has changed by the same factor.*

*Now launch 3 particles such that they define a triangle of surface A. Verify that this linear transport preserves the area of the triangle.*

**Nota Bene**

The 'area preservation' property of the previous exercise is a very profound concept. In linear optics is related to the fact that the matrices we are considering are **symplectic**...

We will continue to use mm and mrad for the transverse coordinates and will now calculate the beam size after every segment in the beamline. We use the previous approach to build a beamline of five consecutive FODO cells. Then we execute the following code:

```
#prepare the optics
f = 2.5
L_2 = 2
fodo_lattice = 5*D(L_2/10)+Q(-f)+10*D(L_2/10)+Q(f)+5*D(L_2/10)

#prepare the beam
Npart = 10000
beam = np.random.randn(2, Npart)
x0 = 0
xp0 = 1
sigx = 1
sigxp = 0.5
beam[0,:] = sigx*beam[0,:]+x0
beam[1,:] = sigxp*beam[1,:]+xp0

output = transportParticles(beam, 5*fodo_lattice)

plt.plot(output['s'], np.std(output['x'],0), 'o-b', lw=3)
plt.grid(True)
plt.xlabel('s [m]')
plt.gca().set_ylabel('std(x) [mm]', color='b')
plt.gca().tick_params(axis='y', labelcolor='b')
```

```
23
24 ax2 = plt.gca().twinx()  # instantiate a second axes that shares the same x-axis
25 ax2.set_ylabel("std(x') [mrad]", color='r')
26 ax2.tick_params(axis='y', labelcolor='r')
27 plt.plot(output['s'], np.std(output['px'],0), 's-r', lw=3)
```

It is important to note that the approach of the previous code is identical to the one for single particle tracking. Indeed the `beam` can represent a single particle or an ensemble of particle. At the end of the script, we display the beam size as a function of $s$ and annotate the axes. Figure 4 shows the resulting plot. We observe that the beam size grows over the first meter, where it traverses a defocusing quadrupole, which increases the beam size further until at $s = 3\,\mathrm{m}$. There the first focusing quadrupole reduces the beam size until the defocusing quadrupole at $s = 5\,\mathrm{m}$ increases the beam size again, then the focusing quadrupole at $s = 7\,\mathrm{m}$ reduces it again; and so forth. We also observe that the `beam` initially defined with `sigx`$= 1\,\mathrm{mm}$ can become larger up to an rms beam size `sigx`$> 3\,\mathrm{mm}$.

**Exercise 14.** *Using the 5 FODO cells of Exercise 9, transport the beam of Exercise 13 and plot its rms size and divergence along the line.*

**Exercise 15.** *Starting from Exercise 14, what happens if you (a) increase or (b) reduce by a factor 2 the initial beam size and divergence (*`sigxp`*)?*

Normally, one is not really interested in the histogram of the particle distribution. Information about the centroid position, the beam size, and possibly the angular divergence are enough. So, wouldn't it be nice, if we could move the interesting quantities around, instead of thousands of sample particles? It turns out that this is possible and that is the topic of the next section.

# 6 Moving beams around

To simplify the writing of many matrix-valued equations, we introduce the notation that the position $x$ is denoted by $x_1$ and the angle $x'$ by $x_2$. This allows us to express the propagation of the particle coordinates as $\hat{x}_i = \sum_{j=1}^{2} R_{ij} x_j$. If we have to deal with many particles we label them by a second subscript, separated from the first by a comma; $x_{2,17}$ is thus the angle of particle number 17.

When we calculated the average beam position and angle using numpy functions, the software actually performed the following operations

$$X_1 = \langle x_1 \rangle = \frac{1}{N} \sum_{m=1}^{N} x_{1,m} \qquad \text{and} \qquad X_2 = \langle x_2 \rangle = \frac{1}{N} \sum_{m=1}^{N} x_{2,m} \;, \tag{3}$$

where the index $m$ sums over the $N$ particles of the beam. The notation with the angle brackets denotes thus averaging over the ensemble of particles. We also introduced the quantities $X_1$ and $X_2$ to denote the averages. Likewise, the rms quantities are calculated via

$$\begin{aligned} \sigma_x^2 &= \sigma_{11} = \langle (x - X_1)^2 \rangle = \frac{1}{N} \sum_{m=1}^{N} (x_{1,m} - X_1)^2 \\ \sigma_{x'}^2 &= \sigma_{22} = \langle (x' - X_2)^2 \rangle = \frac{1}{N} \sum_{m=1}^{N} (x_{2,m} - X_2)^2 \;. \end{aligned} \tag{4}$$

By inspection, we should not be surprised that there is also a third variant of the above sums, given by

$$\sigma_{12} = \langle (x - X_1)(x' - X_2) \rangle = \frac{1}{N} \sum_{m=1}^{N} (x_{1,m} - X_1)(x_{2,m} - X_2) \;, \tag{5}$$

which describes the correlation between position (index 1) and angle (index 2). This quantity $\sigma_{12}$ is actually capable of describing the rotation of the contour in the right-hand plot in Figure 3 that showed up after propagating the initial beam through a 1 m long drift space.

These five quantities $X_1, X_2, \sigma_{11}, \sigma_{12}$, and $\sigma_{22}$ describe all the interesting properties of a beam. Note that $X_1$ and $X_2$ are the first moments of the two-dimensional beam distribution and the $\sigma_{ij}$ with $i = 1, 2$

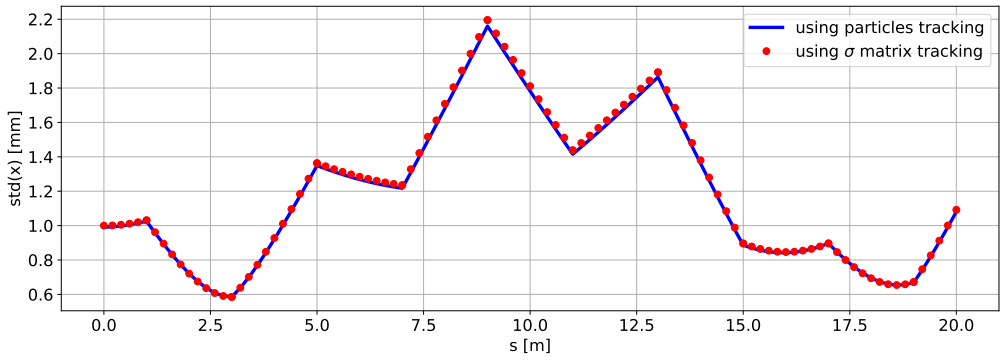


Figure 5: The beam size (blue line) from Figure 4 and calculated from Equation 7 (red markers).

are the three independent second moments; actually they are the central moments, because they describe the second moments with respect to the centroids. Note that usually the three independent second moments can be written to form a symmetric $2 \times 2$–matrix that is given by

$$\sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix} , \tag{6}$$

with $\sigma_{21} = \sigma_{12}$. We use the convention that sigmas without subscripts denote the matrix and sigmas with subscripts denote the matrix elements. We also point out that the sigma matrix changes from one location $s$ in the beamline to another location $\hat{s}$. Remember that Figure 4 shows the beam size $\sigma_x = \sqrt{\sigma_{11}}$ as a function of $s$ along the beamline.

In the more advanced literature [4, 7], but also in Appendix A, it is shown that the first and second moments propagate according to

$$\vec{X}(s_2) = R\,\vec{X}(s_1) \qquad \text{and} \qquad \sigma(s_2) = R\,\sigma(s_1)\,R^t\,, \tag{7}$$

where $\vec{X}(s)$ denotes the column vector with entries $X_1$ and $X_2$ at longitudinal location $s$. The first equation describes the remarkable fact that the centroid of the beam – the first moments – propagate in the same way as single particles. The second equation describes the propagation of the beam matrix, as defined in Equation 6. In Equation 7 we use the common convention to denote the transpose of the matrix $R$ by $R^t$.

The two equations in Equation 7 are particularly convenient and efficient to implement in software, because they just describe matrix multiplication of the centroid $\vec{X}$ and the sigma matrix $\sigma$, both of which describe the beam, and the transfer matrices $R$, which describe the hardware of the beamline. We can now implement these equations in the following code segment following code segment, where we add the calculations according to Equation 7.

```
#lattice
f = 2.5
L_2 = 2
fodo_lattice= 5*D(L_2/10) + Q(f) + 10*D(L_2/10) + Q(-f) + 5*D(L_2/10)

#prepare the beam
Npart = 10000
beam0 = np.random.randn(2, Npart)
x0 = 0;
xp0 = 1
sigx = 1;
sigxp = 0.5/2;
beam0[0,:] = sigx*beam0[0,:] + x0
beam0[1,:] = sigxp*beam0[1,:] + xp0

#prepare the sigma matrix
sigma0 = np.array([[sigx**2, 0], [0, sigxp**2]])
output_a = transportParticles(beam0, 5*fodo_lattice)

```

```
20
21 def transportSigmas(sigma, beamline):
22     '''Transport the sigma matrix along the beamline'''
23     coords = [sigma]
24     s = [0]
25     for elements in beamline:
26         coords.append(elements['matrix'] @ coords[-1] @ elements['matrix'].transpose())
27         s.append(s[-1] + elements['length'])
28     coords = np.array(coords).transpose()
29     return {'sigma11': coords[0][0],
30             'sigma12': coords[0][1],
31             'sigma21': coords[1][0], # equal to sigma12
32             'sigma22': coords[1][1],
33             's':  np.array(s),
34             'coords': coords,}
35
36 output_b = transportSigmas(sigma0, 5*fodo_lattice)
37
38 plt.plot(output_a['s'], np.std(output_a['x'],0), '-b', lw=3, label='using particles
      tracking')
39
40 plt.plot(output_b['s'], np.sqrt(output_b['sigma11']), 'or', lw=3, label='using $\sigma$
      matrix tracking')
41 plt.grid(True)
42 plt.xlabel('s [m]')
43 plt.ylabel('std(x) [mm]')
44 plt.legend()
```

In this code segment, we adopt the same approach used for the `beam0` vector to `sigma0`. We initialize the `sigma0` and then we loop over the beamline element and propagate according to Equation 7. Finally, we plot both the newly calculated beam sizes and those calculated previously as a function of the $s$-position. The result, displayed in Figure 5, shows very good agreement of the beam sizes. Henceforth, we will only use Equation 7 for propagating beams through beamlines.

**Exercise 16.** *Using Equation 7, display (a) the average position of the particles along the beamline. Likewise, (b) display the angular divergence. Compare with the result you found in Exercise 14.*

In Figure 5 the beam size $\sigma_x$ oscillates in a somewhat uncontrolled fashion along the beamline. Next we will explore whether we can find periodic oscillations that repeat after each cell.

**Exercise 17.** *Can you find an initial beam matrix* `sigma0` *that reproduces itself at the end of the beamline?* **Hint***: proceed by try and error, you will see it is not trivial.*

# 7 Periodic systems and beams

To explore the periodicity of a beam optical system, we first consider a single FODO cell only and follow a single particle with initial coordinates $x = 2$ mm and $x' = 2$ mrad over a large number of turns, as shown in the following script. To speed up the computation (not strictly needed for this simple example) we "compress" the FODO lattice in a single transfer matrix (the so-called one-turn-map).

**Nota Bene**

The 'one-turn-map' provide a very crucial insight of a closed machine (e.g., a synchrotron or a circular collider). In linear optics the one-turn-map is a matrix. By analyzing the matrix properties (e.g., eigenvalues decomposition) we can extract important stability and optics information.

The code used for the compression uses the `getEquivalentElement` function.

```
1 f = 2.5
2 L_2 = 2
3 fodo_lattice = 5*D(L_2/10) + Q(f) + 10*D(L_2/10) + Q(-f) + 5*D(L_2/10)
4 OTM = getEquivalentElement(fodo_lattice)
5 particle = [[1],[.3]]
6 output = transportParticles(particle,100*OTM)
7
8 plt.plot(output['s']/(L_2*2), output['x'][0] ,'o-b')
```

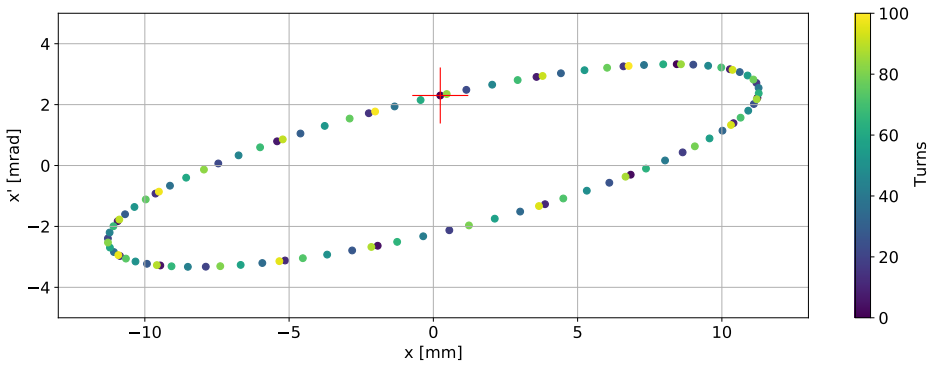


Figure 6: A phase-space plot showing positions $x$ versus angles $x'$ of a particle followed for 100 consecutive turns. The red cross marker indicates the initial position of the particle.

```
plt.xlabel('Turns')
plt.ylabel('x [mm]')
plt.grid(True)
```

Once we got the one-turn-map we can track the particle following the usual approach. It the following code, we used also interactive plots to show the trace-space:

```
from ipywidgets import interactive

f = -2.5
L_2 = 2
fodo_lattice = 5*D(L_2/10) + Q(-f) + 10*D(L_2/10) + Q(f) + 5*D(L_2/10)
OTM = getEquivalentElement(fodo_lattice)
def plotIt(x, xp):
    particle = [[x],[xp]]
    output = transportParticles(particle,100*OTM)
    plt.scatter(output['x'], output['px'], c = output['s']/(L_2*2))
    cb=plt.colorbar()
    cb.set_label('Turns')
    plt.xlabel('x [mm]')
    plt.ylabel("x' [mrad]")
    plt.xlim(-13,13)
    plt.ylim(-5,5)
    plt.grid(True)

interactive_plot = interactive(plotIt,x=(-2,2,.1),xp=(-2,2,.1),continuous_update=True)
output = interactive_plot.children[-1]
output.layout.height = '350px'
interactive_plot
```

The plot, shown in Figure 6, displays an ellipse.

**Exercise 18.** *Explore different initial coordinate and compare the phase-space plots you obtain.*

**Exercise 19.** *Plot the x and x' for the different turns. What do you observe?*

**Exercise 20.** *In the definition of FODO of the previous exercise, reverse the polarity of both quadrupoles and prepare a phase-space plot. How does it differ from the one in Exercise 18?*

**Exercise 21.** *Prepare an array describing a FODO cell that starts immediately following the quadrupole with the negative focal length and prepare the phase-space plot.*

The particle, performing stable oscillations reminiscent of a harmonic oscillator, suggests that we can split the transfer matrix for one turn into a product of three matrices [4]; a rotation matrix, sandwiched between two matrices that distort the coordinates

$$R = \mathcal{A}^{-1}\mathcal{O}\mathcal{A} \quad \text{with} \quad \mathcal{O} = \begin{pmatrix} \cos\mu & \sin\mu \\ -\sin\mu & \cos\mu \end{pmatrix} \quad \text{and} \quad \mathcal{A} = \begin{pmatrix} \frac{1}{\sqrt{\beta}} & 0 \\ \frac{\alpha}{\sqrt{\beta}} & \sqrt{\beta} \end{pmatrix}. \tag{8}$$

Here $\alpha$ and $\beta$ are two, presently undetermined, parameters that describe the distortion. We can, however, determine them from $R$ by comparing coefficients with the result

$$\mu = \arccos\left(\frac{R_{11}+R_{22}}{2}\right), \quad \beta = \frac{R_{12}}{\sin\mu}, \quad \text{and} \quad \alpha = \frac{R_{11}-R_{22}}{2\sin\mu}. \tag{9}$$

These equations are encapsulated in the following function `twiss`

```
def twiss(beamline):
    '''Returns the Q, and the Twiss parameters beta, alpha, gamma of the beamline'''
    OTM = getEquivalentElement(beamline)
    R = OTM[0]['matrix']
    mu = np.arccos(0.5*(R[0,0]+R[1,1]))
    if (R[0,1]<0):
        mu = 2*np.pi-mu;
    Q = mu/(2*np.pi)
    beta = R[0,1]/np.sin(mu)
    alpha = (0.5*(R[0,0]-R[1,1]))/np.sin(mu)
    gamma = (1+alpha**2)/beta
    return Q, beta, alpha, gamma
```

which allows us to determine the parameters $\alpha$, $\beta$, and $\mu = 2\pi Q$ from any transfer matrix $R$ that permits stable oscillations. Note that this is only possible for $|(R_{11}+R_{22})/2| < 1$, which indicates the limit of stability.

**Exercise 22.** *Find the range of focal lengths* `F` *for which the FODO cells permit stable oscillations.*

The parameters returned by `twiss` are commonly called the phase advance $\mu$, the Twiss parameters $\alpha$, $\beta$, and $\gamma = (1+\alpha^2)/\beta$, and $Q$ is referred as the tune.

So, why did we analyze the transfer matrix if we actually want to construct a sigma matrix that is periodic? Because the Twiss parameters make it particularly simple to build a beam matrix `sigma0` that repeats itself after one cell. It is given by

$$\sigma_0 = \varepsilon \begin{pmatrix} \beta & -\alpha \\ -\alpha & \gamma \end{pmatrix}, \tag{10}$$

where we introduced the emittance $\varepsilon$, whose relevance will become obvious in a short while. We can now propagate the matrix $\sigma_0$ through the cell and use Equation 8 to express the transfer matrix $R$ through $\mathcal{A}$ and $\mathcal{O}$. After some algebra, we find

$$R\,\sigma_0\,R^t = \sigma_0\,, \tag{11}$$

which shows that the beam matrix, as defined in Equation 10, reproduces itself after one cell. Let's try out whether this really works in the following code.

```
f = 2.5
L_2 = 2
fodo_lattice = Q(2*f) + 10*D(L_2/10) + Q(-f) + 10*D(L_2/10) + Q(2*f)
OTM = getEquivalentElement(fodo_lattice)
tune, beta, alpha, gamma = twiss(OTM['matrix'])
sigma_0 = np.array([[beta, -alpha],[-alpha, gamma]])

def transportSigmas(sigma, beamline):
    '''Transport the sigma matrix along the beamline'''
    coords = [sigma]
    s = [0]
    for elements in beamline:
        coords.append(elements['matrix'] @ coords[-1] @ elements['matrix'].transpose())
        s.append(s[-1] + elements['length'])
    coords = np.array(coords).transpose()
    return {'sigma11': coords[0][0],
            'sigma12': coords[0][1],
            'sigma21': coords[1][0], # equal to sigma12
            'sigma22': coords[1][1],
            's':  np.array(s),
            'coords': coords,}

output = transportSigmas(sigma_0, fodo_lattice)

plt.plot(output['s'],  output['sigma11'], '-b', lw=3, label='$\\beta_x\ [m]$')
plt.plot(output['s'], -output['sigma12'], '-r', lw=3, label='$\\alpha_x$')

```

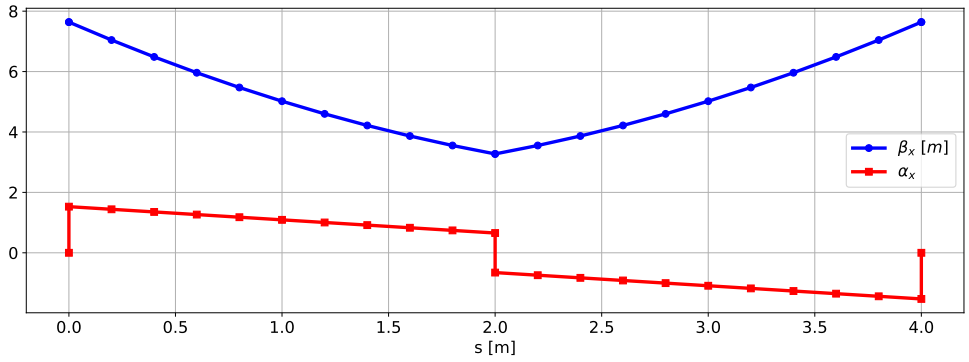


Figure 7: Twiss parameters along the FODO cell.

```
plt.grid(True)
plt.legend(loc='best')

plt.xlabel('s [m]')
```

We also plot the $\beta$ and $\alpha$ function along the FODO lattice (Figure 7).

**Exercise 23.** *Transport the periodic $\sigma$ matrix along the FODO and convince yourself that the $\sigma$ matrix at the end of the FODO is indeed equal to the one at the start.*

**Exercise 24.** *Write down the numerical values of initial beam matrix* `sigma0`*, then build a beamline made of 15 consecutive cells by changing the definition of* `beamline` *and then, using* `sigma0` *with the noted-down numbers, prepare a plot of the beam sizes along the 15 cells. Is it also periodic?*

The emittance $\varepsilon$ that appeared in Equation 10 as an arbitrary constant is actually conserved when propagating the beam matrix, which is easily seen from

$$\det(\sigma) = \det(\tilde{R}\,\sigma_0\,\tilde{R}^t) = \det(\tilde{R})\,\det(\sigma_0)\,\det(\tilde{R}^t) = \det(\sigma_0) = \varepsilon^2 \tag{12}$$

for an arbitrary transfer matrix $\tilde{R}$, because all transfer matrices have unit determinant. This is true for all maps that describe the motion of a beam through static magnetic fields, in particular, for the matrices for drift space and quadrupole and therefore also for products of those elements.

**Exercise 25.** *Verify that the OTM has determinant equal to 1.*

Note also that the matrix with the Twiss parameters $\alpha, \beta$, and $\gamma$, which appears in Equation 10, has unit determinant by construction. The Twiss parameters $\alpha, \beta$, and $\gamma$ describe the relative magnitude of the matrix elements in $\sigma_0$ and the emittance $\varepsilon$ determines the absolute magnitude of the beam. In particular, when considering the $\sigma_{11}$ of a sigma matrix, we see that the absolute beam size is given by $\sigma_x^2 = \sigma_{11} = \varepsilon\beta$. Here the emittance $\varepsilon$ is constant and sets the scale, whereas $\beta(s)$ describes the modulation of the beam size along a beamline from one position $s$ to another.

**Exercise 26.** *Multiply $\sigma_0$ from Exercise 24 by 17 and calculate the emittance. Then propagate the $\sigma$ matrix through the beamline from Exercise 24 and verify that the emittance of the sigma matrix after every element is indeed constant and equal to its initial value.*

Up to now, we have used FODO cells and calculated the phase advance $\mu$ and the Twiss parameters from the transfer matrices. We now reverse the procedure and try to find hardware parameters, such as the focal length of a quadrupole, to achieve certain, desirable values. This procedure, commonly called *matching*, is the topic of the next section.

# 8 Satisfying one's desires: matching

From the analysis of large circular accelerators it is known that an important parameter for the stability of periodically traversed beamlines is the phase advance $\mu$ of a cell, or the tune $Q$, which is the phase advance

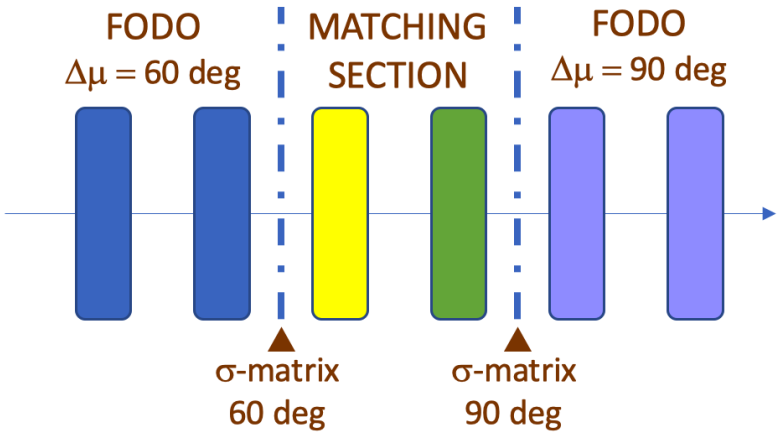


Figure 8: Illustration for Exercise 28, where the strength of the two matching quadrupoles needs to be determined.

for the whole ring, divided by $2\pi$. Let us consider a single cell first and try to adjust the focal length `f` of the quadrupoles to a value, such that the phase advance of the cell is $\mu = 60°$, which corresponds to $Q = \mu/2\pi = 1/6$. The following short script first defines `f` and `fodo_lattice`, then calls `getEquivalentElement` to calculate the transfer matrices, and finally uses it as argument for `twiss()`, which returns the phase advance and the Twiss parameters.

```
f = 2
L_2 = 2
fodo_lattice = Q(2*f) + 10*D(L_2/10) + Q(-f) + 10*D(L_2/10)+Q(2*f)
tune, beta, alpha, gamma = twiss(fodo_lattice)
print(f'The tune is {tune}.')
```

The 'print' statement serves only to display the value of `Q` which is the sought-after phase advance $\mu$ divided by $2\pi$.

**Exercise 27.** *Vary* `f` *by hand and try to find (a) a value that returns* $Q = 1/6$*. (b) Then try to find a value of* `f` *that produces a* $90°$ *phase-advance. What is the corresponding value of* $Q$*?*

This procedure mimics the matching done by the accelerator-design software. There a optimizer minimizes a cost-function that encodes the desired constraints. In Exercise 27 you have to do it by hand.

A common task when designing accelerators is matching one section of a beamline to another one. Here we will assume that the upstream beamline consists of FODO cells with a $60°$ phase advance and the downstream beamline of FODO cells with a $90°$ phase advance. These are the cells with the focal length we calculated in Exercise 27. In between the $60°$ and $90°$, we place a third cell with two quadrupoles that we will use to match the upstream to the downstream beamline. To do so, we need to prepare periodic beam matrices `sigma60` and `sigma90` for the respective sections. Note that `sigma90` only depends on *two* parameters: the Twiss parameters $\alpha$ and $\beta$, and therefore we also need *two* quadrupoles with independently variable focal length to adjust until the final beam matrix equals `sigma90`. See Figure 8 for an illustration of the geometry.

**Exercise 28.** *Implement the procedure described in the previous paragraph and match of a FODO lattice with* $\mu = 60°$ *and* $\mu = 90°$*.*

The discussed matching tasks are only basic examples of the tasks encountered when designing an accelerator. As mentioned before, most software packages have a more or less convenient way to specify the constraints to fulfill and the magnets to vary to satisfy the constraints. Then the software automatically produces the magnet settings.

# 9 ...and beyond...

So far we only considered the very basic optical elements and confined ourselves to a single transverse dimension. During CAS we will build on these basics and extend the software in several directions. In this section, we therefore give some indication of things to come. We start by including quadrupoles that have a finite length. In the introductory beam optics lectures you will learn that the matrix for these elements depends on the polarity of the magnet excitation, as specified by a parameter $k_1 \propto \partial B_y/\partial x$. The matrix is given by

$$Q(l,k_1) = \begin{pmatrix} \cos(\sqrt{k_1}l) & \frac{1}{\sqrt{k_1}}\sin(\sqrt{k_1}l) \\ -\sqrt{k_1}\sin(\sqrt{k_1}l) & \cos(\sqrt{k_1}l) \end{pmatrix} \quad \text{for} \quad k_1 \geq 0 \tag{13}$$

and

$$Q(l,k_1) = \begin{pmatrix} \cosh(\sqrt{|k_1|}l) & \frac{1}{\sqrt{|k_1|}}\sinh(\sqrt{|k_1|}l) \\ \sqrt{|k_1|}\sinh(\sqrt{|k_1|}l) & \cosh(\sqrt{|k_1|}l) \end{pmatrix} \quad \text{for} \quad k_1 < 0 \tag{14}$$

and this brings us directly to

**Exercise 29.** *Introduce as a new element the thick quadrupoles matrices.* **Hint***: write an external function that returns the corresponding list.*

Once the software is able to handle thick quadrupoles, we can replace the thin quadrupoles in the beamline description used earlier.

**Exercise 30.** *Use the beamline from Exercise 27 ($60°$/cell FODO) and replace the thin quadrupoles by long quadrupoles with a length of 0.2, 0.4, 1.0 m. Make sure the overall length and the phase advance of the FODO cell remains unchanged. By how much does the periodic beta function at the start of the cell change? Express the change in percent.*

In the introductory optics lectures you will learn that a so-called sector dipole magnet will exhibit *weak focusing* in the bending plane. The transfer matrix for this magnet will be shown to be

$$B(l,\phi) = \begin{pmatrix} \cos(\phi) & \rho\sin(\phi) \\ -\sin(\phi)/\rho & \cos(\phi) \end{pmatrix} \quad \text{with} \quad \rho = l/\phi\,. \tag{15}$$

**Exercise 31.** *Implement in a Python function the element corresponding to the weak focusing of a sector bend.*

**Exercise 32.** *Insert 1 m long dipoles in the center of the drift spaces of the FODO cells from Exercise 27 while keeping the length of the cell constant. Investigate deflection angles of $\phi = 5°$, $10°$ and $20°$. Check by how much the periodic beta functions change. Why do they change? Explain. Can you compensate the phase advance $\mu$ by adjusting the strength or focal lengths of the quadrupoles?*

All matrices discussed up to now only address one transverse plane and assume that all particles have the same momentum $p_0$.

But this is only approximately true in a real accelerator and we therefore must take small momentum deviations $\delta = (p - p_0)/p_0$ into account. Instead of describing a particle by its transverse coordinates $(x, x')$ alone, we now characterize each particle by $(x, x', \delta)$, which requires to upgrade the transfer matrices to $3 \times 3$ matrices that operate on state vectors $(x, x', \delta)$. The matrix for a drift space therefore turns out to be

$$D(L) = \begin{pmatrix} 1 & L & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{16}$$

with the original $2 \times 2$ matrix from Equation 1 in the top left corner, a $1$ in the bottom right corner and zeros in the other positions of the third row and the third column. The transfer matrices for thin and long quadrupoles are updated likewise. Only the matrix for the sector dipole has non-zero entries in the third column

$$B(l,\phi) = \begin{pmatrix} \cos\phi & \rho\sin\phi & \rho(1-\cos\phi) \\ -\sin(\phi)/\rho & \cos\phi & \sin\phi \\ 0 & 0 & 1 \end{pmatrix}. \tag{17}$$

In order not to mix functions with the same name, but using $3 \times 3$ instead of $2 \times 2$ matrices, name the new elements conveniently or prepare a sub-directory for the upgraded functions.

**Exercise 33.** *Upgrade the software to consistently handle $3 \times 3$ matrices for drift space, quadrupoles, and sector dipoles.*

**Exercise 34.** *Build a beamline of six FODO cells with a phase advance of $60^\circ$/cell (thin quadrupoles are OK to use) and add a sector bending magnet with length 1 m and bending angle $\phi = 10^\circ$ in the center of each drift. You may have to play with the quadrupole values to make the phase advance close to $60^\circ$. But you probably already did this in Exercise 32.*

**Exercise 35.** *Use the starting conditions $(x_0, x_0', \delta) = (0, 0, 0)$ and plot the position along the beamline. Repeat this for $\delta = 10^{-3}$ and for $\delta = 3 \times 10^{-3}$. Plot all three traces in the same graph. Discuss what you observe and explain.*

We also need to upgrade the beam matrix $\sigma$ to be consistent with the $3 \times 3$ transfer matrices. It is then a $3 \times 3$ matrix itself with the original $2 \times 2$ sigma-matrix from Equation 6 in the top-left corner, zeros in the third column and row, except the matrix element in the lower-right corner, which contains the square of the momentum spread $\sigma_p$.

**Exercise 36.** *Work out the transverse components of the periodic beam matrix $\sigma_0$. Assume that the emittance is $\varepsilon_0 = 10^{-6}$ meter-rad. Furthermore, assume that the momentum spread $\sigma_{33} = \sigma_p^2$ is zero and plot the beam size along the beamline.*

**Exercise 37.** *Plot the beam size for for $\sigma_p^2 = 10^{-3}$ and for $\sigma_p^2 = 3 \times 10^{-3}$. What happens if you change the phase advance of the cell? Try out by slightly changing the focal lengths.*

**Exercise 38.** *Determine the periodic dispersion at the start of the cell. Then plot the dispersion in the cell.*

For these exercises you definitely need to consult the slides from the introductory beam optics lectures for background information.

Instead of adding the momentum deviation as additional degree of freedom for the simulation, we can extend the simulation to comprise of both transverse degrees of freedom, the horizontal and the vertical plane. The matrices that describe the dynamics are then $4 \times 4$ matrices with the matrices for the respective planes filling the upper-left and the lower-right $2 \times 2$ sub-matrices. As an illustration, we show the matrix for a focusing ($k_1 > 0$) quadrupole

$$Q(l, k_1) = \begin{pmatrix} Q_F & 0_2 \\ 0_2 & Q_D \end{pmatrix} , \tag{18}$$

where $Q_F$ is the matrix from Equation 13, $Q_D$ from Equation 14, and $0_2$ is the $2 \times 2$ matrix containing only zeros. For $k_1 < 0$ the sub-matrices $Q_F$ and $Q_D$ are exchanged. The matrix for a thin quadrupole is constructed likewise. From the introductory lectures, we know that the matrix for sector dipole only shows weak focusing in the horizontal plane, but behaves as a drift space in the vertical plane. Its matrix is therefore given by

$$B(l, \phi) = \begin{pmatrix} B_2 & 0_2 \\ 0_2 & D_2 \end{pmatrix} , \tag{19}$$

where $B_2$ is the $2 \times 2$ matrix from Equation 15 and $D_2$ the matrix for a drift space from Equation 1. Note that in this framework, we assume that all particles have the reference momentum $p_0$ and we can therefore ignore the momentum deviation $\delta$.

**Exercise 39.** *Convert the code to use $4 \times 4$ matrices, where the third and fourth columns are associated with the vertical plane.*

Using this software to address the following exercises. Note that changing a quadrupole simultaneously affects the phases advances $\mu_x, \mu_y$ and the beta functions in the respective planes.

**Exercise 40.** *Start from a single FODO cell with $60^\circ$/cell you used earlier. Insert sector bending magnets with a bending angle of $\phi = 10^\circ$ in the center of the drift spaces. The bending magnets will spoil the phase advance in one plane. Now you have two phase advances and need to adjust both quadrupoles (by hand to 2 significant figures) such that it really is $60^\circ$ in both planes.*

**Exercise 41.** *Use the result from exercise 2 and adjust the two quadrupoles such that the phase advance in the horizontal plane is 90 degrees, cell, while it remains $60^\circ$/cell in the vertical plane.*

**Exercise 42.** *Prepare a beamline with eight FODO cells without bending magnets and with $60°$/cell phase advance in both planes. (a) Prepare the periodic beam matrix* `sigma0` *(4x4, uncoupled) as the initial beam and plot both beam sizes along the beamline. (b) Use* `sigma0` *as the starting beam, but change the focal length of the second quadrupole by 10 % and plot the beam sizes once again. Discuss your observations.*

**Exercise 43.** *From the lecture about betatron coupling identify the transfer matrix for a solenoid and write a function that receives the longitudinal magnetic field $B_s$ and the length of the solenoid as input and returns the transfer matrix. Then extend the simulation code to handle solenoids. Finally, define a beamline where you place the solenoid in the middle of a FODO cell and follow a particle with initial condition $(x_0, x_0', y_0, y_0') = (10^{-3}\,\mathrm{m}, 0, 0, 0)$. What do you observe? Is the motion confined to the horizontal plane?*

And this brings us to the end of this tutorial. Here, we try to give some idea about what goes on "under the hood" of full-blown beam optics codes, such as MAD-X [10]. Moreover we hope to have provided some guidance on how to write such a simulation code yourself and how to use it to "play around" with simple beam-optical system and get a feeling for their behavior. But the story does not stop here. There are many more topic covered in the CERN Accelerator School and the more advanced textbooks [4, 7, 8, 9].

# References


[1] Python web site: `https://www.python.org`

[2] Octave web site: `https://www.octave.org`

[3] MATLAB web site: `https://www.mathworks.com`

[4] V. Ziemann, *Hands-On Accelerator Physics Using MATLAB,* CRC Press, Boca Raton, 2019.

[5] V. Ziemann, *Beam Optics Primer Using Octave or MATLAB*, `http://arxiv.org/abs/arXiv:1907.10987`.

[6] A. Siegman, *Lasers,* University Science Books, Sausalito, 1986.

[7] A. Wolski, *Beam dynamics in high-energy particle accelerators,* Imperial College Press, London, 2014.

[8] H. Wiedemann, *Particle Accelerator Physics (2 vol.),* Springer Verlag, Heidelberg, 1998.

[9] S.Y. Lee, *Accelerator Physics,* World Scientific, New Jersey, 2004

[10] MAD-X web site: `http://madx.web.cern.ch/madx/`

[11] G. Sterbini, A. Latina, V. Ziemann, *Solutions of the Exercises of the Transverse Linear Beam Dynamics Primer using Python*.


# A Propagating the moments of a distribution

Here we work out how the moments propagate through the beamline as a function of the transfer matrix $R$ that maps particle coordinates from one place to another. If we assume that the downstream coordinates $\hat{x}$ are given by $\vec{\hat{x}} = R\vec{x}$ with $\vec{\hat{x}} = (\hat{x}, \hat{x}')^t = (\hat{x}_1, \hat{x}_2)^t$, and the upstream coordinates by $\vec{x} = (x, x')^t = (x_1, x_2)^t$. Note that here subscript $1$ labels the position and $2$ the angle. The equation that relates $\vec{\hat{x}}$ to $\vec{x}$ can then be written as $\hat{x}_i = \sum_{j=1}^{2} R_{ij} x_j$, which is valid for every particle in the ensemble. We can therefore calculate the averages over the final distribution as

$$\hat{X}_i = \langle \hat{x}_i \rangle = \langle \sum_{j=1}^{2} R_{ij} x_j \rangle = \sum_{j=1}^{2} R_{ij} \langle x_j \rangle = \sum_{j=1}^{2} R_{ij} X_j \ . \tag{20}$$

Here the first equality is simply the definition of $\hat{X}_i$ and the second follows from replacing the particle coordinates $\hat{x}_i$ through the transfer matrix elements $R_{ij}$ and the upstream coordinates $x_j$. Since the same matrix $R$ applies to *all* particles, we can pull it out of the averaging, likewise with the summation over $j$. But then we are left with the averages of $\langle x_j \rangle$, which is the same as $X_j$. This simple calculation proves the

remarkable fact that the *beam centroids propagate in the same way as individual particles* and we can visualize the beam centroid as some sort of super-particle. Note, however, that this feature depends crucially that the propagation is described by a linear operation — a matrix multiplication. If non-linear elements, such as sextupoles are present in the beamline, this correspondence is no longer strictly true.

Now we consider the second moments, but assume that the centroids are zero $X_i = 0$, which simplifies the notation. The sigma-matrix elements $\hat{\sigma}_{ij}$ "on the other end of the transfer matrix $R$" can be calculated in the following way

$$\hat{\sigma}_{ij} = \langle \sum_{k=1}^{2} R_{ik}\, x_k \sum_{l=1}^{2} R_{jl}\, x_l \rangle = \sum_{k=1}^{2} R_{ik} \sum_{l=1}^{2} R_{jl}\, \langle x_k x_l \rangle = \sum_{k=1}^{2} \sum_{l=1}^{2} R_{ik} R_{jl}\, \sigma_{kl}\,, \tag{21}$$

where the first equality follows from inserting $\hat{x}_i = \sum_{j=1}^{2} R_{ij}\, x_j$ in the definition of $\hat{\sigma}_{ij} = \langle \hat{x}_i \hat{x}_j \rangle$. Since the transfer matrices are constant and summing is linear, we can extract both sum and $R$ from the averaging, denoted by the angle brackets and find that the sigma matrix transform according to

$$\hat{\sigma} = R\, \sigma\, R^t\,, \tag{22}$$

where we converted the expression from Equation 21, which is given in components, into a matrix equation. Note that $R^t$ denotes the transpose of the matrix $R$. Equations 20 and 22 are given in Equation 7 in Section 5 in the main part of the text.