Chapter I.5

MAD-X simulation code

Nuria Fuster Martínez
IFIC (CSIC-UV), Valencia, Spain

MAD-X is a general-purpose beam optics code with a long history, aiming to be at the forefront of computational physics in the field of particle accelerator design and simulations. In this chapter, the potential and the limitations of the MAD-X code are discussed along with fundamental and practical steps to get started and fully-fledged examples. The goal of this course is to work on the understanding of transverse beam dynamics concepts in magnetic lattices through a hands-on approach, complementing the Transverse Beam Dynamics course.

I.5.1 Introduction

The comprehensive understanding of how charged particles interact with electromagnetic fields, forms the foundation upon which particle accelerators technology is built. It enables scientists and engineers to design accelerators meeting specific research and application goals and to predict the behavior of particle beams in those accelerators with a high degree of accuracy. General-purpose beam optics codes, such as MAD-X (Methodical Accelerator Design-X) [1], are developed to fulfill these aims. The equation of motion of charged particles is solved under the effect of the external electromagnetic fields generated by the different accelerator components. Then, the transport of the beam properties along the accelerator is performed based on the matrix multiplication formalism and the Lie algebra methods. The explanation of how the physics are implemented in the MAD-X software is beyond the scope of this document. Further information on this topic can be found in [2].

MAD-X was first released in June 2002 and it is compatible with Linux, Mac OS X and Windows operating systems. It is coded in C/C++/Fortran and distributed freely under the CERN copyright [3]. It is a multipurpose code developed mainly for complex applications and large projects, capable of handeling machines with more than 10^4 elements (LEP [4], LHC [5], ILC [6], CLIC [7]...). It is used from early to final stages of accelerator design studies. In particular, it is used to: compute the linear optics functions and beam properties along a defined linear or circular accelerator; design a lattice for obtaining the desired global or local machine properties (matching); simulate accelerator imperfections such as misalignments and magnetic errors to name a few; design correction schemes; and simulate beam dynamics of single particle motion.

In summary, MAD-X is a single particle beam dynamics code used to document the accelerator lattice and compute, simulate and improve its optics and beam parameters. It is worth noting that,

This chapter should be cited as: MAD-X simulation code, N. Fuster Martinez, DOI: 10.23730/CYRSP-2024-003.257, in: Proceedings of the Joint Universities Accelerator School (JUAS): Courses and tutorials, E. Métral (ed.), CERN Yellow Reports: School Proceedings, CERN-2024-003, DOI: 10.23730/CYRSP-2024-003, p. 257.

when multi-particle and multi-bunch simulations are required, MAD-X cannot be used, as well as for simulations requiring non-static machines, i.e., when the beam changes its environment to account for space charge, instabilities and beam-beam effects. However, often, other simulation programs, able to study the mentioned effects, use MAD-X inputs to perform their calculations.

In this chapter, we first provide an introduction to the MAD-X computational tool, widely employed in the design of particle accelerators. Note that a significant part of the capabilities of the code are either briefly discussed or not covered at all. The reader willing to study in more detail the subject can find more material on the official MAD-X web page [1]. In this first part, we also outline the fundamental steps for installation and getting started with the software. Then, in the second part of this chapter, we present a collection of MAD-X tutorials, aiming to complement the JUAS Transverse Beam Dynamics course from a practical perspective. The goal of these tutorials is to work on the understanding of transverse beam dynamics concepts in magnetic lattices through a hands-on approach to gain operational and design experience.

I.5.2 Getting started with MAD-X

MAD-X is an interpreter that accepts and executes statements. The statements defined by the user can be either actions, such as optics functions and matching calculations or the evaluation of expressions, or assignments, including machine elements, beam properties, and variable definitions. It is worth-mentioning, that MAD-X allows regular ($a = b$, if b changes a does not) and deferred assignments ($a := b$, if b changes then a is updated too). The last one becomes crucial when variables are initially defined and subsequently employed within a matching block, as shown in the tutorials in Section 7.

For the statements, MAD-X has its own scripting language, which has a strong resemblance to the C-language but does not require the declaration of the variables' types and is not case-sensitive except for strings enclosed in double quotes (" "). Many of the features of programming languages such as loops and macros can be used, as well as basic functions (exponential, logarithmic, trigonometric functions...), built-in random number generators and predefined constants to be employed in algebraic expressions. It is important to note, that for MAD-X to run successfully, all the statements defined by the user need to end with a semicolon. The statements can occupy any number of input lines and several statements may be placed on the same line.

It is always a good scripting practice to add comments. To do so in MAD-X, you should add two slashes ($//$) or an exclamation mark (!) at the beginning of the line. For multiple lines, you should enclose them with the ($/*$) and ($*/$) sequence of characters.

The statements defining a simulations should comprise the following three primary blocks:

- Description of the machine: definition of the accelerator element's physical attributes and their location in the machine.
- Description of the beam: definition of the beam properties, such as type of particle and energy.
- Actions: definition of the tasks to be executed on a specific machine and particle beam, including optics calculations, optics matching, and tracking of particles.

When setting up the machine description, you should take into account that MAD-X assumes the

horizontal plane to be the bending plane. The different accelerator components are placed and aligned along the reference orbit moving along s ($x = y = 0$ in a curvilinear system), which is the path of a charged particle having the reference momentum. See Fig. I.5.1 for more details. Due to various errors like momentum error, misalignment errors, field errors and intentional orbit corrections, the beam orbit, also known as the closed orbit, does not coincide with the reference orbit. The closed orbit is described with respect to the reference orbit, using the local reference system (x, y, s), and includes all nonlinear effects.

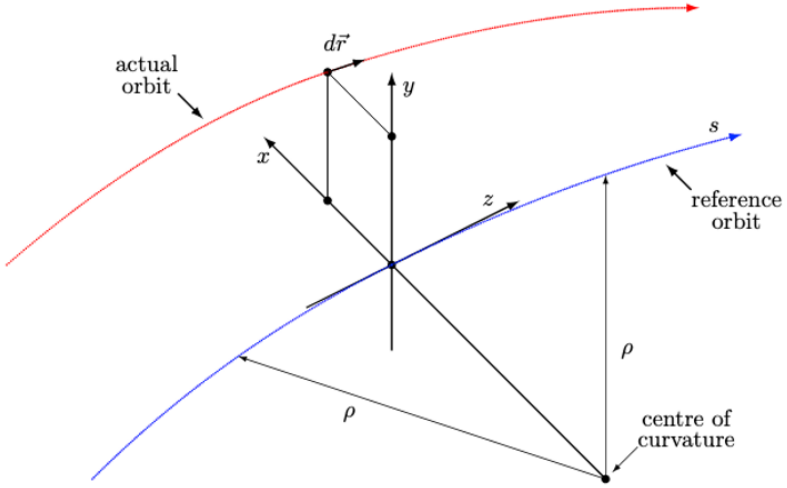

Fig. I.5.1: MAD-X reference system [8].

For the physical quantities, MAD-X employs the "Système International" (SI) units, with the exception of energy, momentum and mass, which are given in GeV, GeV/c and GeV/c^2, respectively. Table I.5.1 gives as reference the names and units in MAD-X of the canonical variables describing the motion of a particle and the linear optics functions.

I.5.3 MAD-X commands

Commands are a type of statement used for communicating with the MAD-X software. They are characterized by a keyword, which is a protected term specifying the intended action to be executed. The user can assign a label to a specific command. When this is done, MAD-X retains the command in memory, allowing it to be used later in the script via the $exec$ control command. Each keyword in MAD-X has associated specific attributes. These attributes can be in the format of strings, logical values, integers, expressions, or range selections. Figure I.5.2 illustrates the generic pattern of a MAD-X command.

I.5.3.1 Lattice elements and sequence

To initiate a study with MAD-X, we first need to define the accelerator components and their respective locations, thereby describing the machine, as illustrated in Fig. I.5.3.

All machine elements must be defined with a command. The label of the command can refer to one element in the accelerator or to a class of elements. In this case, the keyword defines the type of element with specific attributes attached to it, defining its physics behavior. The elements can be

Table I.5.1: Names and units in MAD-X of the canonical variables describing the motion of a particle and the linear optics functions [8].

Description	MAD-X variable	Units
Horizontal and vertical position of the (closed) orbit, referred to the reference orbit, x, y	x, y	[m]
Horizontal and vertical momentum of the (closed) orbit referred to the reference orbit, divided by the reference momentum, $p_{x,y}/p_0$	px, py	[-]
Longitudinal position with respect to the reference particle, $t = -c\Delta t$	t	[m]
Energy error, divided by the reference momentum times the velocity of light, $p_t = \Delta E/p_s c; p_s = p_0(1 + \Delta p/p_0)$	pt	[-]
Arc length, s	s	[m]
Momentum deviation from the design momentum, $\Delta p/p_0$	deltap	[-]
Horizontal and vertical β-function, $\beta_{x,y}$	betx, bety	[m]
Horizontal and vertical α-function, $\alpha_{x,y}$	alfx, alfy	[-]
Horizontal and vertical phase advance, $\mu_{x,y}$	mux, muy	$[2\pi]$
Horizontal and vertical dispersion function, $D_{x,y}$	dx, dy	[m]

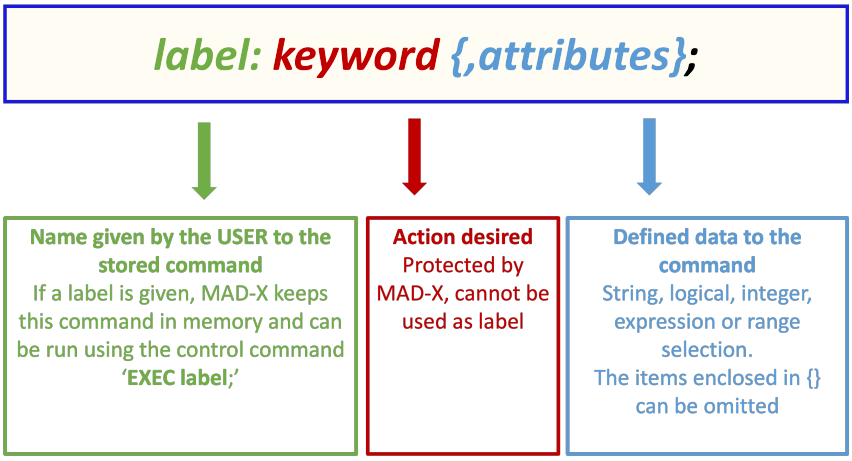

Fig. I.5.2: Generic pattern of a MAD-X command.

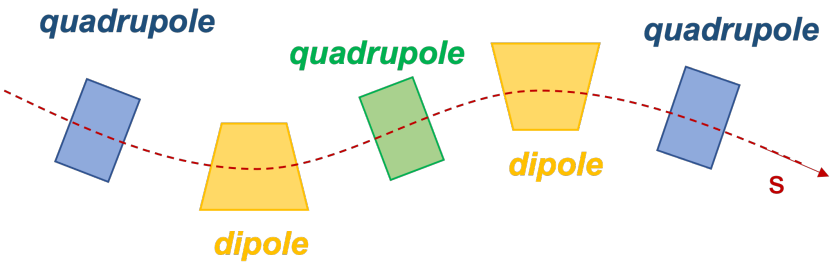

Fig. I.5.3: Accelerator components and sequence illustration.

magnets, markers, RF cavities and collimators, among others. A comprehensive list of all types of elements available and their attributes can be found in the MAD-X manual [8].

Among the various attributes of the different elements available in MAD-X, we want to highlight here the strength of the magnetic elements. This property is a key parameter in the magnetic lattice beam dynamics design. For a dipole magnet, the bending angle must be specified in units of radians, while for higher-order magnets, the normalized magnetic strength relative to the beam rigidity, denoted as k, must be defined. The attribute name for k depends on the type of magnet. The following example illustrates the implementation of a dipole, a quadrupole, and a sextupole magnet, along with their specified magnetic strength and length.

```
mbl: sbend, angle = 0.001, l = 10;
mq: quadrupole, k1 = 0.005, l = 3.3;
msf: sextupole, k2 = 0.0001, l = 1;
```

Note that in the case of a dipole magnet, a positive bend angle represents a bend to the right, i.e. towards negative x values. For the quadrupole, a positive normalized quadrupole strength implies horizontal focusing, irrespective of the charge of the particles. Additionally, octupole magnets are also available, along with thin lens multiple magnet elements that can be defined of any order.

Once the accelerator elements have been defined, the lattice sequence needs to be constructed. A lattice sequence is an ordered collection of machine elements with their position defining the accelerator. The generic block used to define the sequence of an accelerator lattice begins with the keyword $sequence$ and concludes with $endsequence$. An illustrative example is provided below.

```
sequencename: sequence, refer = centre, l = length;
elementname1: elementtype1, at = pos1;
elementname2: elementtype2, at = pos2;
...
endsequence;
```

In the example above, $sequencename$ is the name given by the user to the sequence, and it comprises the defined accelerator elements ($elementname1$, $elementname2$) arranged in the desired order within the sequence. Each element within the sequence is associated with an $elementtype$ corresponding to the previously defined command that specifies the type of element and its associated physical attributes. Additionally, the position of each element within the sequence must be defined. This positioning can be indicated with respect to the $centre$, $exit$, or $entry$ of the element, as well as relative to the sequence start or the position of another element within the sequence. Several examples illustrating how to define the location of elements within a sequence, can be found in the tutorials of Section 7.

Once defined, the sequence needs to be activated in order to be used. In order to do so, the use command has to be run. This command expands the specific sequence inserting drift spaces between elements making it active and usable for other operations.

```
use, sequence = sequencename;
```

For large machines and, especially when multiple users are studying the same lattice, it is useful to define the accelerator sequence in a separate ASCII file. This allows each user to execute the defined sequence for their study using the $call$ command as following: call, file = name;

I.5.3.2 Beam

Another important block concerns the beam definition. The *beam* command and the basic physical attributes to describe the beam (particle type and total beam energy) are the following:

```
beam, particle = proton, energy = 7000;
```

More beam attributes can be found in Ref. [8].

Note that many commands in MAD-X require the prior setting of various quantities related to the beam in the machine. Therefore, MAD-X will stop with a fatal error if an attempt is made to expand a sequence with the *use* command for which no *beam* command has been defined before.

I.5.3.3 Actions

The MAD-X action commands enable us to perform computations on the defined machine for a given beam. In this course, we introduce how to:

- compute the linear lattice functions using the *twiss* command,
- perform design and optimization studies using the *match* command,
- perform beam dynamics studies using the *track* command.

The *twiss* command

To compute the linear lattice functions (optical functions and closed orbit) around the machine and optionally the chromatic functions, the *twiss* command can be used. Below, two examples are presented to show how to use it for a periodic (first) and an initial condition (IC) solution (second) lattice:

```
twiss, sequence = name, centre, file = test.txt, table = name;
twiss, sequence = name, betx = 1, alfx = 0, bety = 1, alfy = 0, centre, file = test.txt, table = name;
```

By default, the optics functions are calculated at the exit of the accelerator element, however the user can modify this by adding the position as an attribute to the *twiss* command. In the examples above the centre position is chosen. Note that in order to compute the linear optics functions for the IC solution lattice, the user needs to add as input the linear optics functions at the entrance of the sequence.

The *twiss* command saves the results of the computation in three outputs: a summary table called *summ* (whose output data cannot be selected by the user) printed in the command line or terminal where the MAD-X executable has been run (see Fig. I.5.4); a table named by default *twiss* with all the calculated parameters; and optionally, an ASCII file with the same content as the *twiss* table. The *twiss* table and the ASCII file names can be specified by the user as illustrated in the examples above. In addition, the *twiss* ASCII file content can be modified by the user using the *select* command as follows:

```
select, flag = twiss, column = name, keyword, s, betx, bety, alfx, alfy, mux, muy;
```

The user can access and print data from the created tables using the *value* command. To get the value of a variable in a table, the attribute to this command should be *table* and, enclosed in brackets, the name of the table and the variables you want to print should be specified as follows: `value, table (summ, dq1, dq2);`. Once you run the *value* command, the value of the selected variable is displayed in the command window. It is also possible to print the value at the location of a given element using the previous command as following: `value, table (twiss, elementname, betx);`.

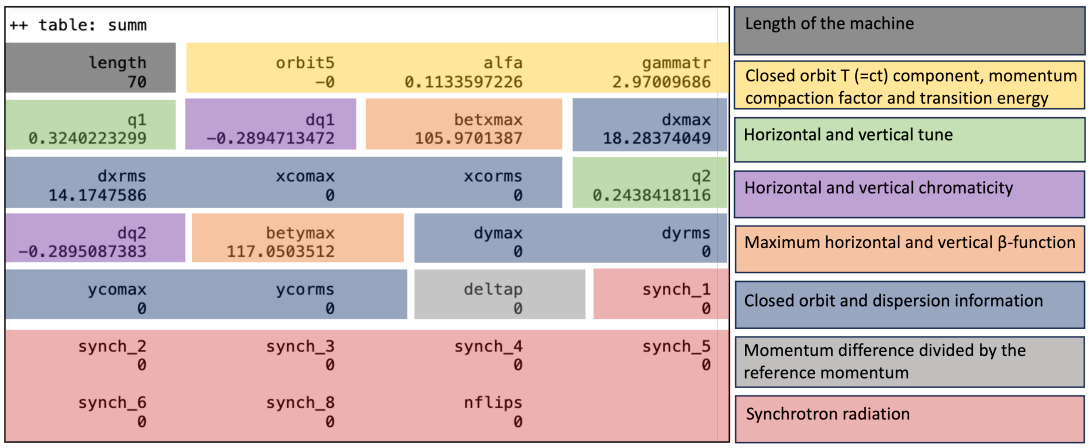

Fig. I.5.4: *summ* table example with the main output data indicated in different colors.

In addition, a graphical output can be produced using the *plot* command as illustrated below.

```
plot, haxis = s, vaxis = betx, bety, color = 100, file = fodo;
```

The above example shows how to plot the horizontal and vertical β-functions as a function of the longitudinal position s, as well as how to save the resulting figure under the name $fodo$.

The *match* command

One of the main applications of general-purpose beam optics codes is to perform matching studies in order to design and optimize accelerator lattices. With a matching routine, one computes the magnetic strength required by the magnets in order to get a desired global or local machine property. Global properties, such as the horizontal and vertical tunes, or the horizontal and vertical chromaticities, are not functions of the s-position along the accelerator. On the contrary, local properties stand for any MAD-X variable or any user defined variable that depends on the s-coordinate and can be adjusted locally. The local matching is usually used to design insertions, or adjust the orbit or the optics locally.

The matching block needs to be started with the *match* command and ended with the *endmatch* command. Inside the matching block the user defines the global or local properties to be matched and the element's attributes to vary, as well as the optimization method to be used. Currently, there are four methods available in MAD-X: $lmdif$, $migrad$, $simplex$ and $jacobian$ [8].

Below, we present an example of global matching. The objective of this example is to match the horizontal ($q1$) and vertical ($q2$) tunes of the machine to 26.51 and 26.62, respectively, by changing the magnetic strengths of the focusing and defocusing quadrupoles, named kqf and kqd, respectively.

```
match, sequence = name;
global, q1 = 26.51;
global, q2 = 26.62;
vary, name = kqf, step = 0.00001;
vary, name = kqd, step = 0.00001;
lmdif, call = 50, tolerance = 1e-6;
endmatch;
```

In the above example, the $lmdif$ minimization method has been used, for which $call$ indicates the maximum number of iterations in the optimization process and $tolerance$ the tolerance of the penalty function.

Next, we provide an example of local matching where the objective is to attain a horizontal β-function of 50 m and an α-function of 2 at the exit of the lattice. As before, these goals are to be met by adjusting the magnetic strengths of the focusing and defocusing quadrupoles, named kqf and kqd, respectively.

match, sequence = name;

constraint, range = $\#e$, betx = 50;

constraint, range = $\#e$, alfx = 2;

vary, name = kqf, step = 0.00001;

vary, name = kqd, step = 0.00001;

lmdif, call = 50, tolerance = 1e-6;

endmatch;

The $track$ command

Last, the $track$ command is introduced. This command is used to perform single particle tracking studies in thin lens lattices. In order to make the lattice compatible with the module, the $makethin$ command has to be used as makethin, sequence = name; before the $track$ block is executed. This command substitutes all the magnets by the number of selected thin lenses and drift spaces. In order to select the number of slices, the $select$ command has to be used before the $makethin$ command as follows: select, flag = makethin, slice = 5;

If tracking studies with thick elements are of interest, the Polymorphic Tracking Code (PTC) module can be used [9]. However, the use of this module is beyond the scope of this course.

The tracking block begins with the $track$ command and concludes with the $endtrack$ command. Within the tracking block, the user must specify the initial coordinates of each particle to be tracked, as well as the number of turns. Below we provide an example.

track, dump, file = name, deltap = 0.01;

start, x = 1e-3, px = 0, y = 1e-3, py = 0;

start, x = 1e-1, px = 0, y = 1e-1, py = 0;

observe, place = string;

run, turns = 100;

endtrack;

In the example above, the tracking is carried out for two particles. The first particle has, at the entrance of the machine, a horizontal and vertical amplitude excursion of 1 mm while the second particle, has a horizontal and vertical amplitude excursion of 100 mm. Both particles have initially a $deltap$ (momentum difference divided by the reference momentum) of 0.01. The tracking is performed, in this case, for 100 turns indicated in the last line as an attribute for the run command.

To save the particle's coordinates turn by turn, the attribute $dump$ must be included in the first

line of the $track$ block. Then, the particle's trajectory data at the start of the accelerator is saved, both in a MAD-X table and in an ASCII file for each particle tracked. Additional observable points can be defined using the $observe$ command before the run command is executed inside the $track$ block. If this is implemented, additional MAD-X tables and ASCII files are created with the particle's trajectory data at the locations specified by the attribute $place$ of the $observe$ command. The output files are named automatically. The name given by the user (attribute $file$ of the $track$ command) is followed by $.obsnnnn$, where $nnnn$ is the observation point number, and also followed by $.pnnnn$ where $nnnn$ is now the particle number. Hence, the default filename for the first observation point and first particle looks like $file.obs0001.p0001$.

Furthermore, the command $plot$ can also be used here to create a graphical output with the tracking data as follows:

```
plot, file = tracking, table = track, haxis = x, vaxis = px, colour = 100;
```

The above example shows how to plot the horizontal phase space coordinates for both particles and save the figure under the name $tracking$.

I.5.4 How to run MAD-X

First of all, you need to install the MAD-X software. In order to do that, you need to download the latest release from the repository and follow the instructions provided in Ref. [1]. After the installation, you have two options for running MAD-X. You can run it interactively or in batch mode. The first approach is done by running the MAD-X executable and then typing and running the MAD-X commands one after the other at the command line. To run in batch mode, the user executes a pre-defined input ASCII file containing all the MAD-X commands describing the simulation to be performed. Considering the batch approach, the typical three-steps MAD-X workflow is illustrated in Fig. I.5.5.

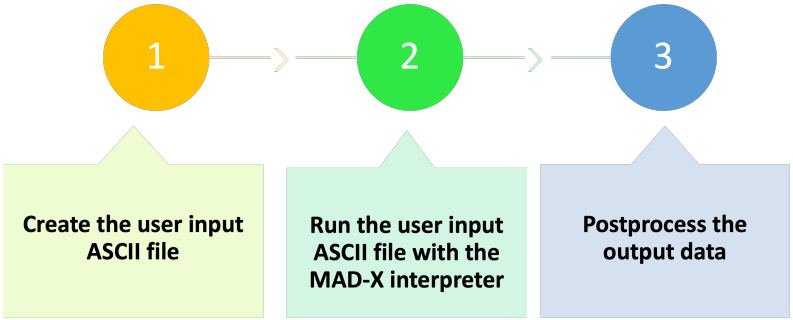

Fig. I.5.5: Typical MAD-X workflow.

If you're running on a Windows machine, the pre-defined input ASCII file is executed after running the MAD-X executable using the $call$ command, like this: `call, file = myfile.txt;`. While in Linux or MAC OS, you need to run on a terminal the MAD-X executable followed by the name of the input ASCII file as follows: $./madx\ myfile.madx$.

Note, that MAD-X offers actions for basic plotting but lacks a graphical user interface. Because of that, the post-processing is generally carried out using other programming languages such as Python [10], Matlab [11], Root [12], or Gnuplot [13].

I.5.5 MAD-X Python interface

Instead of following a pure MAD-X approach, one can use the code through a Python [10] interface. This approach was adopted for the JUAS MAD-X course since the 2020 edition.

Python is a high-level, interpreted, general-purpose programming language widely used in the physics community, which provides powerful numerical libraries and plotting routines for the analysis. You can find several quality Python courses, videos and resources on the internet [14, 15]. A Python library called $Cpymad$ [16] is being developed and maintained since 2014 and allows us to use the MAD-X software through Python.

In order to use the Pythonic approach, the essential steps to set up the environment properly are provided below. To install Python, we recommend using the Anaconda distribution, which can be downloaded from here [17]. The installation procedure depends on the operating system. We suggest following the official documentation for Windows, Linux, or Mac OS as appropriate. In addition, in order to run successfully the tutorials presented in Section 7, the following Python packages need to be installed: $Numpy$ [18], $Matplotlib$ [19], $Cpymad$ [16] and $Pandas$ [20].

Most MAD-X commands have their corresponding method in the $Cpymad$ library. All the available methods can be found in [16]. However, for the purpose of this course, only a few $Cpymad$ methods are used. This is done in order to not overload the reader with too many new commands, and to focus the learning on the MAD-X ones while using Python to visualize and analyze the output data. The $Cpymad$ library methods used in the solutions presented in Section 7 are the $input()$ method to communicate with the active MAD-X process, and the $table.tablename.dframe()$ method to save MAD-X output data from various tables into a $Pandas$ dataframe. Additionally, during the in-person MAD-X JUAS course, the $call()$ method is used. This method enables the execution of a pre-defined ASCII file containing MAD-X commands directly from the Python interface.

In the following, we provide the code snippets required to run MAD-X within a Python interface i.e., using $Jupyter$ [21].
The first step is to import the $Cpymad$ library:

```
from cpymad.madx import Madx
```

Secondly, we have to create an instance of the $Madx$ object, which initializes and manages a MAD-X process in the background. This object allows us to establish communication with the running process and issue commands to it.

```
myMad = Madx()
```

Then, we have to define the MAD-X input statements defining our study as Python strings.

```
myString= '''MAD-X statements;'''
```

Finally, we run the defined strings with the MAD-X commands in the running MAD-X process using the $Cpymad$ library $input$ method:

```
myMad.input(myString)
```

If a $twiss$ or $track$ command is executed one can save the resulting output data in a $Pandas$ dataframe [20]. $Pandas$ is a Python library designed to facilitate data manipulation and visualization

through its powerful data structures. To do this, you can first check the available tables and their corresponding names as follows:

```
print(list(madx.table))
["summ", "twiss","track.obs0001.p0001"]
```

And then, the *Pandas* dataframes are built as follows:

```
myMad.table.twiss.dframe()
myMad.table.summ.dframe()
myMad.table[track.obs0001.p0001].dframe()
```

As mentioned earlier, it is also possible to define MAD-X commands in an ASCII file (*filename* in the example below) and run this file within the active MAD-X process using the *call* method, as shown below.

```
myMad.call(filename)
```

I.5.6 Tutorials

The JUAS MAD-X workshop is comprised of an introductory lecture on MAD-X and six tutorials. During each tutorial, the students have dedicated time to work on the proposed questions, receiving guidance from tutors. At the conclusion of each session, the solutions are discussed. In this section, we introduce the tutorials while the solutions are presented in Section 7.

I.5.6.1 Tutorial 1: My first accelerator, a FODO cell

The main goal of this tutorial is to learn how to define a simple magnetic lattice and compute the linear optics functions using MAD-X. For that, we are going to define a FODO lattice, which is the simplest configuration we can design to get a net focusing effect of the beam in both transverse planes.

Questions:

1. Define a FODO lattice (see Fig. I.5.6) with the following characteristics:
   - Length of the cell, L_{cell} = 100 m,
   - Quadrupoles length, L_q = 5 m,
   - The first quadrupole is placed at the start of the sequence,
   - Quadrupoles focal length, f_q = 200 m.
2. Define a proton beam with a total energy, E_{tot}, of 2 GeV. Activate the sequence and compute the periodic linear optics functions with the *twiss* MAD-X command. Then, plot the β-functions. If you found β_{max} = 463.6 m, you succeeded!
3. Using the β-function plot obtained, can you estimate the phase advance of the FODO cell? How does this value compare to the tune computed by MAD-X?
4. Try to run the *twiss* command with E_{tot} = 0.7 GeV. What is the MAD-X error message? And if you change the focal length to 20 m?

The solutions to the questions of this tutorial are in Section 7.1.

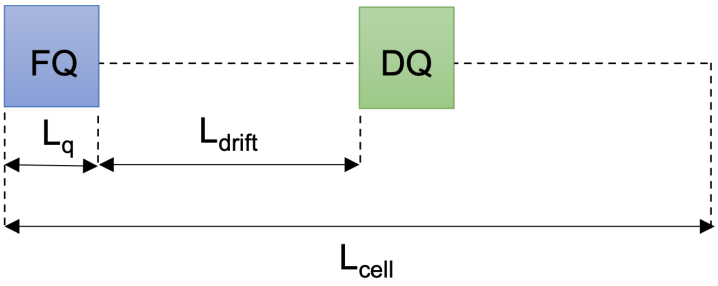

Fig. I.5.6: FODO cell illustration with the main parameters depicted.

I.5.6.2 Tutorial 2: My first matching

The main goal of this tutorial is to study the behavior of the linear optics functions when varying the FODO cell magnetic properties. To do so, we use the linear thin lens optics solution and the $twiss$ MAD-X module. The results of the two approaches are compared and discussed.

By considering the periodic solution of the equation of motion for a symmetric FODO cell, and applying the thin lens approximation and the stability conditions, we can derive the following relationships between the optical parameters $\Delta\mu$, β_{max} and β_{min} and the magnets and cell specifications K, L_q and L_{cell}:

$$\frac{\Delta\mu}{\pi} = \frac{2\arcsin(\frac{KL_{cell}L_q}{4})}{\pi}, \tag{I.5.1}$$

$$\frac{\beta_{min}}{L_{cell}} = \frac{-\frac{KL_{cell}L_q}{4}+1}{\sin(2\arcsin(\frac{KL_{cell}L_q}{4}))}, \tag{I.5.2}$$

$$\frac{\beta_{max}}{L_{cell}} = \frac{\frac{KL_{cell}L_q}{4}+1}{\sin(2\arcsin(\frac{KL_{cell}L_q}{4}))}, \tag{I.5.3}$$

where $\Delta\mu$ is the phase advance, β_{min} and β_{max} correspond to the minimum and maximum β-functions, respectively, K is the strength of both quadrupoles, L_q is the length of the quadrupoles and L_{cell} is the total length of the FODO cell. Figure I.5.7 shows the thin lens approximation phase advance and extreme β-functions for a FODO cell as a function of K, L_{cell} and L_q, computed using Eqs.(I.5.1), (I.5.2) and (I.5.3).

Questions:

1. Using the thin lens approximation solution from Fig. I.5.7 (left) compute the required strength to power the quadrupoles to obtain a $\Delta\mu$ of approximately 90° in the FODO cell. Then, power the quadrupoles of the FODO cell from Tutorial 1, with the computed strength values and compute the linear optics functions with MAD-X. What is the phase advance computed by MAD-X?
2. What is the β_{max} computed by MAD-X? Compare the obtained values with the thin lens approximation solution from Fig. I.5.7 (right).
3. Reduce by half the focusing strength of the quadrupoles. What is the effect on the β_{max}, β_{min} and

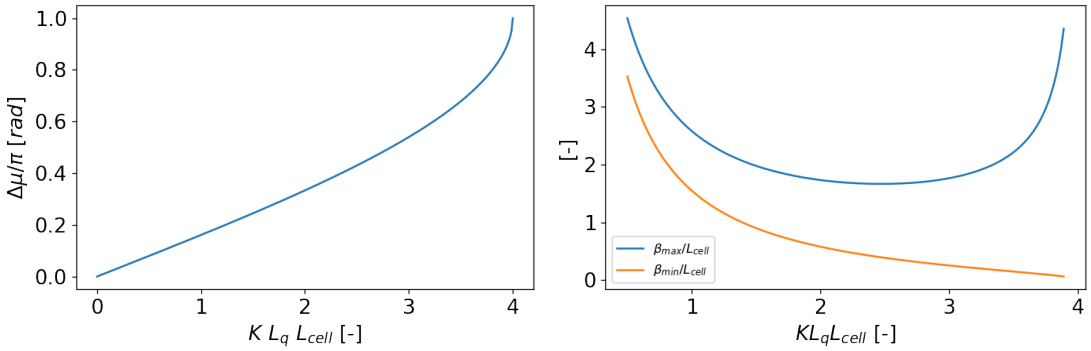

Fig. I.5.7: Symmetric FODO cell thin lens approximation solution phase advance (left) and extreme β-functions (right) as a function of K, L_q and L_{cell}.

$\Delta\mu$? Compare the obtained values with the values from Fig. I.5.7.

4. Compute the maximum beam size $\sigma_{x,y}$ assuming a normalized emittance, $\epsilon_n^{x,y}$, of 3 mrad·mm and a total beam energy of E_{tot} = 7 TeV. Use the relation between the beam size and the β-function:

$$\sigma_{x,y} = \sqrt{\frac{\beta_{x,y}\epsilon_n^{x,y}}{\gamma}}, \tag{I.5.4}$$

where γ is the relativistic factor.

The solutions to the questions of this tutorial are in Section 7.2.

I.5.6.3 Tutorial 3: Building a circular machine

The main goal of this tutorial is to install dipole magnets in the FODO cell designed in question 1 Tutorial 2 to build a circular machine, as well as to study the impact of the dipole magnets on the linear optics functions. In addition, the MAD-X matching module is used to adjust the quadrupoles' strength to achieve a desired tune of the machine. The tune, defined as the phase advance normalized by 2π, is a crucial parameter in the design of a circular machine for getting the desired beam quality and stability.

Questions:

1. Consider the FODO cell designed in question 1 Tutorial 2 and add 4 sector dipoles of 15 m length, L_d, assuming a drift space between the magnets as illustrated in Fig. I.5.8. For computing the required bending angle, consider a ring with 736 dipoles with equal bending angles.
2. Using the $twiss$ command compute the linear optics functions. Do the dipoles (weak focusing) affect the maximum and minimum β-functions? What about the dispersion?
3. From the phase advance of the FODO cell compute the horizontal and vertical tunes of the machine.
4. Using the $match$ command on a single FODO cell, match the tunes of the machine to 46.0 in both planes.
5. If we change the beam energy to 7 TeV, what are the new tunes of the machine? Why?

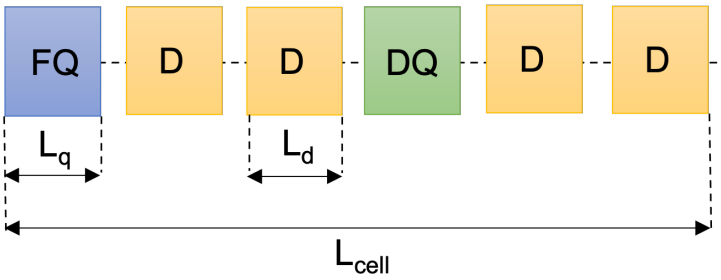

Fig. I.5.8: FODO cell scheme with 4 sector dipoles.

6. What is the maximum tune that you can reach with such lattice? Hint: what is the maximum phase advance per FODO cell in the thin lens approximation?

   The solutions to the questions of this tutorial are in Section 7.3.

I.5.6.4 Tutorial 4: Natural chromaticity

The main objective of this tutorial is to study the impact of the natural chromaticity of a FODO cell on the particle beam dynamics by means of particle tracking studies. Figure I.5.9 illustrates the concept of chromaticity in a quadrupole magnet. Orange and blue lines correspond to off-momentum particles and the green line represents an on-momentum particle. In this illustration, we observe a spread in the focusing effect of the quadrupole, which is caused by the energy spread of the beam, known as chromaticity.

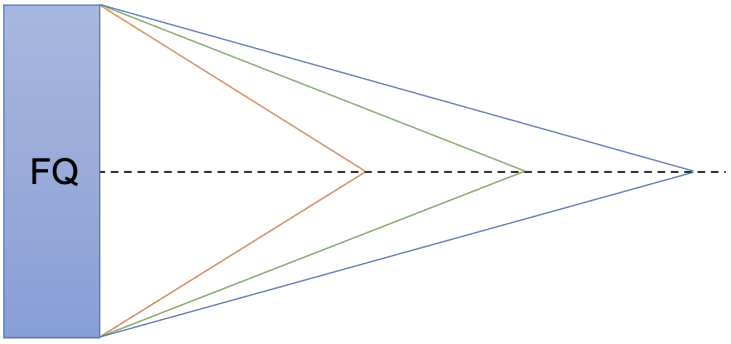

Fig. I.5.9: Concept of chromaticity illustration.

For this tutorial, the thin lens version of the lattice designed in Tutorial 3 is used as starting point together with a 7 TeV total energy proton beam. Note that the *track* module only works with thin lens lattices. To do the proper conversion of the lattice, the *makethin* command is used. Additionally, after running the *makethin* command, it is necessary to perform a rematch of the lattice to ensure that the horizontal and vertical tunes of the FODO cell remain at 0.25.

Questions:

1. Using the chromaticities computed using the *twiss* command, compute the tunes for off-

momentum particles with $\Delta p/p$= 10^{-3}, using the following equation:

$$\Delta Q = dq \times \frac{\Delta p}{p}. \tag{I.5.5}$$

2. Track two particles, one with initial coordinates of (x, y, px, py) = (1 mm, 1 mm, 0, 0) ($particle1$) and another one with initial coordinates of (x, y, px, py) = (100 mm, 100 mm, 0, 0) ($particle2$) for 100 turns. Plot the horizontal and vertical phase space coordinates, $x-px$ and $y-py$, respectively. How do the particles move in the phase space turn after turn? Do you see the tunes? Do you see any difference between the two particles? It may help to look only at the first 4 turns to get a clear picture.
3. Repeat the tracking exercise, but now for two off-momentum particles by adding a $\Delta p/p$ = 10^{-2} to the initial particles' conditions. How does the phase space look now? Is the tune still the same?

The solutions to the questions of this tutorial are in Section 7.4.

I.5.6.5 Tutorial 5: Non-linearities

The main objective of this tutorial is to install sextupole magnets in the FODO cell from Tutorial 4 to correct for the natural chromaticity as well as to study the impact of the sextupole magnets on the beam dynamics of the particles.

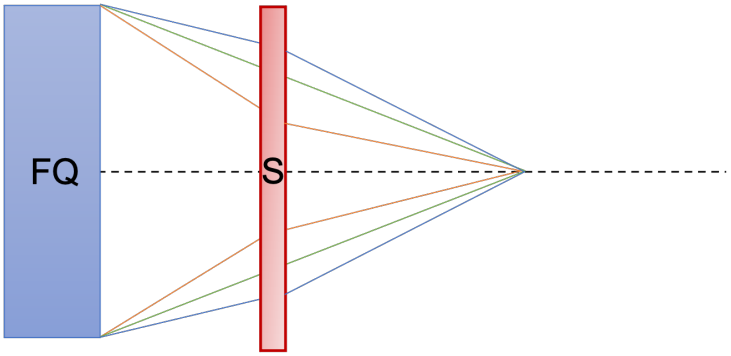

Fig. I.5.10: Chromaticity correction with sextupole magnets concept illustration.

Questions:

1. Install two 0.5 m long sextupole magnets attached to the two quadrupoles. Then, with a MAD - X matching block adjust the vertical and the horizontal chromaticity of the FODO cell (global parameters: dq1 and dq2) to zero, by powering the two sextupoles ($k2_1$ and $k2_2$).
2. Using the strength of the sextupoles, $k2_1$ and $k2_2$ and the linear optics functions (β-function and dispersion) at the sextupoles' location, evaluate the sextupoles' contribution to the chromaticity on the horizontal plane using the following equation:

$$\zeta_x = \frac{1}{4\pi}(\beta_{x,s1} k2_1 D_{x,s1} + \beta_{x,s2} k2_2 D_{x,s2}). \tag{I.5.6}$$

Then, compare the obtained value with the chromaticity value obtained in Tutorial 4.

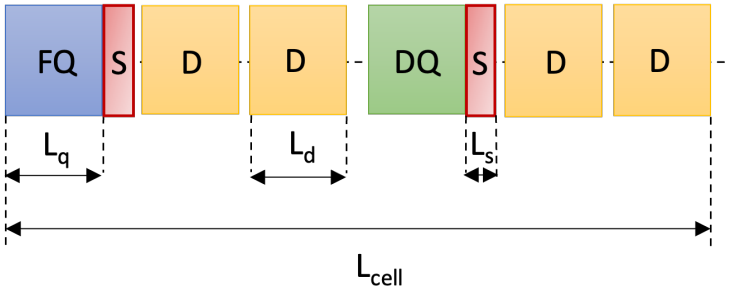

Fig. I.5.11: FODO lattice with dipole and sextupole magnets.

3. Track two particles, one with initial coordinates of (x, y, px, py) = (1 mm, 1 mm, 0, 0) ($particle1$) and another one with initial coordinates of (x, y, px, py) = (100 mm, 100 mm, 0, 0) ($particle2$), both with $\Delta p/p$ = 10^{-2}, for 100 turns. Plot the horizontal and vertical phase space coordinates, $x - px$ and $y - py$, respectively. Do you see the tunes? Did you manage to recover the original tunes for the off-momentum particles? What is going on?
4. Move the tunes to (0.23, 0.23) and repeat the tracking exercise described in question 3. Are the particles stable?

The solutions to the questions of this tutorial are in Section 7.5.

I.5.6.6 Tutorial 6: Building a transfer line

The main objective of this tutorial is to design a transfer line and match the linear optics functions at the end of the line to some desired values. Matching studies for different initial conditions are performed and the results are discussed.

Questions:

1. Build a transfer line for a 2 GeV proton beam of 10 m total length, L_{tot}, with 4 quadrupoles of 0.4 m length, L_q, and 0.1 m^{-1} of magnetic strength. Place the quadrupoles centered at 2, 4, 6 and 8 m. What is the error message that you get if you try to find a periodic solution? Why?

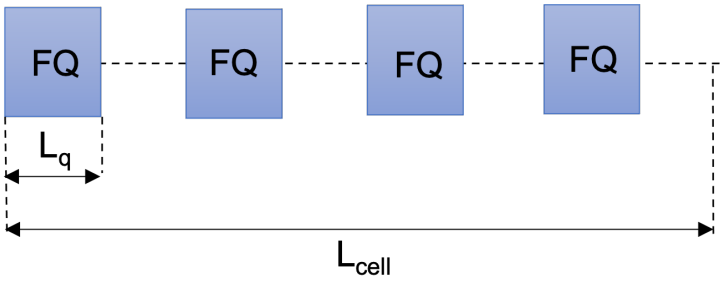

Fig. I.5.12: Transfer line with four focusing quadrupoles scheme.

2. Calculate the linear optics functions for the transfer line assuming (β_x, α_x, β_y, α_y) = (1 m, 0, 2 m, 0) at the start of it. What are the linear optics functions (β_x^{end1}, α_x^{end1}, β_y^{end1}, α_y^{end1}) at the end of the transfer line?

3. Starting from (β_x, α_x, β_y, α_y) = (1 m, 0, 2 m, 0), match the transfer line to get the following optics (β_x^{end2}, α_x^{end2}, β_y^{end2}, α_y^{end2})= (2 m, 0, 1 m, 0) at the end.
4. Now, starting from (β_x, α_x, β_y, α_y) = (1 m, 0, 2 m, 0), and using the quadrupoles' strength computed in question 3, match the transfer line to get the (β_x^{end1}, α_x^{end1}, β_y^{end1}, α_y^{end1}) obtained in question 2. Can you find back the initial quadrupoles' strength from question 1 of 0.1 m^{-2}?
5. Consider that the quadrupoles have an excitation current factor of 10 A·m^2, an excitation magnetic factor of 2 T/(m·A) and an aperture of 40 mm diameter. Compute the magnetic fields at the poles of the quadrupoles for the two matching solutions obtained in question 2 and 4, assuming a linear regime and using a dimensional approach.

The solutions to the questions of this tutorial are in Section 7.6.

I.5.7 Solutions

In this section, the solutions to the tutorials proposed in Section 6, implemented using Python as the programming language, are presented. To execute the code snippets provided, you should use Python or a Python interface like Jupyter [21]. If you prefer to use a pure MAD-X approach, you need to create an input ASCII file containing the MAD-X statements defined as strings alongside this section and follow the instructions detailed in Section 4 to run it in MAD-X.

To run all the tutorials presented in this section using the Python approach, you should start by loading the relevant Python libraries and instantiating the MAD-X process. Here are the code lines that show how to do this:

```
# Loading the libraries of interest
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from cpymad.madx import Madx
# Initialization of the MAD-X process
myMad = Madx()
```

Please ensure that you execute the code lines defined above for all the tutorials before proceeding with the solutions presented along this section. Bare in mind that if the MAD-X process stops due to an error, you need to initialize the MAD-X process again.

I.5.7.1 Tutorial 1: My first accelerator, a FODO cell

Question 1. The string containing the MAD-X statements and commands required to define the FODO cell parameters, magnets, and sequence is presented below.

```
myString= '''
!************************************************
!Definition of the main parameters              **
!************************************************
l_cell = 100;
quadrupolelength = 5;
f = 200;
```

```
myk := 1/f/quadrupolelength;
!*********************************************
!Definition of the magnets                    **
!*********************************************
qf: quadrupole, l = quadrupolelength, k1 := myk;
qd: quadrupole, l = quadrupolelength, k1 := -myk;
!*********************************************
!Definition of the sequence                    **
!*********************************************
myCell: sequence, refer = entry, l = l_cell;
quadrupole1: qf, at = 0;
marker1: marker, at = 25;
quadrupole2: qd, at = 50;
marker2: marker, at = 75;
endsequence;'''
```

Then, the *input* method from the Python library *Cpymad* needs to be used to run the predefined string with the FODO cell set-up, within the background MAD-X process, as illustrated below:

```
myMad.input(myString)
```

If MAD-X does not produce any error as output, it indicates that the input statements have been defined correctly, and there are no syntax or logical errors in the defined list of commands. This means that MAD-X has successfully processed the input statements, and you can proceed with further actions.

Question 2. A similar procedure must be followed to establish the beam parameters and activate the sequence using the following code lines:

```
myString='''
!*****************************************
!Definition of the beam                   **
!*****************************************
beam, particle = proton, energy = 2;
!*****************************************
!Activation of the sequence               **
!*****************************************
use, sequence = myCell;'''
myMad.input(myString)
```

Then, to calculate the linear optics functions, the following commands need to be run:

```
myString='''
!*****************************************
!MAD-X Actions                            **
!*****************************************
select, flag = twiss, column = ["name", "keyword", "betx", "bety"];
twiss, file = MyfirstFODO.madx;'''
myMad.input(myString)
```

After running the *twiss* command, the linear optics functions of the FODO lattice are calculated. Two tables (by default, *summ* and *twiss*) containing the output data are generated. If the computation is done satisfactory, the summary output table, shown in Fig. I.5.13 with the parameters defined in Fig. I.5.4, should be displayed in the command window.

```
  ++++++++++++++++++++++++++++++++++++++++++++
  +     MAD-X 5.09.00  (64 bit, Darwin)      +
  + Support: mad@cern.ch, http://cern.ch/mad +
  + Release   date: 2023.05.05               +
  + Execution date: 2024.03.28 12:19:35      +
  ++++++++++++++++++++++++++++++++++++++++++++
•••
enter Twiss module

iteration:   1 error:   0.000000E+00 deltap:   0.000000E+00
orbit:   0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00

++++++ table: summ

            length             orbit5               alfa            gammatr
               100                 -0                  0                  0

                q1                dq1            betxmax              dxmax
     0.03853349451     -0.04384718845        463.6232883                  0

             dxrms             xcomax             xcorms                 q2
                 0                  0                  0      0.03853349451

               dq2            betymax              dymax              dyrms
    -0.04384718845        463.6232883                  0                  0

            ycomax             ycorms             deltap            synch_1
                 0                  0                  0                  0

           synch_2            synch_3            synch_4            synch_5
                 0                  0                  0                  0

           synch_6            synch_8             nflips              dqmin
                 0                  0                  0                  0

       dqmin_phase
      0.7706482429
```

Fig. I.5.13: Tutorial 1 MAD-X summary table.

In order to visualize the data from the $summ$ table, an example is given below using both, the MAD-X (with the $value$ command) and the Pythonic approach (with the $Cpymad$ library method $myMad.table.[tablename].dframe()$). In this example, the horizontal tune and the maximum horizontal β-function are printed.

```
# MAD-X approach:
myString='''
value, table(summ, q1);
value, table(summ, betymax);'''
myMad.input(myString);
# Output
# table (summ q1) = 0.03853349451;
# table (summ betymax ) = 463.6232883;
# Pythonic approach
# First, the names of the tables generated can be printed using the code below
print(list(myMad.table))
# Output
# ["summ", "twiss"]
```

```
# Then, the Pandas dataframes can be created and the data desired selected
myDF = myMad.table.summ.dframe()
myDF["q1"]
myDF["betymax"]
# Output
# 0.038533;
# 463.62;
```

If you found that the β_{max} is 463.6 m, you succeeded!

The data from the $twiss$ table can be stored in a $Pandas$ dataframe (named below as $myDFAll$), and the desired columns can be selected and printed in a different table (named below as $myDF$ and shown in Fig. I.5.14) as follows:

```
myDFAll = myMad.table.twiss.dframe()
myDF = myDFAll[["name", "keyword", "s", "betx", "bety", "alfx", "alfy",
                "mux", "muy", "dx", "dy", "x", "y"]]
myDF
```

	name	keyword	s	betx	bety	alfx	alfy	mux	muy	dx	dy	x	y
#s	mycell$start:1	marker	0.0	463.623288	369.779162	-1.156109	0.929316	0.000000	0.000000	0.0	0.0	0.0	0.0
quadrupole1	quadrupole1:1	quadrupole	5.0	463.623288	369.779162	1.156109	-0.929316	0.001709	0.002161	0.0	0.0	0.0	0.0
drift_0[0]	drift_0:0	drift	25.0	419.394867	408.967742	1.055312	-1.030113	0.008930	0.010350	0.0	0.0	0.0	0.0
marker1	marker1:1	marker	25.0	419.394867	408.967742	1.055312	-1.030113	0.008930	0.010350	0.0	0.0	0.0	0.0
drift_1[0]	drift_1:0	drift	50.0	369.779162	463.623288	0.929316	-1.156109	0.019041	0.019493	0.0	0.0	0.0	0.0
quadrupole2	quadrupole2:1	quadrupole	55.0	369.779162	463.623288	-0.929316	1.156109	0.021202	0.021202	0.0	0.0	0.0	0.0
drift_2[0]	drift_2:0	drift	75.0	408.967742	419.394867	-1.030113	1.055312	0.029391	0.028423	0.0	0.0	0.0	0.0
marker2	marker2:1	marker	75.0	408.967742	419.394867	-1.030113	1.055312	0.029391	0.028423	0.0	0.0	0.0	0.0
drift_3[0]	drift_3:0	drift	100.0	463.623288	369.779162	-1.156109	0.929316	0.038533	0.038533	0.0	0.0	0.0	0.0
#e	mycell$end:1	marker	100.0	463.623288	369.779162	-1.156109	0.929316	0.038533	0.038533	0.0	0.0	0.0	0.0

Fig. I.5.14: $twiss$ output table printed as a $Pandas$ dataframe table.

Next, you can plot the β-functions and the horizontal dispersion as functions of the position s, using either the $matplotlib$ library and your own script or using the source code provided in Appendix 1.A. In order to use the function in Appendix 1.A you need to save the script given as a Python script and run it as follows:

```
import sys
sys.path.append('path to the location where the script is saved')
import 'name of the script' as lib
lib.plot_layout(myDFAll)
```

Note that the $plot\_layout$ function needs as input the linear optics functions and the strength of the magnets in order to run successfully. This function is employed all along this section to plot the linear optics functions.

In Fig. I.5.15, the horizontal β-function (in blue), the vertical β-function (in red), and the horizontal dispersion D_x (in green) are displayed as a function of the position s, along with an overview of the

lattice layout in the top figure, including the strength of the magnets.

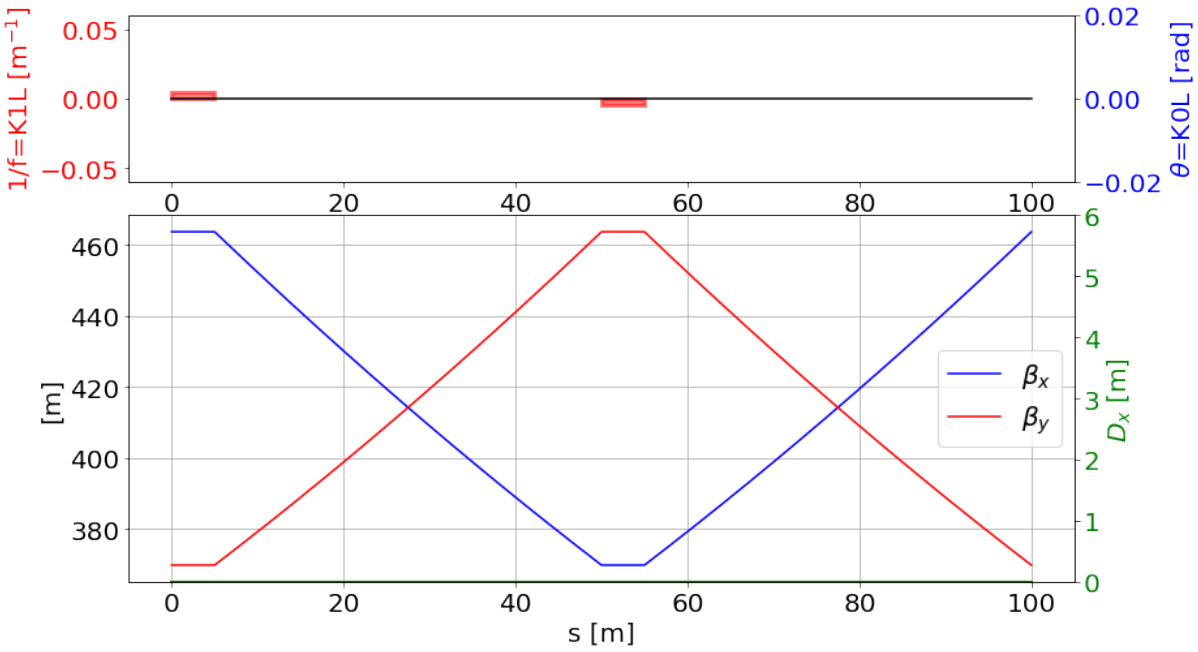

Fig. I.5.15: Horizontal (blue) and vertical (red) β-functions and horizontal dispersion (green) as a function of the position s along with an overview of the lattice layout in the top figure including the strength of the magnets. Note that for an ideal lattice without errors and no dipoles, no dispersion is expected.

Question 3. Now, using Fig. I.5.15, we can estimate the horizontal and vertical phase advance of the FODO cell and compare them with the values obtained by MAD-X. Regarding the phase advance, denoted as $\mu_{x,y}$, we can consider the following definition:

$$\mu_{x,y} = \int \frac{1}{\beta_{x,y}(s)} ds. \tag{I.5.7}$$

A straightforward approximation can be made by assuming a constant β-function and calculating the mean β-function value in the FODO cell. It is worth noting, that the phase advance in MAD-X is provided in units of 2π. Taking all of these considerations, the resulting horizontal phase advance using Eq. (I.5.7) is:

```
(1/((np.max(myDF["betx"])+np.min(myDF["betx"]))/2))*100/2/np.pi
# Output
# 0.03819401853309916
```

Note that in the code lines above, the max and min functions of the $Numpy$ Python's library are used.

Another approach to compute the phase advance, consists in evaluating the integral in Eq. (I.5.7) using the method $trapz$ from the Python library $NumPy$. This method performs integration along a specified axis using the composite trapezoidal rule.

```
np.trapz(1/myDF["betx"],myDF["s"])/2/np.pi
# Output
# 0.038571104937361426
```

Both calculations can be compared with the MAD-X computed value and printed as follows:

```
myDF.iloc[-1]["mux"]
# Output
# 0.03853349450910022
```

As observed, there is a strong agreement up to the third decimal digit for the first approximation and up to the fourth decimal digit for the second approximation.

Due to the symmetry of the FODO cell under study, the same values obtained for the horizontal plane are also obtained for the vertical plane.

Question 4. In the following, we rerun the $twiss$ method, but this time for a beam with E_{tot} = 0.7 GeV.

```
myString='''
beam, particle = proton, energy = 0.7;'''
myMad.input(myString);
```

When the above string is executed, a MAD-X error is shown in the command window because the beam energy defined is lower than the proton rest mass and the total energy must be given.

As a consequence of the above error, the MAD-X process running in the background stops. To proceed with the tutorial, you need to re-instantiate the MAD-X object and run the FODO lattice defined at the beginning of the tutorial again. Then, you can change the quadrupole focal length to 20 m and compute the linear optics functions using the following commands:

```
myString='''
f = 20;
myK := 1 / f / quadrupolelength;
beam, particle = proton, energy = 2;
use, sequence = myCell;
twiss, sequence = myCell;'''
myMad.input(myString);
```

Once again, you encounter an error, this time due to the instability of the cell. In this case, the $twiss$ module fails to find a periodic stable solution because the quadrupole's focal length is too short, and it does not satisfy the stability condition.

I.5.7.2 Tutorial 2: My first matching

Question 1. From Fig. I.5.7 one can determine that, in order to achieve a phase advance of 90° in a FODO cell, the product $(KL_{cell}l_q)$ should be approximately 2.8. Using this value, you can calculate the necessary magnetic strength of the Tutorial 1 FODO cell quadrupole magnets to get a 90° phase advance. The string containing the new machine parameters, beam and actions to be run is presented below.

```
myString='''
!*********************************************
!Definition of the main parameters            **
!*********************************************
l_cell = 100;
quadrupolelength = 5;
myk := 2.8 / l_cell / quadrupolelength;
!*********************************************
```

```
!Definition of the magnets                          **
!************************************************
qf: quadrupole, l = quadrupolelength, k1 := myk;
qd: quadrupole, l = quadrupolelength, k1 := -myk;
!************************************************
!Definition of the sequence                         **
!************************************************
myCell: sequence, refer = entry, l = l_cell;
quadrupole1: qf, at = 0;
marker1: marker, at = 25;
quadrupole2: qd, at = 50;
marker2: marker, at = 75;
endsequence;
!************************************************
!Definition of beam                                 **
!************************************************
beam, particle = proton, energy = 2;
!************************************************
!Activation of the sequence                         **
!************************************************
use, sequence = myCell;
!************************************************
!Twiss                                              **
!************************************************
twiss, file=MyfirstFODO.madx;'''
myMad.input(myString)
```

From this point onward, we will exclusively follow the Pythonic approach to analyze the results. The following code lines show how to retrieve the phase advance values computed by the *twiss* command used before.

```
myDFSumm = myMad.table.summ.dframe()
myDFTwiss = myMad.table.twiss.dframe()
myDFTwiss[["name","s","betx","bety","alfx","alfy","mux","muy"]]
# Phase advance in radians
myDFSumm["q1"]*2*np.pi
myDFSumm["q2"]*2*np.pi
# Output
# 1.485174
# 1.485174
# Phase advance in degrees
myDFSumm["q1"]*2*np.pi*180/np.pi
myDFSumm["q2"]*2*np.pi*180/np.pi
# Output
# 85.094226
# 85.094226
```

The phase advance computed by MAD-X is 85° in both planes.

Question 2. Now, to obtain the value of β_{max}, we can calculate the maximum value from the corresponding column in the β-function *twiss* table (see Fig. I.5.16) as follows:

```
myDFTwiss["betx"].max()
# Output
# 160.60365457633446
myDFTwiss["bety"].max()
# Output
# 160.60365457633446
```

The β_{max} from the thin lens approximation from Fig. I.5.7 (right) and for a ($KL_{cell}l_q$) value of 2.8, is approximately 169.7 m in both planes. The relative variation obtained between the two calculations is about 5%.

	name	keyword	betx	bety	alfx	alfy	mux	muy
#s	mycell$start:1	marker	160.603655	34.217492	-2.259847	0.548735	0.000000	0.000000
quadrupole1	quadrupole1:1	quadrupole	160.603655	34.217492	2.259847	-0.548735	0.004841	0.023892
drift_0[0]	drift_0:0	drift	85.419675	71.376768	1.499352	-1.309229	0.032151	0.090204
marker1	marker1:1	marker	85.419675	71.376768	1.499352	-1.309229	0.032151	0.090204
drift_1[0]	drift_1:0	drift	34.217492	160.603655	0.548735	-2.259847	0.108661	0.127712
quadrupole2	quadrupole2:1	quadrupole	34.217492	160.603655	-0.548735	2.259847	0.132553	0.132553
drift_2[0]	drift_2:0	drift	71.376768	85.419675	-1.309229	1.499352	0.198864	0.159863
marker2	marker2:1	marker	71.376768	85.419675	-1.309229	1.499352	0.198864	0.159863
drift_3[0]	drift_3:0	drift	160.603655	34.217492	-2.259847	0.548735	0.236373	0.236373
#e	mycell$end:1	marker	160.603655	34.217492	-2.259847	0.548735	0.236373	0.236373

Fig. I.5.16: Tutorial 2 main $twiss$ output data printed as a $Pandas$ dataframe table.

Question 3. Next, we repeat question 2, but this time for a lattice with the quadrupoles' strength reduced by half:

```
myString='''
myk := 1.4/l_cell/quadrupolelength;
twiss, file = firstTwiss.txt;'''
myMad.input(myString);
```

The tune and the values of the maximum and minimum β-functions can be visualized using the Pythonic approach as follows:

```
myDFTable_half = myMad.table.twiss.dframe()
myDFTable_half
print("horizontal bmax:")
display(myDFTable_half["betx"].max())
print("vertical bmax:")
display(myDFTable_half["bety"].max())
print("horizontal bmin:")
display(myDFTable_half["betx"].min())
print("vertical bmin:")
display(myDFTable_half["bety"].min())
print("q1:")
display(myDFTable_half["mux"].max())
```

```
print("q2:")
display(myDFTable_half["muy"].max())
# Output
# Horizontal bmax:205.46040125139584
# Vertical bmax:205.46040125139584
# Horizontal bmin:106.48386027996813
# Vertical bmin:106.48386027996813
# q1:0.10979220062366979
# q2:0.10979220062366979
```

To be compared with the previous values:

```
# Output
# Horizontal bmax:160.60365457633446
# Vertical bmax:160.60365457633446
# Horizontal bmin:34.21749204847468
# Vertical bmin:34.21749204847468
# q1:0.23637284979737802
# q2:0.23637284979737802
```

From this exercise we can conclude that when we reduce the strength of the quadrupoles, effectively focusing less, both the β_{max} and β_{min} increase, resulting in larger beam sizes. Conversely, the tune is reduced. Using the thin lens approximation from Fig. I.5.7 and considering a ($KL_{cell}l_q$) value of 1.4, we obtain a β_{max} value of 204.2 m. In this case, the relative variation between the thin and thick lens computations is approximately 0.6%. A closer agreement is observed between the thin and thick lens calculations compared to the results obtained in question 2. This is attributed to the fact that as we move towards the left on the horizontal axis of Fig. I.5.7 (resulting in smaller k for fixed L_q and L_{cell} values), the condition for the thin lens approximation is better satisfied.

Question 4. Finally, the beam size can be computed using Eq. (I.5.4):

```
emittance_n  = 3e-6 # units of m rad
beta_gamma = 7000/.938
np.sqrt(myDFTwiss["betx"].max()*emittance_n/beta_gamma)
# Horizontal
# 0.0002540918517774357
np.sqrt(myDFTwiss["bety"].max()*emittance_n/beta_gamma)
# Vertical
# 0.0002540918517774357
```

I.5.7.3 Tutorial 3: Designing a circular machine

Question 1. In this tutorial, we are adding dipole magnets to the FODO cell designed in question 1 from Tutorial 2, in order to use it to build a circular machine. To determine the required strength of the dipoles, we must first calculate the necessary bending angle to achieve a closed circular orbit with the proposed 736 dipoles using the following condition:

$$2\pi = N\theta, \tag{I.5.8}$$

where N is the total number of dipoles and θ the bending angle of each dipole magnet.

Using Eq. (I.5.8), the needed bending angle in units of radians is:

```
2*np.pi/736
# Output
# 0.008536936558667916
```

Below, the string containing the new lattice configuration with the four sector dipoles installed, the beam parameters, and the actions to be performed is shown.

```
myString='''
!************************************************
!Definition of parameters                       **
!************************************************
l_cell = 100;
quadrupolelength = 5;
dipolelength = 15;
nBend = 736;
myAngle = 2*pi / nBend;
mykf := 2.8 / l_cell / quadrupolelength ;
mykd := 2.8 / l_cell / quadrupolelength ;
!************************************************
!Definition of magnets                          **
!************************************************
qf: quadrupole, l = quadrupolelength, k1 := mykf;
qd: quadrupole, l = quadrupolelength, k1 := -mykd;
bm: sbend, l = dipolelength, angle := myAngle;
!************************************************
!Definition of sequence                         **
!************************************************
myCell: sequence, refer = entry, l = l_cell;
q1: qf, at = 0;
b1: bm, at = 5 + quadrupolelength/2, from=q1;
b2: bm, at = 5 + dipolelength/2, from=b1;
q2: qd, at = l_cell/2;
b3: bm, at = 5 + quadrupolelength/2, from = q2;
b4: bm, at = 5 + dipolelength/2, from = b3;
endsequence;
!************************************************
!Definition of beam                             **
!************************************************
beam, particle = proton, energy = 2;
!************************************************
!Activation of the sequence                     **
!************************************************
use, sequence = myCell;
!************************************************
!Twiss                                          **
!************************************************
twiss, file = MyfirstFODOwithDipoles.madx;'''
myMad.input(myString)
```

Note that in this case, when defining the sequence, the position of the FODO lattice elements have

been defined with respect to the previous element in the sequence.

Question 2. Next, we analyze the impact of the dipoles on the linear optics functions. To do so, the data from the *summ* and *twiss* tables, obtained from the first *twiss*, are stored in two *Pandas* dataframes containing the columns of interest, as follows:

```
first_summ_with_dipoles = myMad.table.summ.dframe()
first_twiss_with_dipoles = myMad.table.twiss.dframe()
first_twiss_with_dipoles[["name", "keyword", "s", "x", "y","px","py",
                          "betx", "alfx", "mux", "bety", "alfy", "muy",
                          "dx", "dy", "dpx", "dpy"]]
```

In Fig. I.5.17, the resulting horizontal (blue) and vertical (red) β-functions, as well as the horizontal dispersion, D_x, as functions of s, are depicted including the machine layout on the top figure with the magnets' strength. The plot is generated using the *plot_layout* function from Appendix 1.A.

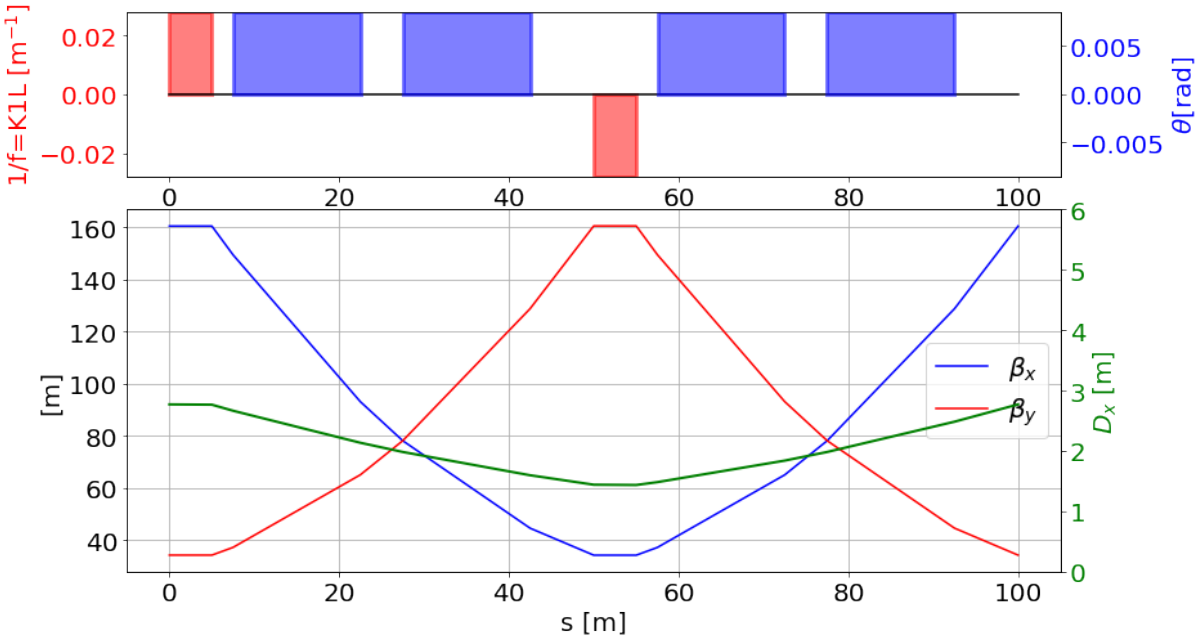

Fig. I.5.17: Horizontal (blue) and vertical (red) β-functions and horizontal dispersion, D_x, as a function of s including the machine layout on the top figure and the strength of the magnets.

Next, we repeat the linear optics calculation but without the dipole magnets by setting the dipoles' strength to 0. The new data is saved in a new *Pandas* dataframes.

```
myString='''
myAngle = 0;
use, sequence = myCell;
twiss;'''
myMad.input(myString);
first_summ_without_dipoles = myMad.table.summ.dframe()
first_twiss_without_dipoles = myMad.table.twiss.dframe()
first_twiss_without_dipoles[["name", "keyword", "s", "x","y",
                             "px","py", "betx", "alfx", "mux",
                             "bety", "alfy", "muy", "dx", "dy", "dpx", "dpy"]]
```

Now, we create a *Pandas* dataframe combining the data from both calculations in order to compare the two scenarios studied. The code lines to perform this task are provided below and Fig. I.5.18 illustrates the resulting output table with the maximum values of the linear optics functions along the lattice with and without dipoles.

```
comparison = pd.DataFrame({'With dipoles':first_twiss_with_dipoles.max(),
                           'Without dipoles':first_twiss_without_dipoles.max()})
comparison.loc[["betx", "alfx", "mux", "bety", "alfy", "muy",
                "dx","dy","dpx","dpy"]]
```

	With dipoles	Without dipoles
betx	160.548167	160.603655
alfx	2.258972	2.259847
mux	0.236501	0.236373
bety	160.603655	160.603655
alfy	2.259847	2.259847
muy	0.236373	0.236373
dx	2.770935	0.0
dy	0.0	0.0
dpx	0.038252	0.0
dpy	0.0	0.0

Fig. I.5.18: Maximum values of the linear optics functions in the FODO lattice with and without dipoles. The effect of the dipoles can be observed on the horizontal parameters.

The effect of the dipole magnets on the horizontal and vertical β_{max} can be quantified by computing the relative variation between the two cases, using the code lines presented below.

```
betx_rel=(np.max(first_twiss_without_dipoles.betx)-np.max(
    first_twiss_with_dipoles.betx))/np.max(first_twiss_without_dipoles.betx)
print ("On the horizontal plane [%]")
print(betx_rel*100)
bety_rel=(np.max(first_twiss_without_dipoles.bety)-np.max(
    first_twiss_with_dipoles.bety))/np.max(first_twiss_without_dipoles.bety)
print("On the vertical plane [%]")
print(bety_rel*100)
# Output
# Relative variation on the horizontal plane [%]: 0.03495607663009094
# Relative variation on the vertical plane [%]: 0.0
```

From this tutorial we can conclude, that the dipole magnets affect the horizontal linear optics functions, as well as the closed orbit. They introduce horizontal dispersion in the lattice, causing particles with different momentum to follow betatron oscillations around a different closed orbit. The small effect

observed on the horizontal β-function is known as weak focusing, and it was used in the first low-energy circular accelerators to keep the particles confined in them, before the strong focusing concept with quadrupoles was developed.

Question 3. The horizontal and the vertical tunes of the machine can now be computed using the phase advance values obtained in the previous question, as follows:

```
Ncells=736/4
HorTune=Ncells*first_twiss_with_dipoles["mux"].max()
display(HorTune)
VerTune=Ncells*first_twiss_with_dipoles["muy"].max()
display(VerTune)
# Output
# Horizontal tune: 43.516126037479125
# Vertical tune: 43.49260436271754
```

Question 4. In the last part of this tutorial, the *match* module is used in order to adjust the tunes of the machine to 46.0 in both planes. Given that our machine consists of 184 FODO cells, it is necessary to adjust the tunes of a single FODO cell to 0.25. The sequence of actions required to accomplish this task is provided below.

```
# The phase advance of one FODO cell is:
Ncells=736/4
np.array([46.0, 46.0])/Ncells
# Output
# array([0.25, 0.25])
myString='''
!************************************************************
!We first switch on the dipole magnets                       **
!************************************************************
nBend = 736;
myAngle = 2*pi / nBend;
use, sequence = myCell;
!************************************************************
!Twiss before matching                                       **
!************************************************************
twiss, table = beforematching, file = BeforeMatching.txt;
!************************************************************
!Matching                                                    **
!************************************************************
phaseWantedX = 46.0/(736/4);
phaseWantedY = 46.0/(736/4);
match, sequence = myCell;
global, q1 = phaseWantedX;//H-tune
global, q2 = phaseWantedY;//V-tune
vary, name = mykf, step = 0.00001;
vary, name = mykd, step = 0.00001;
lmdif, calls = 50, tolerance = 1e-6;
endmatch;
!************************************************************
!Twiss after matching                                        **
!************************************************************
```

```
twiss,table = aftermatching, file = AfterMatching.txt;
!*********************************************************
!Final tune values                                       **
!*********************************************************
value, table(summ,q1)*(736/4);
value, table(summ,q2)*(736/4);'''
myMad.input(myString)
```

If the matching block runs successfully you should get a matching summary table as the one in Fig. I.5.19.

```
MATCH SUMMARY

Node_Name                 Constraint   Type  Target Value       Final Value        Penalty
-------------------------------------------------------------------------------------------
Global constraint:        q1           4     2.50000000E-01     2.50000123E-01     1.51549100E-12
Global constraint:        q2           4     2.50000000E-01     2.50000123E-01     1.51676380E-12


Final Penalty Function =   3.03225480e-12



Variable                Final Value  Initial Value Lower Limit  Upper Limit
-------------------------------------------------------------------------------
mykf                     5.85410e-03  5.60000e-03 -1.00000e+20  1.00000e+20
mykd                     5.85570e-03  5.60000e-03 -1.00000e+20  1.00000e+20

END MATCH SUMMARY
```

Fig. I.5.19: Tutorial 3 MAD-X matching summary table.

Again, the *summ* and *twiss* tables can be stored as *Pandas* dataframes as follows:

```
twiss_after_matching = myMad.table["aftermatching"].dframe()
summary_after_matching  =myMad.table["summ"].dframe()
# To print out the horizontal tune
display(summary_after_matching["q1"])
# Output: 0.25
# To print out the vertical tune
display(summary_after_matching["q2"])
# Output: 0.25
```

Question 5. Now, we can explore the impact of changing the beam energy to 7 TeV on the tune. First we run the *twiss* calculation for the new beam using the code lines below.

```
myString='''
!*********************************************************
!Definition of beam                                      **
!*********************************************************
beam, particle = proton, energy = 7000;
!*********************************************************
!Twiss                                                   **
!*********************************************************
twiss, sequence=myCell;'''
myMad.input(myString)
```

```
twiss_after_matching_7TeV = myMad.table['twiss'].dframe()
summary_after_matching_7TeV = myMad.table['summ'].dframe()
```

Then, we can display the tunes for each beam energy:

```
# For 2 GeV
display(Ncells*twiss_after_matching["mux"].max())
display(Ncells*twiss_after_matching["muy"].max())
# Output
# 46.00002265137154
# 46.000022660881534
# For 7 TeV
display(Ncells*twiss_after_matching_7TeV["mux"].max())
display(Ncells*twiss_after_matching_7TeV["muy"].max())
# Output
# 46.00002265137154
# 46.000022660881534
```

As illustrated in this tutorial, the tunes obtained are the same for the two beam energies. This is because in MAD-X the strength of the quadrupoles, k, is normalized to the beam rigidity and therefore the beam energy change has no impact on the β-functions and on the phase advance. However, be careful with the definition of the chromaticity ($dq1$ and $dq2$) and all the momentum derivative quantities in MAD-X, which are derivatives with respect to the longitudinal variable PT. Since $PT \approx \beta_{\text{rel}} \times \Delta p/p_0$ (see derivation in chapter 39 of Ref. [8]), where β_{rel} is the relativistic Lorentz factor, those functions given by MAD-X must be multiplied by β_{rel} a number of times equal to the order of the derivative, to find the functions given in the literature. See below the chromaticity for the two cases studied:

```
# For 2 GeV
display(summary_after_matching.iloc[0][["dq1","dq2"]])
# Output
# dq1    -0.359777
# dq2    -0.360008
# For 7 TeV
display(summary_after_matching_7TeV.iloc[0][["dq1","dq2"]])
# Output
# dq1    -0.317728
# dq2    -0.317932
```

Question 6. Finally, we can compute the maximum achievable tune in units of 2π of the full machine, taking into account the fact that the maximum phase advance for a FODO cell is 180 degrees (see Fig. I.5.7 left):

```
Ncells*.5
# Output
# 92.0
```

I.5.7.4 Tutorial 4: Natural chromaticity

The goal of this tutorial is to investigate the impact of the natural chromaticity of the FODO cell designed in Tutorial 3 for a 7 TeV proton beams by means of tracking studies. Note that in order to run the

track command, a thin lens lattice must be provided. To create a thin version of the lattice designed in Tutorial 3, the elements in the sequence must be centered (the reference location must be the position corresponding to the middle of the element) and the *makethin* command must be used. In order to keep the horizontal and vertical tunes to 0.25, the matching block must be run again for the thin lens lattice created after the *makethin* command is executed.

Below, the string with all the MAD-X statements defining the lattice, the beam and the actions described in the previous paragraph, is given.

```
myString='''
!*********************************************************
!Definition of parameters                                **
!*********************************************************
l_cell =  100;
quadrupolelength = 5;
dipoleLength = 15;
nBend = 736;
myAngle = 2*pi / nBend;
mykf = 2.8 / l_cell /quadrupolelength;
mykd = 2.8 / l_cell /quadrupolelength;
!*********************************************************
!Definition of magnets                                   **
!*********************************************************
qf: quadrupole, l = quadrupolelength, k1 := mykf;
qd: quadrupole, l = quadrupolelength, k1 := -mykd;
bm: sbend, l = dipolelength, angle := myAngle;
!*********************************************************
!Definition of sequence                                  **
!*********************************************************
myCell: sequence, refer = centre, l = l_cell;
q1: qf, at = 0 + quadrupolelength/2;
b1: bm, at = 5 + quadrupolelength/2 + dipolelength/2, from = q1;
b2: bm, at = 5 + dipolelength/2 + dipolelength/2, from = b1;
q2: qd, at = l_cell/2 + quadrupolelength/2;
b3: bm, at = 5 + quadrupolelength/2 + dipolelength/2, from = q2;
b4: bm, at = 5 + dipolelength/2 + dipolelength/2, from = b3;
endsequence;
!*********************************************************
!Definition of beam                                      **
!*********************************************************
beam, particle = proton, energy = 7000;
!*********************************************************
!Activation of the sequence                              **
!*********************************************************
use, sequence = myCell;
!*********************************************************
!Twiss                                                   **
!*********************************************************
twiss, file = MyfirstFODOwithDipoles.txt;
!*********************************************************
!Makethin                                                **
```

```
!*********************************************************************
select, flag = makethin, slice = 5;
makethin, sequence = myCell;
use, sequence = myCell;
!*********************************************************************
!Twiss before matching                                              **
!*********************************************************************
twiss, table = beforematching, file = BeforeMatching.txt;
!*********************************************************************
!Matching                                                           **
!*********************************************************************
match, sequence = myCell;
global, q1 = 0.25;
global, q2 = 0.25;
vary, name = mykf, step = 0.00001;
vary, name = mykd, step = 0.00001;
lmdif, calls = 50, tolerance = 1e-6;
endmatch;
!*********************************************************************
!Twiss after matching                                               **
!*********************************************************************
twiss, file = AfterMatching.txt;
!*********************************************************************
!Final tune values                                                  **
!*********************************************************************
value, table(summ,q1)*(nBend/4);
value, table(summ,q2)*(nBend/4);
value, table(summ,dq1);
value, table(summ,dq2);
'''
myMad.input(myString)
```

Question 1. Now, using the chromaticity value, dq, obtained, the variation on the tune, ΔQ, for an off-momentum particle with $\Delta p/p = 10^{-3}$ can be computed using Eq. (I.5.5). Remember that the momentum derivative functions in MAD-X are normalized by β_{rel}. However for this case, we could assume an ultra-relativistic beam and therefore $\beta_{\text{rel}} = 1$.

```
# Cpymad method in order to get the relativistic factor
beta_rel = myMad.sequence["mycell"].beam.beta
print (beta_rel)
# Output
# 0.9999999910167906
# Horizontal tune of the machine
tune_onmomentum = myMad.table["summ"].q1[0]
print ("On-momentum tune")
print(tune_onmomentum)
# Variation on the tune for the off-momentum particle using Eq.1.5
tune_onmomentum*beta_rel*1e-3
# -0.0003177280054266581
# Horizontal tune for the off-momentum particle
tune_offmomentum = myMad.table["summ"].q1[0]-0.00031
```

```
print("Off-momentum tune")
print(tune_offmomentum)
# Output
# Horizontal on-momentum tune: 0.25000012325703125
# Horizontal off-momentum tune: 0.24969012325703124
```

A similar result is obtained in the vertical plane by changing $q1$ by $q2$ in the above code lines.

Question 2. The impact of the detuning effect computed in question 1 on the beam dynamics of two particles with different initial conditions is illustrated in this exercise by means of tracking studies using the $track$ command. Two particles, $particle1$ and $particle2$, with initial horizontal and vertical amplitudes of 1 and 100 mm, respectively, are tracked along the lattice using the code lines presented below.

```
myString='''
!****************************************
!Tracking                                **
!****************************************
track,dump,file=linear_machine, deltap = 0;
start, x = 1e-3, px = 0, y = 1e-3, py = 0;
start, x = 1e-1, px = 0, y = 1e-1, py = 0;
run, turns = 100;
endtrack;
value, table(summ, q1);
value, table(summ, q2);'''
myMad.input(myString);
```

If the tracking module runs successfully you should get a summary table like the one in Fig. I.5.20.

```
++++++ table: tracksumm

    number       turn                x               px
         1          0            0.001                0
         2          0              0.1                0
         1        100    0.0009998140232   -3.216288976e-09
         2        100      0.09938085905   -1.072122643e-05

                y               py                t               pt
            0.001                0                0                0
              0.1                0                0                0
      0.00100004125   -3.219486416e-09   -8.278033339e-06                0
       0.1001360287   -1.072882436e-05     -0.08277774821                0

                s                e
                0                0
                0                0
              100                0
              100                0
exit TRACK module

table( summ  q1 )  =       0.2500001233 ;
table( summ  q2 )  =       0.2500001233 ;
```

Fig. I.5.20: Tutorial 4 MAD-X tracking summary table.

When activating the option $dump$ on the $track$ module, new tables are generated by MAD-X with the tracking results for each particle at each observable point. If no observable points are defined in the sequence, only the trajectory data at the start of the lattice is stored. One can check the names of the

output tables using the following command:

```
print(list(myMad.table))
# Output
# ['summ', 'twiss', 'beforematching', 'aftermatching', 'mytracksumm',
   'tracksumm', 'track.obs0001.p0001', 'track.obs0001.p0002']
```

Then, the tracking output data can be saved into $Pandas$ dataframes as:

```
particle1 = myMad.table["track.obs0001.p0001"].dframe()
particle2 = myMad.table["track.obs0001.p0002"].dframe()
```

We can now plot the phase space of the particles at the entrance of the FODO cell using the data from the created $Pandas$ dataframes $particle1$ and $particle2$. In order to see clearly what happens, the pair of coordinates $x - px$ is plotted for the first 5 turns using the code lines below.

```
plt.rcParams["figure.dpi"] = 100
turn0=particle1[particle1["turn"] == 0]
turn1=particle1[particle1["turn"] == 1]
turn2=particle1[particle1["turn"] == 2]
turn3=particle1[particle1["turn"] == 3]
turn4=particle1[particle1["turn"] == 4]
turn5=particle1[particle1["turn"] == 5]
plt.plot(turn0["x"]*1e3,turn0["px"],'xb',markersize=12,label='turn0')
plt.plot(turn1["x"]*1e3,turn1["px"],'xr',markersize=12,label='turn1')
plt.plot(turn2["x"]*1e3,turn2["px"],'xg',markersize=12,label='turn2')
plt.plot(turn3["x"]*1e3,turn3["px"],'xy',markersize=12,label='turn3')
plt.plot(turn4["x"]*1e3,turn4["px"],'ob',label='turn4')
plt.plot(turn5["x"]*1e3,turn5["px"],'or',label='turn5')
plt.xlabel('x [mm]')
plt.ylabel('px [-]')
plt.legend(loc='best')
```

In order to visualize the results for $particle2$ you need to change $particle1$ by $particle2$ in the above code lines. For the vertical phase space you need to take the corresponding data y from the $Pandas$ dataframes.

In Fig. I.5.21, the horizontal phase space coordinates are shown for $particle1$ (left) and $particle2$ (right). The solution of the Hill's equation represents a particle tracing out on an ellipse in the phase space. For a periodic lattice and at a given location, the linear optics functions return to the same value every turn. However, the particle phase space coordinates $x - px$ and $y - py$ do not. They undergo a shift every turn to different parts of the ellipse. The tune is the number of "phase advance oscillation" a particle undergoes in one turn (number of times around the ellipse), the integer part representing completed oscillations and the fractional part indicating the phase ellipse displacement after one turn. In this example, being the tune 0.25, the particles move one quarter of the ellipse each turn and this explains why the phase space coordinates of the tracked particles repeat every 4 turns as seen in Fig. I.5.21. A similar phase space is found for both particles but with different amplitudes and momenta.

In Fig. I.5.22, the vertical phase space coordinates are shown for $particle1$ (left) and $particle2$ (right) for completeness. In the vertical plane, the same behavior as in the horizontal plane is observed.

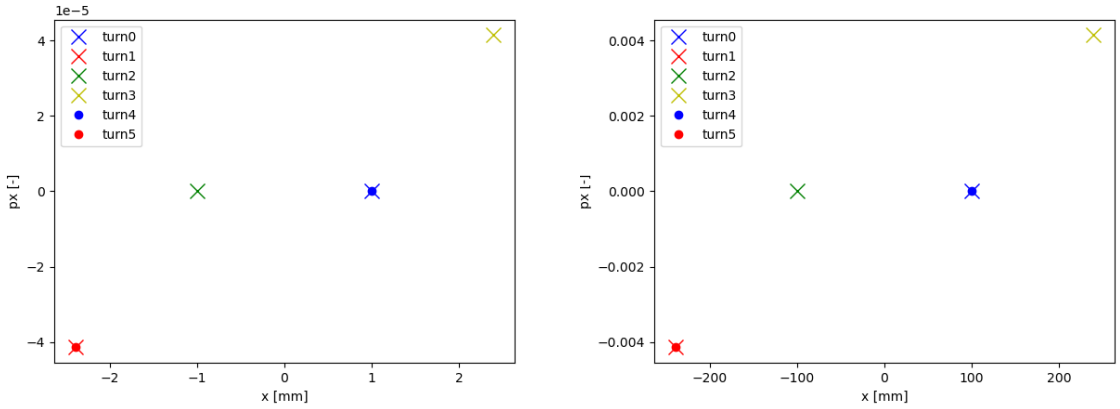

Fig. I.5.21: Horizontal phase space coordinates for $particle1$ (left) and $particle2$ (right) for the first 5 turns of the tracking.

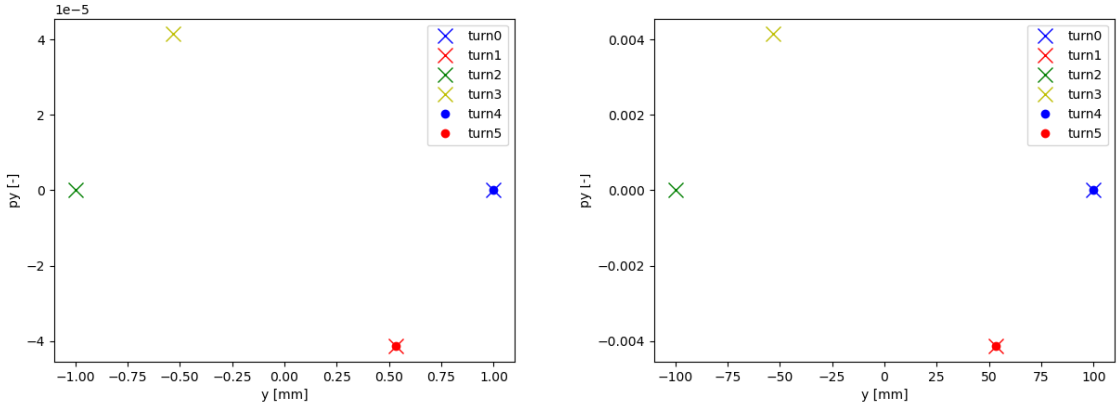

Fig. I.5.22: Vertical phase space coordinates for $particle1$ (left) and $particle2$ (right) for the first 5 turns of the tracking.

In Fig. I.5.23 (left), the horizontal amplitude versus the turn number is depicted for each particle. It is worth mentioning here that with the turn-by-turn amplitude data, the tunes of the machine can be computed by making a Fast Fourier Transform (FFT) analysis. In Fig. I.5.23 (right), the FFT amplitude of the turn-by-turn data is shown for $particle1$ and $particle2$, in blue and red, respectively. In this example, the tune of the machine is 0.25, value at which we observe the FFT peak amplitude. The Python code lines needed to reproduce the plots in Fig. I.5.23 are given below. A similar plot can be obtained for the vertical plane.

```
# Code lines to plot the particles' amplitude versus the s position
plt.plot(particle1['turn'],particle1['x'],'.-b', label='Particle 1')
plt.plot(particle2['turn'],particle2['x'],'.-r', label='Particle 2')
plt.xlabel('Turn')
plt.ylabel('x [m]');
plt.legend(loc='best');
# For the FFT plot
```

```
FFTamp1=np.abs(np.fft.fft(particle1['x']))
FFTamp2=np.abs(np.fft.fft(particle2['x']))
plt.figure()
plt.plot(np.linspace(0,1,len(particle1['x'])), FFTamp1, 'b', label='Particle 1')
plt.plot(np.linspace(0,1,len(particle2['x'])), FFTamp2, 'r', label='Particle 2')
plt.xlabel('Q1 from X')
plt.ylabel('FFT amplitude [arb. units]');
plt.xlim(0,0.5)
plt.legend(loc='best');
# The maximum possible frequency for a given sampling rate that can be
# reconstructed is given by the Nyquist limit = sampling frecuency/2.
# In our case the sampling rate is 1, so the limit is 0.5.
```

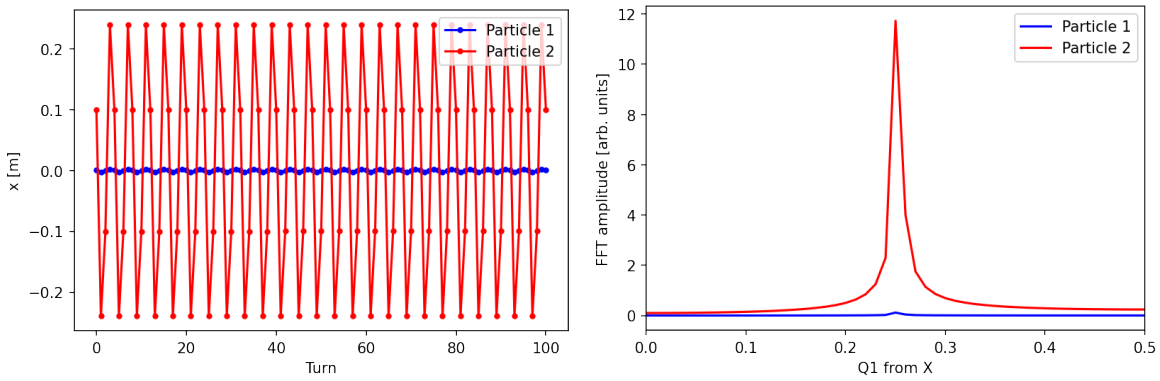

Fig. I.5.23: Horizontal amplitude versus the turn number for $particle1$ in blue and $particle2$ in red (left). FFT amplitude of the turn-by-turn horizontal amplitude data for $particle1$ in blue and $particle2$ in red (right).

Question 3. Now we can repeat the tracking exercise, this time for off-momentum particles, by adding the attribute $deltap = 0.01$ in the $track$ module.

```
myString='''
!*********************************************************
!Tracking                                                **
!*********************************************************
track, dump , file=linear_machine_off_energy, deltap = 1e-2;
start, x = 1e-3, px = 0, y = 1e-3, py = 0;
start, x = 1e-1, px = 0, y = 1e-1, py = 0;
run, turns = 100;
endtrack;'''
myMad.input(myString);
# Saving the data of each particle in a Pandas dataframe
off_momentum_particle1 = myMad.table['track.obs0001.p0001'].dframe()
off_momentum_particle2 = myMad.table['track.obs0001.p0002'].dframe()
# Horizontal phase space plot
plt.rcParams["figure.dpi"] = 100
plt.plot(off_momentum_particle1["x"],off_momentum_particle1["px"],'ob',label = "
    X-phase space")
plt.plot(off_momentum_particle1["y"],off_momentum_particle1["py"],'.r',label = "
```

```
    Y-phase space")
plt.xlabel("x [m]")
plt.ylabel("px")
plt.legend(loc="best")
```

The horizontal (blue) and the vertical (red) phase space coordinates are shown for $particle1$ (left) and $particle2$ (right) in Fig. I.5.24.

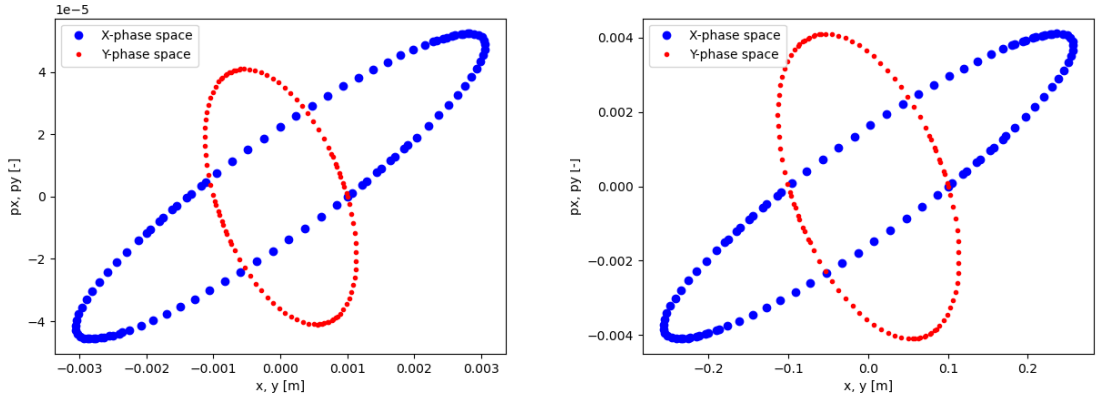

Fig. I.5.24: Horizontal and vertical phase space coordinates for $particle1$ (left) and $particle2$ (right).

As can be seen in Fig. I.5.24, for off-momentum particles, the tune is affected by the natural chromaticity of the FODO cell and it is no longer 0.25. The phase space coordinates of the particles do not repeat now every 4 turns, as it was observed in Fig. I.5.21, but their motion remains stable, and the phase space coordinates describe an ellipse. In this example, the observed chromaticity effect is small, however in machines with stronger quadrupoles, the impact on the beam dynamics could be greater, and the tune spread can lead to unstable motion due to resonances. Because of that, correction schemes are necessary such as the one to be designed in Tutorial 5.

I.5.7.5 Tutorial 5: Non-linearities

Question 1. The goal of this tutorial is to design a chromaticity correction scheme for the FODO cell of Tutorial 4 by using sextupole magnets. Then, by means of tracking studies, the impact of the sextupoles on the particles' beam dynamics is studied.

A pair of sextupole magnets can be used to compensate for the natural chromaticity of a FODO cell. The string below defines the new lattice, in which the two sextupoles have been installed next to the two quadrupoles, along with the beam and the actions to be performed.

```
myString='''
!************************************************************************
!Definition of parameters                                              **
!************************************************************************
l_cell =  100;
quadrupolelength = 5;
dipolelength = 15;
ls = 0.5;
```

```
nBend = 736;
myAngle = 2*pi / nBend;
mykf := 2.8 / l_cell / quadrupolelength;
mykd := 2.8 / l_cell / quadrupolelength;
k2f := 0.0;
k2d := -k2f;
!****************************************************************************
!Definition of magnets                                                      **
!****************************************************************************
qf: quadrupole, l = quadrupolelength, k1 := mykf;
qd: quadrupole, l = quadrupolelength, k1 := -mykd;
bm: sbend, l = dipolelength, angle := myAngle;
sf: sextupole, l = ls, k2 := k2f;
sd: sextupole, l = ls, k2 := k2d;
!****************************************************************************
!Definition of the sequence                                                 **
!****************************************************************************
myCell: sequence, refer = centre, l = l_cell;
qf: qf, at = 0 + qf->l/2;
s1 : sf, at = 5 + sf->l/2;
b1:  bm, at = 10 + bm->l/2;
b2:  bm, at = 30 + bm->l/2;
qd: qd, at = 50 + qd->l/2;
s2 : sd, at = 55 + sd->l/2;
b3:  bm, at = 60 + bm->l/2;
b4:  bm, at = 80 + bm->l/2;
endsequence;
!****************************************************************************
!Definition of beam                                                         **
!****************************************************************************
beam, particle = proton, energy = 7000;
!****************************************************************************
!Activation of the sequence                                                 **
!****************************************************************************
use, sequence = myCell;
!****************************************************************************
!Twiss                                                                      **
!****************************************************************************
twiss, table = thick_sequence;'''
myMad.input(myString)
```

Next, we need to make thin the new lattice with the sextupoles in order to use the tracking module, and we need to rematch the tune of the machine to 0.25 after the *makethin* command, to compare the results with those obtained in Tutorial 4. To accomplish these tasks, the following code lines have to be run:

```
myString='''
!****************************************************************************
!Makethin                                                                   **
!****************************************************************************
makethin, sequence = myCell;
```

```
!*****************************************************************************
!Activate sequence                                                           **
!*****************************************************************************
use, sequence = myCell;
!*****************************************************************************
!Twiss                                                                       **
!*****************************************************************************
twiss, table = thin_sequence;
!*****************************************************************************
!Matching of the tunes                                                       **
!*****************************************************************************
match, sequence = myCell;
global, q1 = 0.25;
global, q2 = 0.25;
vary, name = mykf, step = 0.00001;
vary, name = mykd, step = 0.00001;
lmdif, calls = 100, tolerance = 1e-6;
endmatch;
!*****************************************************************************
!Twiss                                                                       **
!*****************************************************************************
twiss, table = twiss_after_matching;'''
myMad.input(myString)
thinNonZeroChromaSUMMTable = myMad.table["summ"].dframe()
print(thinNonZeroChromaSUMMTable["dq1"])
# Output
# -0.318184
thinNonZeroChromaTWISSTable = myMad.table["twiss_after_matching"].dframe()
```

Next, the strength of the sextupoles required to compensate the chromaticity in the FODO cell is calculated using the matching module as follows:

```
myString='''
!*********************************************************
!Matching the chromaticity.                              **
!*********************************************************
match, sequence = myCell;
global, dq1 = 0.0;
global, dq2 = 0.0;
vary, name = k2f, step = 0.0001;
vary, name = k2d, step = 0.0001;
lmdif, calls = 100, tolerance=1E-12;
endmatch;
!*********************************************************
!Twiss                                                   **
!*********************************************************
select, flag = twiss, column = name,s,betx,bety,dx,k1l,k2l;
twiss, file = "twiss_after_chroma_correction.txt";'''
myMad.input(myString)
thinZeroChromaDFTable = myMad.table["twiss"].dframe()
```

The computed new optics have been saved into a new $Pandas$ dataframe and the summary match-

ing table is shown in Fig.I.5.25.

```
MATCH SUMMARY

Node_Name                   Constraint   Type  Target Value       Final Value        Penalty
-------------------------------------------------------------------------------------------
Global constraint:          dq1          4     0.00000000E+00    -1.51076348E-14     2.28240629E-28
Global constraint:          dq2          4     0.00000000E+00    -2.98265351E-14     8.89622199E-28


Final Penalty Function =   1.11786283e-27



Variable                 Final Value  Initial Value Lower Limit  Upper Limit
-------------------------------------------------------------------------------
k2f                       2.86234e-02  0.00000e+00 -1.00000e+20  1.00000e+20
k2d                      -5.54390e-02  0.00000e+00 -1.00000e+20  1.00000e+20

END MATCH SUMMARY
```

Fig. I.5.25: Tutorial 5 MAD-X matching summary table.

Question 2. The contribution to the chromaticity by the sextupoles can be computed using the results from the previous $twiss$ and Eq. (I.5.6) with the following code lines:

```
aux = thinZeroChromaDFTable
(aux["betx"]*aux["dx"]*1*aux['k2l']).sum()/4./np.pi
# Output: 0.3182143327879991
```

The previous obtained value has to be compared with the horizontal tune value before compensation printed below:

```
thinNonZeroChromaSUMMTable["dq1"]
# Output
# -0.318214
```

A very good compensation is found. The same verification can be done for the vertical plane.

Question 3. Now, we are going to perform tracking studies for the new lattice with sextupoles to evaluate the impact of the non-linear elements installed in the FODO cell on the beam dynamics of the particles. The same tracking module used in Tutorial 4 should be used here to track both particles ($particle1$ and $particle2$), but now with an off-momentum $\Delta p = 10^-2$ as described below.

```
myString='''
!************************************************
!Tracking                                        **
!************************************************
track, dump, file = withSextupoles, deltap = 1e-2;
start, x = 1e-3, px = 0, y = 1e-3, py = 0;
start, x = 1e-1, px = 0, y = 1e-1, py = 0;
run, turns = 100;
endtrack;'''
myMad.input(myString);
```

```
# Saving the output data in Pandas dataframes
off_momentum_nl_particle1=myMad.table["track.obs0001.p0001"].dframe()
off_momentum_nl_particle2=myMad.table["track.obs0001.p0002"].dframe()
```

The phase space coordinates for both, horizontal (blue) and vertical (red) planes, are shown in Fig. I.5.26 for $particle1$ (left) and $particle2$ (right). Figure I.5.27 displays the horizontal (left) and vertical (right) amplitudes for both particles, serving to illustrate the evolution of the amplitude of the particles in the new lattice turn-after-turn.

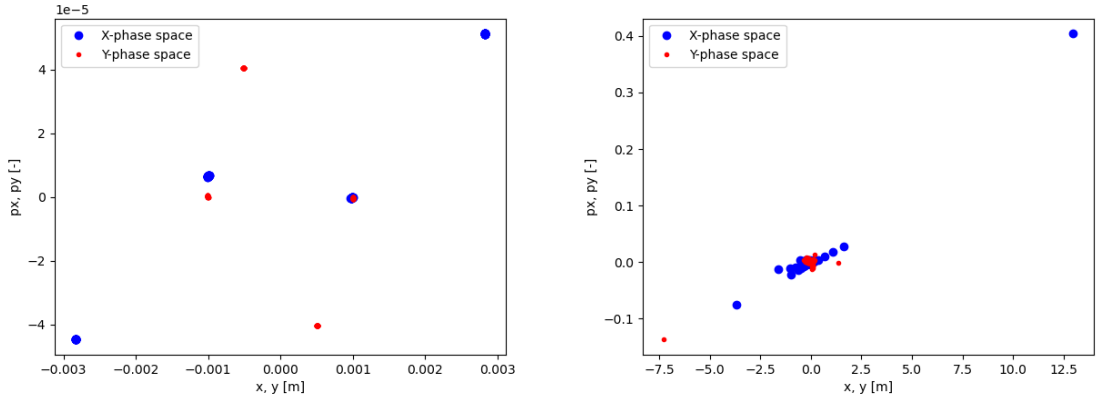

Fig. I.5.26: Horizontal (blue) and vertical (red) phase space coordinates for $particle1$ (left) and $particle2$ (right).

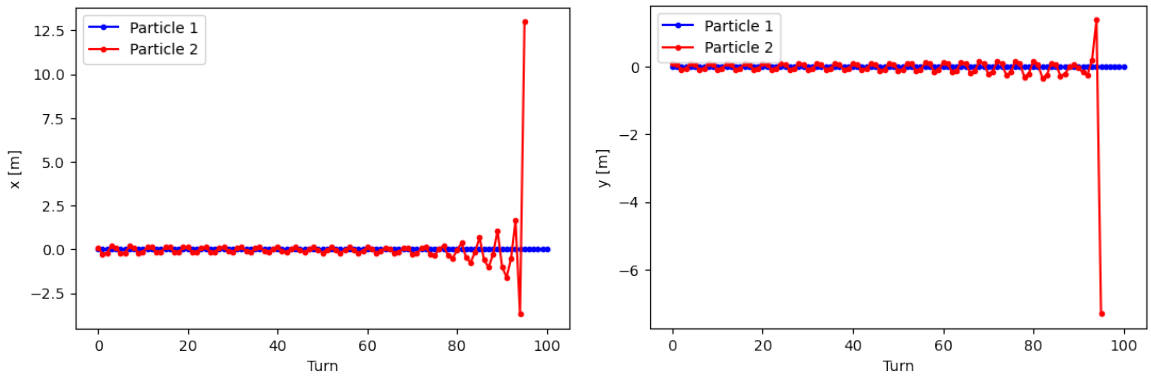

Fig. I.5.27: Horizontal (left) and vertical (right) amplitude versus the turn number for $particle1$ (blue) and $particle2$ (red).

As can be seen in Fig. I.5.26 (right) and Fig. I.5.27, for large initial amplitude excursion, the particle motion loses its stability. The cost of increasing the energy acceptance by introducing non-linear elements in the beamline is a reduction of the transverse acceptance. The code lines used to plot Fig. I.5.26 and Fig. I.5.27 are presented below.

```
# Phase space plot
plt.figure()
plt.rcParams["figure.dpi"] = 100
```

```
plt.plot(off_momentum_nl_particle1["x"],off_momentum_nl_particle1["px"],'ob',
    label = "X-phase space")
plt.plot(off_momentum_nl_particle1["y"],off_momentum_nl_particle1["py"],'.r',
    label = "Y-phase space")
plt.xlabel("x, y [m]")
plt.ylabel("px, py [-]")
plt.legend(loc="best")
plt.figure()
plt.rcParams["figure.dpi"] = 100
plt.plot(off_momentum_nl_particle2["x"],off_momentum_nl_particle2["px"],'ob',
    label = "X-phase space")
plt.plot(off_momentum_nl_particle2["y"],off_momentum_nl_particle2["py"],'.r',
    label = "Y-phase space")
plt.xlabel("x, y [m]")
plt.ylabel("px, py [-]")
plt.legend(loc="best")

# Amplitude versus turn number plot
plt.figure()
plt.plot(off_momentum_nl_particle1['turn'],off_momentum_nl_particle1['x'],'.-b',
     label='Particle 1')
plt.plot(off_momentum_nl_particle2['turn'],off_momentum_nl_particle2['x'],'.-r',
     label='Particle 2')
plt.xlabel('Turn')
plt.ylabel('x [m]');
plt.legend(loc='best');
plt.figure()
plt.plot(off_momentum_nl_particle1['turn'],off_momentum_nl_particle1['y'],'.-b',
     label='Particle 1')
plt.plot(off_momentum_nl_particle2['turn'],off_momentum_nl_particle2['y'],'.-r',
     label='Particle 2')
plt.xlabel('Turn')
plt.ylabel('y [m]');
plt.legend(loc='best');
```

Question 4. In the final question of this tutorial we are going to adjust the tune of the machine to 0.23 in both the horizontal and the vertical planes, and repeat the chromaticity correction matching and the tracking studies using the code lines below.

```
myString='''
!************************************************************************
!Matching the tunes                                                     **
!************************************************************************
match, sequence = myCell;
global, q1 = 0.23;
global, q2 = 0.23;
vary, name = mykf, step = 0.00001;
vary, name = mykd, step = 0.00001;
lmdif, calls = 100, tolerance = 1e-6;
endmatch;
!************************************************************************
```

```
!Matching the chromaticity                                                    **
!*****************************************************************************
match, sequence = myCell;
global, dq1 = 0.0;
global, dq2 = 0.0;
vary, name = k2f, step = 0.0001;
vary, name = k2d, step = 0.0001;
lmdif, calls = 100, tolerance = 1e-12;
endmatch;
!*****************************************************************************
!Tracking                                                                     **
!*****************************************************************************
track, dump, file = new, deltap = 1e-2;
start, x = 1e-3, px = 0, y = 1e-3, py = 0;
start, x = 1e-1, px = 0, y = 1e-1, py = 0;
run, turns = 100;
endtrack;'''
myMad.input(myString);
# Saving the data in Pandas dataframes
optWP_offmomentum_nl_particle1 = myMad.table["track.obs0001.p0001"].dframe()
optWP_offmomentum_nl_particle2 = myMad.table["track.obs0001.p0002"].dframe()
```

Figure I.5.28 shows the new results for the horizontal and the vertical phase space coordinates for $particle1$ (left) and $particle2$ (right). In Fig. I.5.29, the resulting horizontal (left) and vertical (right) amplitude versus the turn number are shown for both particles. Now, both particles are stable. To generate these plots, you can use the code lines employed in the previous questions but taking the data from the new $Pandas$ dataframes.

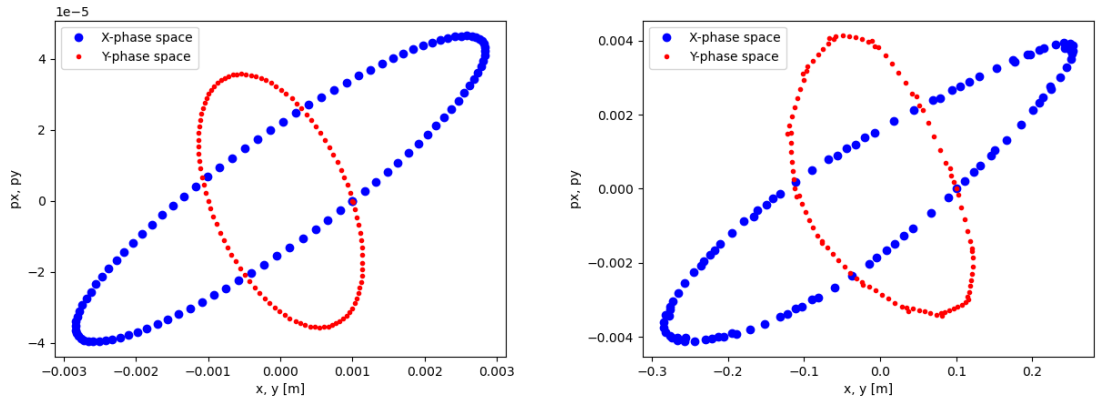

Fig. I.5.28: Horizontal (blue) and vertical (red) phase space coordinates for $particle1$ (left) and $particle2$ (right).

In conclusion, sextupole magnets play a crucial role for chromaticity correction, but at the expense of introducing non-linear fields into the lattice, resulting in non-linear particle motion. This eventually leads to particle instability and loss for large amplitudes. The maximum particle amplitude that can be effectively tracked in the machine defines the dynamic aperture and as demonstrated, the dynamic

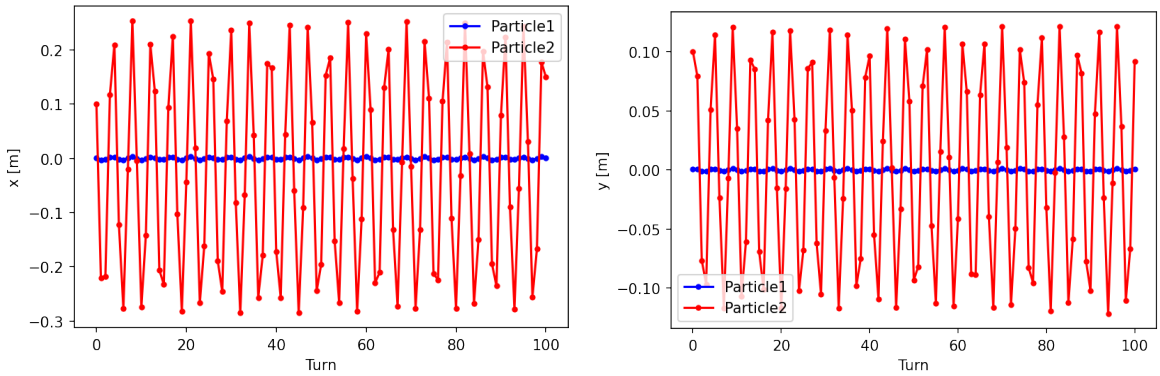

Fig. I.5.29: Horizontal (left) and vertical (right) amplitude versus turn number for $particle1$ (blue) and $particle2$ (red).

aperture can be enhanced by selecting an optimal working point of the tune that is far from resonances.

I.5.7.6 Tutorial 6: Building a transfer line

Question 1. The objective of this tutorial is to design a transfer line and to understand the main differences from designing a periodic lattice. Below is the string defining the proposed lattice along with the beam and the sequence.

```
myString='''
!****************************************
!Definition of parameters                **
!****************************************
quadrupolelength = 0.1;
l_cell = 10;
myk1 = 0.1;
myk2 = 0.1;
myk3 = 0.1;
myk4 = 0.1;
!****************************************
! Definition of magnets                  **
!****************************************
q: quadrupole, l = quadrupolelength;
!****************************************
!Definition of sequence                  **
!****************************************
myCell: sequence, refer = centre, l = l_cell;
myStart: marker, at = 0;
q1: q, k1 := myk1, at = 2;
q2: q, k1 := myk2, at = 4;
q3: q, k1 := myk3, at = 6;
q4: q, k1 := myk4, at = 8;
myEnd: marker, at = 10;
endsequence;
!****************************************
!Definition of beam                     **
```

```
!*****************************************
beam, particle = proton, energy = 2;
!*****************************************
!Activation of the sequence                **
!*****************************************
use, sequence = myCell;'''
myMad.input(myString)
```

Next, we are trying to find the periodic solution of the linear optics functions using the $twiss$ command.

```
myString='''
!*****************************************
! Twiss                                    **
!*****************************************
twiss;'''
myMad.input(myString)
```

When executing the $twiss$ command, MAD-X outputs an error message, as depicted in Fig. I.5.30. A periodic solution is unattainable for a magnetic lattice consisting only of focusing quadrupoles.

```
enter Twiss module

iteration:   1 error:   0.000000E+00 deltap:   0.000000E+00
orbit:   0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00  0.0000
00E+00  0.000000E+00
++++++ warning: TWCLORB:  Vertical plane might be unstable More informa
tion with the debug flag on.
++++++ warning: TWCPIN:  Mode 2 is unstable for delta(p)/p =     0.0000
00: cosmux =     0.805975, cosmuy =     1.206091
++++++ warning: Twiss failed:  MAD-X continues
```

Fig. I.5.30: Tutorial 6 MAD-X output error message.

Question 2. Instead of a periodic solution, an initial condition (IC) solution can be explored. To do so, we must add the values of the linear optics functions at the entrance of the lattice as follows:

```
myString='''
twiss, betx = 1, bety = 2;'''
myMad.input(myString)
# Saving the output TWISS data in a Pandas dataframe
myDFTable1 = myMad.table["twiss"].dframe()
# Pandas dataframe with only the optics functions at the end of the beamline
optics_at_end = myDFTable1.iloc[[-1]]
optics_at_end = optics_at_end[["name","keyword","betx","bety","alfx","alfy"]]
display(optics_at_end)
```

The $Pandas$ dataframe table showing the linear optics functions at the end of the transfer line is illustrated in Fig. I.5.31. In Fig. I.5.32, the horizontal (blue) and vertical (red) β-functions and the horizontal dispersion, D_x (green), are depicted as a function of the position s. In addition, the layout of the transfer line is overlaid on the top of the figure, including the strength of the magnets. Note, that the dispersion is zero for this lattice, as no errors and dipole magnets have been considered. To generate this figure, the Python function provided in Appendix 1.A is used as it is illustrated in Tutorial 1.

	name	keyword	betx	bety	alfx	alfy
#e	mycell$end:1	marker	85.599525	61.413366	-7.397891	-6.624547

Fig. I.5.31: *Pandas* dataframe table with the linear optics functions at the end of the transfer line.

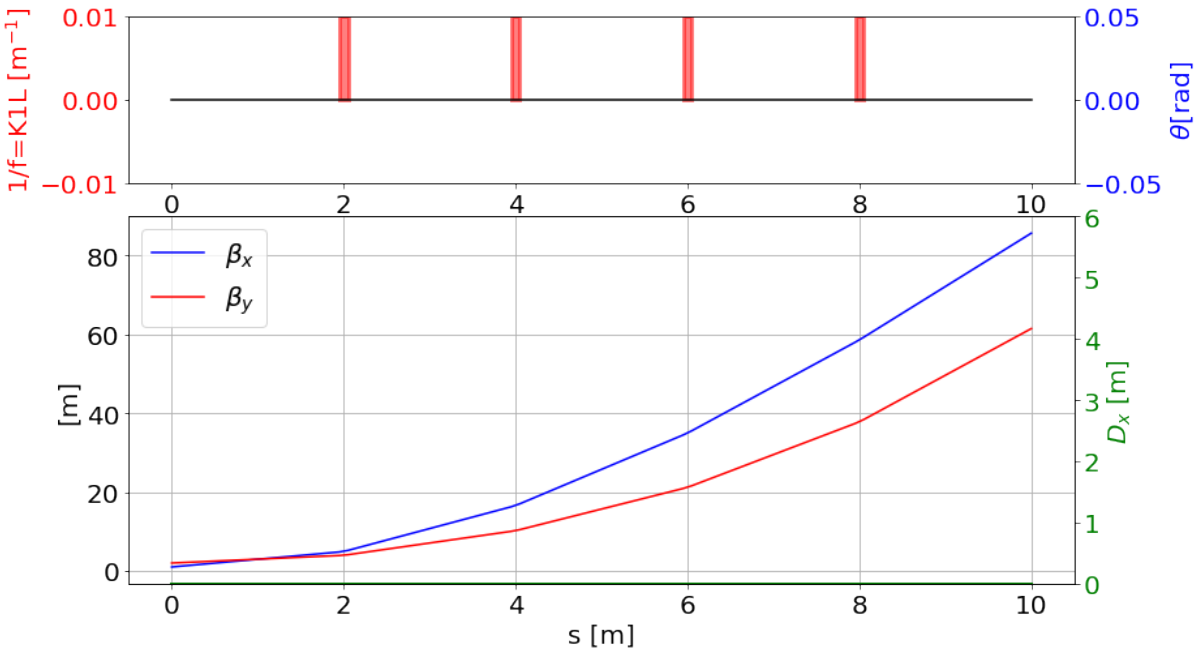

Fig. I.5.32: Horizontal (blue) and vertical (red) β-functions and horizontal dispersion (green) as a function of the position s along the transfer line. In addition, the layout of the transfer line is added on the top of the figure including the strength of the magnets.

Question 3. Next, we want to match the strength of the quadrupoles to obtain the following optical functions at the end of the transfer line: (β_x^{end1}, α_x^{end1}, β_y^{end1}, α_y^{end1}) = (2 m, 0, 1 m, 0). To accomplish this, we use the *match* block as follows:

```
myString='''
myk1 = 0.1;
myk2 = 0.1;
myk3 = 0.1;
myk4 = 0.1;
match, sequence = myCell, betx = 1, bety = 2;
constraint, betx = 2, range = #e;
constraint, alfx = 0, range = #e;
constraint, bety = 1, range = #e;
constraint, alfy = 0, range = #e;
vary, name = myk1, step = 0.00001;
vary, name = myk2, step = 0.00001;
vary, name = myk3, step = 0.00001;
vary, name = myk4, step = 0.00001;
```

```
jacobian, calls = 50, tolerance = 1e-20;
endmatch;
twiss, betx = 1, bety = 2, file=AfterMatching1.txt;'''
myMad.input(myString);
# Saving the output data in a Pandas dataframe
myDFTable2 = myMad.table["twiss"].dframe()
myDFTable2
```

In Fig. I.5.33 the main parameters of the quadrupole magnets after the first matching are depicted. In order to print the table (as in Fig. I.5.33) you should employ the code lines below.

```
aux2 = myDFTable2[myDFTable2['keyword']=='quadrupole']
aux2 = aux2[["name","keyword","s","k1l","l","betx","bety"]]
display (aux2)
```

In Fig. I.5.34, the horizontal and vertical β-functions, along with the horizontal dispersion D_x after the first matching, are shown.

	name	keyword	s	k1l	l	betx	bety
q1	q1:1	quadrupole	2.05	-0.067697	0.1	5.236870	4.074009
q2	q2:1	quadrupole	4.05	0.654514	0.1	18.765565	9.169430
q3	q3:1	quadrupole	6.05	-0.680251	0.1	0.989436	58.865109
q4	q4:1	quadrupole	8.05	0.824398	0.1	3.901250	4.802500

Fig. I.5.33: Quadrupoles' main parameters after the first matching.

Question 4. Next, we are performing a second local matching by using as a starting point, the gradients obtained in question 3 to get the initial optics at the end of the transfer line (see Table I.5.31) from question 2. In summary, we aim to match the end optics obtained in the second question but using a different starting point for the quadrupoles' strength.

```
myString='''
myK1 = -0.676969;
myK2 = 6.54514;
myK3 = -6.80251;
myK4 = 8.24398;
match, sequence = myCell, betx = 1, bety = 2;
constraint, betx = 85.599525, range = #e;
constraint, alfx = -7.397891, range = #e;
constraint, bety = 61.413366, range = #e;
constraint, alfy = -6.624547, range = #e;
vary, name = myk1, step = 0.00001;
vary, name = myk2, step = 0.00001;
vary, name = myk3, step = 0.00001;
vary, name = myk4, step = 0.00001;
jacobian, calls = 50, tolerance = 1e-20;
endmatch;
```

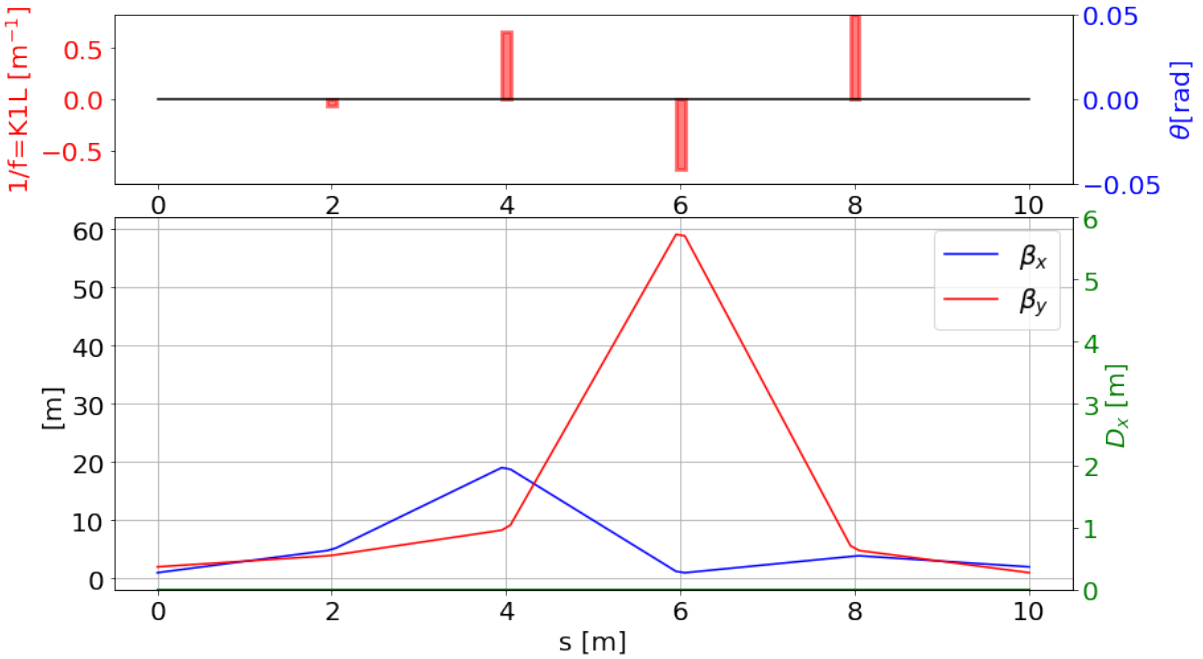

Fig. I.5.34: Horizontal (blue) and vertical (red) β-functions and horizontal dispersion D_x (green) as a function of the position s. The lattice layout is also included on the top figure as well as the strength of the magnets after the first matching.

```
twiss, betx = 1, bety = 2;'''
myMad.input(myString);
# Saving the twiss output data in a Pandas dataframe
myDFTable3 = myMad.table["twiss"].dframe()
myDFTable3
```

A summary of the main quadrupoles' parameters, resulting from the second matching, is presented in Fig. I.5.35. Additionally, Fig. I.5.36 shows the new horizontal and vertical β-functions and the horizontal dispersion D_x.

	name	keyword	s	k1l	l	betx	bety
q1	q1:1	quadrupole	2.05	-1.257661	0.1	5.866198	3.614683
q2	q2:1	quadrupole	4.05	0.864066	0.1	91.715371	5.583809
q3	q3:1	quadrupole	6.05	-0.588784	0.1	0.272915	97.370215
q4	q4:1	quadrupole	8.05	0.442574	0.1	59.223333	38.356731

Fig. I.5.35: Quadrupoles' main parameters after the second matching.

As can be seen from the previous results, the second matching calculation does not converge to the initial solution obtained in the second question. In a transfer line, it is important to note that there can be multiple solutions. Furthermore, the solution obtained in the second matching appears to be highly

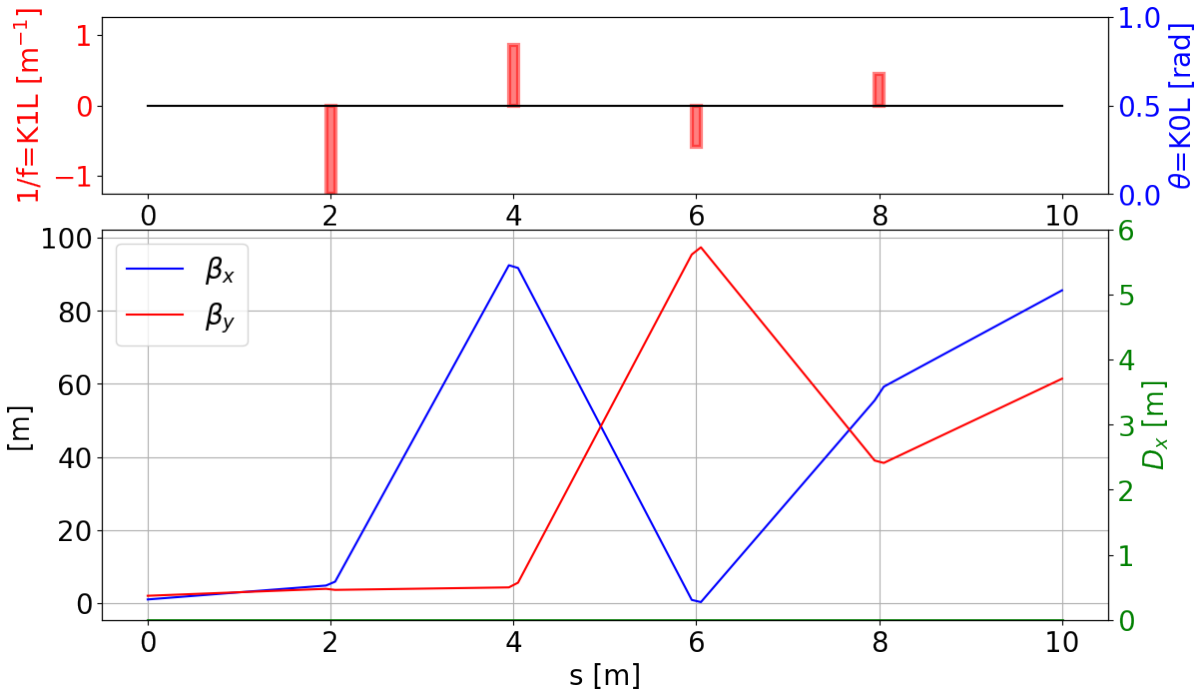

Fig. I.5.36: Horizontal (blue) and vertical (red) β-functions and horizontal dispersion D_x (green) as a function of the position s. The lattice layout is also included on the top of the figure as well as the strength of the magnets after the second matching.

suboptimal, requiring the use of stronger quadrupole magnets (see the values in Fig. I.5.36 compared to the values in Fig. I.5.34).

Question 5. Last, we compute the magnetic field corresponding to the gradients obtained initially (question 2) and after the second matching (question 4) using the magnet's properties described in the statement of this tutorial.

```
# For the initial conditions
aux = myDFTable1[myDFTable1["keyword"]=="quadrupole"]
aux = aux["k1l"]/aux["l"]
# Excitation current
aux=aux*10
# Excitation magnetic factor
aux=aux*2
# Aperture diameter
aux=aux*.04
# Final magnetic field in T
print(np.abs(aux))
# Output
# q1    0.08
# q2    0.08
# q3    0.08
# q4    0.08
```

```
17
18 # For the second matching
19 aux = myDFTable3[myDFTable3["keyword"] == "quadrupole"]
20 aux = aux["k1l"]/aux["l"]
21 # Excitation current
22 aux = aux*10
23 # Excitation magnetic factor
24 aux = aux*2
25 # Aperture diameter
26 aux = aux*.04
27 # Final magnetic field in T
28 print(np.abs(aux))
29 # Output
30 # q1    10.061285
31 # q2     6.912524
32 # q3     4.710268
33 # q4     3.540589
```

As can be seen, the second magnets are extremely difficult to build (superconductive) due to their high magnetic field. For a 2 GeV machine, it would better to stick with normal conducting magnets. However, in some cases such as the Large Hadron Collider, which has to handle 7 TeV proton beams, superconductive magnets are the only valid approach.

I.5.8 Acknowledgments

I would like to express my gratitude to Dr. G. Sterbini, who taught MAD-X at JUAS for over a decade, for his support in preparing this course over the past years and for proofreading this document. I have had the privilege of inheriting a large wealth of knowledge and course content from him. I am also deeply thankful to P. Martínez-Reviriego, E. Martínez-López, P. Martín-Luna and L. Karina-Pedraza for proofreading this document and providing valuable feedback.

Appendix

I.5.A Python script for linear optics functions plot with machine layout on the top

The following python script can be used to plot the β-functions and the horizontal dispersion D_x of a given lattice with the layout depicted in the top figure as well as the strength of the magnets. Quadrupoles and dipoles are considered in this function but the code can be expanded to take into account also higher order magnets.

```
#!/usr/bin/env python3
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib.patches as patches

def plotLatticeSeries(ax,series, height=1., v_offset=0., color='g',alpha=0.5,lw=3):
    aux=series
    ax.add_patch(
    patches.Rectangle((aux.s-aux.l, v_offset-height/2.),    # (x,y)
                      aux.l,                                # width
                      height,                               # height
                      color=color, alpha=alpha,lw=lw))
    return;


def plot_layout(myTwiss):

    fig = plt.figure(figsize=(13,8))

    ax1=plt.subplot2grid((3,3), (0,0), colspan=3, rowspan=1)
    plt.plot(myTwiss['s'],0*myTwiss['s'],'k')
    DF=myTwiss[(myTwiss['keyword']=='quadrupole')]
    print(DF)
    for i in range(len(DF)):
        aux=DF.iloc[i]
        print(aux.k1l)
        plotLatticeSeries(plt.gca(),aux, height=aux.k1l, v_offset=aux.k1l/2, color='r')
    color = 'red'
    ax1.set_ylabel('1/f=K1L [m$^{-1}$]', color=color,fontsize=20)
    ax1.tick_params(axis='y',labelsize=20,labelcolor=color)
    ax1.tick_params(axis='x',labelsize=20)
    ax1.set_ylim(-np.max(abs(myTwiss.k1l)),np.max(abs(myTwiss.k1l)))

    ax2 = ax1.twinx()
    color = 'blue'
    ax2.set_ylabel('$\\theta$[rad]', color=color,fontsize=20)
    ax2.tick_params(axis='y', labelsize=20,labelcolor=color)
    DF=myTwiss[(myTwiss['keyword']=='sbend')]
    for i in range(len(DF)):
```

```
        aux=DF.iloc[i]
        plotLatticeSeries(plt.gca(),aux, height=aux.angle, v_offset=aux.angle/2,
    color='b')
    ax2.set_ylim(-np.max(abs(myTwiss.angle)),np.max(abs(myTwiss.angle)))


    axbeta=plt.subplot2grid((3,3), (1,0), colspan=3, rowspan=2,sharex=ax1)
    plt.plot(myTwiss['s'],myTwiss['betx'],'b', label='$\\beta_x$')
    plt.plot(myTwiss['s'],myTwiss['bety'],'r', label='$\\beta_y$')
    plt.legend(loc='best',fontsize=20)
    plt.ylabel('[m]',fontsize=20)
    plt.xlabel('s [m]',fontsize=20)
    axbeta.tick_params(axis='both', labelsize=20)
    plt.grid()

    ax3 = plt.gca().twinx()
    plt.plot(myTwiss['s'],myTwiss['dx'],'green', label='$D_x$', lw=2)
    ax3.set_ylabel('$D_x$ [m]', color='green',fontsize=20)
    ax3.tick_params(axis='y', labelsize=20,labelcolor='green')
    ax3.tick_params(axis='x', labelsize=20)
    ax3.set_ylim(0, 6);

    return
```

The main function is called $plot\_layout$ and needs as input to work a $Pandas$ dataframe with the $keyword$, the s location, the strength of the magnets, the β-function and the horizontal dispersion data for all the elements in the beamline. In order to run it, you can copy and paste the code in the Python interface where the tutorials are being solved or you can save the script given below as a Python file and run it as follows:

```
import sys
sys.path.append('path to the location where the script is saved')
import 'name of the script' as lib
lib.plot_layout(myDF)
```

where $myDF$ is the $Pandas$ dataframe containing the data to be plotted.

References

[1] MAD-X web site, last accessed 31 October 2022.

[2] MAD-X physics manual.

[3] MAD-X github repository.

[4] Large Electron-Positron (LEP) collider.

[5] Large Hadron Collider (LHC).

[6] International Linear Collider (ILC) project.

[7] The Compact Linear Collider (CLIC) project.

[8] MAD-X online manual.

[9] E. Forest *et al.*, Introduction to the Polymorphic Tracking Code, 4 Jul. 2002.

[10] Python web site, last accessed 31 October 2022.

[11] Matlab programming and numeric computing platform.

[12] Root open-source data analysis framework.

[13] Gnuplot portable command-line driven graphing utility.

[14] Python for Beginners - Learn Python in 1 Hour.

[15] Learn Python - Full Course for Beginners.

[16] T. Gläßle, Y.I. Levinsen and K. Fuchsberger, Cpymad library documentation, last accessed 31 October 2022.

[17] Anaconda Python distribution.

[18] Numpy Python library.

[19] Matplotlib Python library.

[20] Pandas Python library.

[21] Jupyter Python web-based interactive computing platform.